

POLITECNICO DI MILANO
School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering
Department of Electronics, Information and Bioengineering



Integration of DNA variation data into a GDM repository and API development for identification of genomic populations

Laboratory of Data Science and Bioinformatics

Supervisor: Prof. Marco MASSEROLI
Co-advisors: Anna BERNASCONI, MSc
Arif ÇANAKOĞLU, PhD

Master Thesis of:
Tommaso Alfonsi, Student ID 883772

Academic Year 2018-2019

Abstract

English version The technology advancements in the sequencing of genetic material occurred during the last 20 years lead to an impressive amount of genomic data available. Inferring notions from the captured data, however, is still a challenging task, which requires powerful tools other than strong domain knowledge. In this context, the bioinformatics community aims to implement technologies capable of sustaining the research and enabling the so-called tertiary analysis, i.e. the interpretation of the genomic sequences and the evaluation of the clinical relevance of genomic features. In this thesis, we explain how we support these challenges, in the first place by integrating the 1000 Genomes Project into the GenoMetric Query Language (GMQL) system, and then by developing a platform capable of aggregating such enormous quantity of information to provide immediate insights on human diversity and susceptibility to various pathologies. The result of this work allows expert users to exploit the powerful capabilities of the GMQL to answer complex queries over the 1000 Genomes Project data together with other genetic catalogues already available in the GMQL repository. Also, we made available the 1000 Genomes Project variation data and The Cancer Genome Atlas Program somatic mutation datasets through our novel Data Summarization API, a simple platform to query whole-genome data, to describe a population from a genetic perspective and to collect relevant statistics in a privacy-sensitive way.

Italian version Negli ultimi 20 anni, il progresso della tecnologia di sequenziamento del genoma ha portato ad un aumento straordinario della quantità di dati genomici disponibili per la ricerca. Tuttavia, lo studio di tali dati e l'estrazione di nuove conoscenze utili da essi è ancora un processo difficile che richiede una approfondita conoscenza del settore e strumenti efficaci. In tale contesto, lo scopo della comunità bioinformatica è quello di sviluppare soluzioni capaci di supportare efficacemente la ricerca e l'analisi terziaria di dati genomici, ovvero la comprensione dei processi biologi-

ci ottenibile dall'unione di informazioni eterogenee. Un obiettivo, questo, che passa anche attraverso l'integrazione di grandi banche dati genomiche. Dunque, l'obiettivo di questa tesi è quello di integrare i dati di mutazione genetica prodotti dal 1000 Genomes Project all'interno del sistema di elaborazione GenoMetric Query Language, ed, al tempo stesso, di sviluppare un software (Data-Summarization-API), capace di sfruttare questa grande quantità di dati per fornire statistiche utili allo studio della diversità e alla predisposizione verso varie patologie. Questo lavoro renderà possibile l'utilizzo delle mutazioni genetiche del 1000 Genomes Project per rispondere a domande complesse, utilizzando le potenzialità di GMQL su un dataset integrato contenente ulteriori sette sorgenti di dati. Inoltre, utilizzeremo quelle stesse mutazioni all'interno del software da noi sviluppato, assieme ai dati ottenuti dal The Cancer Genome Atlas Program, per fornire statistiche in grado di descrivere accuratamente il quadro genetico di una popolazione, pur rispettando i criteri di privacy imposti per l'utilizzo di questo tipo di informazioni.

Estratto

La scoperta della tecnologia Next Generation Sequencing (NGS) ha permesso di ridurre enormemente i tempi ed i costi prima necessari per il sequenziamento di materiale genetico. Ciò ha prodotto la rapida crescita del numero dei progetti di ricerca, nonché un altrettanto veloce aumento della quantità dei dati disponibili. Tuttavia, l'ottenimento di nuova conoscenza utile a partire dai dati è un processo ancora particolarmente complesso, che richiede di integrare informazioni relative a fenomeni ed entità eterogenee. Ciò è reso ancora più difficile dalla mancanza di strumenti adeguati a trattare enormi volumi di dati e permettere la loro analisi integrata. Uno dei principali obiettivi della comunità informatica è infatti quello di realizzare strumenti e soluzioni per permettere l'analisi terziaria dei dati genomici, ovvero la ricerca delle cause e la comprensione dei fenomeni che interessano la vita cellulare attraverso l'unione di banche dati che descrivono molteplici caratteristiche del genoma. Si tratta di un obiettivo importante e ambizioso, specialmente alla luce dell'attuale scenario in cui i dati sono prodotti da laboratori sparsi in tutto il mondo, e che operano in maniera totalmente indipendente o quasi. Pertanto, alla difficoltà di confrontare entità genomiche diverse (come sequenze di RNA, mutazioni, dati epigenomici, proteomici, etc.), si aggiunge quella di riconciliare tra loro entità simili ma distinte per formato, qualità e completezza.

Lo scopo che questa tesi si prefigge è perciò quello di contribuire al raggiungimento di tale obiettivo, sia aumentando l'integrazione fra banche dati genomiche, sia realizzando uno strumento software utile all'ottenimento di metriche descrittive del DNA di una popolazione. A questo scopo abbiamo utilizzato il dataset prodotto dal 1000 Genomes Project, uno studio nato nel 2008 con lo scopo di sequenziare il genoma di 1000 individui. L'obiettivo originario fu poi ampiamente superato, ed il progetto si concluse nel 2015 con il raggiungimento di oltre 2500 DNA campionati e rappresentativi di 26 popolazioni di tutto il mondo, realizzando così il più grande dataset pubblico di variazioni genetiche ad oggi disponibile.

La prima parte della tesi descrive l'integrazione di questa importante collezione all'interno di un repository di dati genomici contenente altre sette sorgenti. Tale processo è stato ottenuto attraverso lo sviluppo di moduli software specifici per la sorgente, in grado di catturare i risultati del 1000 Genomes Project e trasformarli in un formato omogeneo e adatto a rappresentare varie entità genomiche. I dati così trasformati sono stati processati ulteriormente per facilitarne il confronto con le altre sorgenti, ed infine sono stati resi disponibili all'interno del sistema GenoMetric Query Language (GMQL), creato dal team di Genomic Computing del Politecnico di Milano.

Studiando i dati ottenuti dal 1000 Genomes Project abbiamo poi ottenuto un modello di data warehousing estensibile a collezioni di mutazioni genetiche diverse. Tale modello è alla base di Data-Summarization-API, una nuova interfaccia realizzata per permettere l'utilizzo di dati di mutazione genetica, allo scopo di descrivere e confrontare popolazioni le cui caratteristiche possono essere definite dall'utente, sia attraverso filtri sui dati degli individui che ne fanno parte, sia sui dati dei loro DNA, realizzati con il più alto livello di dettaglio possibile. Quindi, nella seconda parte della tesi, si descrivono gli aspetti più rilevanti del software realizzato, mostrando una visione di alto livello sull'architettura e, soprattutto, le considerazioni svolte in materia di usabilità, estensibilità e tutela della privacy che hanno guidato la fase progettuale e hanno contribuito a definire le caratteristiche del software da sviluppare. Tali considerazioni hanno portato alla definizione di una API facilmente integrabile con gli strumenti già in uso da ricercatori e bioinformatici, che può facilmente essere estesa con nuove e future banche dati, e che permette di applicare restrizioni specifiche per ogni sorgente, ad esempio per prevenire il rischio di discriminazione etnica o di identificazione degli individui analizzati. Infine, la tesi documenta le proprietà di scalabilità e versatilità dell'architettura attraverso l'estensione dell'API con le mutazioni somatiche del The Cancer Genome Atlas Program, catalogate in oltre 6000 pazienti affetti da cancro.

Il risultato di questa tesi è perciò la possibilità di eseguire analisi approfondite sfruttando le potenzialità offerte da GMQL sui dati del 1000 Genomes Project uniti ad altre sette sorgenti genomiche importanti (ad esempio ENCODE, TCGA, Roadmap Epigenomics), ed al tempo stesso, la realizzazione di un software specifico per la descrizione di popolazioni genomiche usabile e capace di rispondere a domande nuove e complesse, utilizzabile per studi di associazione, diversità genetica ed efficacia del trattamento di patologie rare.

Ringraziamenti

Un sentito ringraziamento al Prof. Marco Masseroli per avermi introdotto al mondo della bioinformatica, una disciplina affascinante che ho scoperto per caso, ma che sono certo continuerò ad approfondire con sempre maggiore interesse. Un immenso grazie anche ai collaboratori Arif Çanakoglu e Anna Bernasconi con i quali ho condiviso la passione per questa materia, nonché dubbi, gioie e dolori di questa tesi. Grazie anche a Laura Riva per la sua disponibilità ed il contributo che ha saputo fornire. Ringrazio ancora tutti loro per non aver fatto mai mancare il loro sostegno, nonostante il periodo certamente difficile e complicato che stiamo vivendo.

Infine, ringrazio infinitamente la mia famiglia, che mi ha accompagnato durante tutto questo percorso ed ha saputo sempre indicarmi la via giusta. Grazie per avermi permesso di raggiungere questo prezioso traguardo.

Contents

Abstract	I
Estratto	V
Ringraziamenti	IX
1 Introduction	1
2 Background	5
2.1 Genomic Data Model	5
2.2 Genomic Conceptual Model	6
2.3 GenoMetric Query Language	6
2.4 Metadata-Manager	8
3 State of the art	11
3.1 gnomAD	11
3.2 Ensembl analysis tools	13
4 Thesis Goals	15
5 1000 Genomes Project data source	17
5.1 Source data	18
5.2 Source metadata	28
6 1000 Genomes Project Data Transformation	33
6.1 1000 Genomes Project region data transformation	33
6.1.1 Overview	33
6.1.2 Splitting of multiallelic mutations	34
6.1.3 Conversion of genomic coordinates	36
6.1.4 Reduction to minimal-form	39
6.1.5 Computing the length of a mutation	41
6.1.6 Decoding of the genotype string	43

6.1.7	Recognition of the type of mutation	43
6.1.8	Format of output regions	44
6.2	1000 Genomes Project metadata transformation	48
6.2.1	Family ID and Gender	48
6.2.2	DNA Source from Coriell	49
6.2.3	Experiment, Population, Analysis group and others meta	49
6.2.4	Super-population, DNA from Blood	51
6.2.5	Manually curated metadata	52
6.2.6	Format of output metadata	54
6.3	Generated datasets	54
6.3.1	Test performed	55
6.3.2	Statistics	55
7	Implementation of Metadata-Manager modules for 1000 Genomes Project data source	57
7.1	Download of the source files	58
7.1.1	Resilience to network issues	63
7.1.2	Reliability aspects	63
7.2	Data transformation	65
7.3	Modifications to the framework	70
7.4	Final integration steps	72
7.5	Performance optimizations	74
7.5.1	Dynamic source code generation and compilation	75
7.5.2	Parallel transformation	76
8	Data-Summarization API	79
8.1	Design of supporting Data Warehouse	80
8.2	Requirements	83
8.3	Assumptions	84
8.4	Software design and architecture	84
8.5	API definition	88
8.6	Combining the results of the sources - An example	88
8.7	Dealing with usability of API	90
8.8	Measures for guaranteeing scalability	93
8.8.1	Integration of TCGA	94
8.9	Measures for improving performance	95
8.10	Discussion on privacy and dynamic incompatibility	97
8.11	Use case - Genes functionally involved in skin melanoma by differential analysis	104

9	Conclusions and future prospects	111
	Bibliography	113
Appendix A	Assessment of 1000 Genomes Project source	
	access methods	117
A.1	Access methods	117
A.1.1	Data browsers	117
A.1.2	Application Programming Interfaces	118
A.1.3	File Transfer Protocol	127
A.1.4	Amazon S3 bucket	129
A.1.5	Other transfer services	133
A.2	Selection of the best access method	133
Appendix B	Examples of transformed data	139
B.1	Region data examples	139
B.2	Metadata examples	141
Appendix C	Rules of mapper module for transformed	
	metadata of 1000 Genomes Project	143

List of Figures

2.1	GCM schema	7
3.1	Ensemble analysis tool. Detail of variant genotype frequencies.	13
5.1	Ambiguity of expression in VCF variant encoding	26
7.1	Hierarchical organization of dataset versions and files	60
7.2	Structure of the <i>tree file</i>	60
7.3	Overview of candidate names generation	66
7.4	Overview of the transformation	68
7.5	Callgraph of the transformation and post-processing operations in the modified framework.	72
7.6	Entities and relations described by the Genomic Conceptual Model	74
8.1	Schema of the data warehouse for Data-Summarization-API .	80
8.2	Data-Summarization-API architecture overview	85
8.3	Online documentation of Data-Summarization-API	92
8.4	Workflow of the message subsystem for Data-Summarization-API.	104
8.5	Gene scores in tumor cohort.	108
8.6	Gene scores in healthy cohort.	108
8.7	Differential analysis on gene mutational profiles	109
A.1	Excerpt of the 1000 Genomes Project phase 3 genome browser	119
A.2	IGSR Data browser	120
A.3	Overview of the Ensembl API	127
A.4	Overview of the Ensembl Structural Variation API	128

List of Tables

5.1	List of the region files for the assembly hg19 available via FTP.	19
5.2	List of the region files for the assembly GRCh38 available via FTP.	20
5.3	Accuracy and error probability for common Phred quality scores.	21
5.4	Number of samples and alleles participating to the study. . .	23
5.5	Region attributes in 1000 Genomes Project source data	26
6.1	Example of multiallelic variant in 1000 Genomes Project source data	34
6.2	Biallelic representation of the Variant from Table 6.1	35
6.3	Region attributes cardinality descriptors in 1000 Genomes Project source data	35
6.4	Allele representation in structural variants from 1000 Genomes Project source data	38
6.5	Examples of variant length computation from the source region attributes	42
6.6	Rationale for identifying the type of a short variant	44
6.7	List of region attributes generated by the transformation of the source	44
6.8	Example of map generated by the transformation of metadata file of class <i>individual details</i>	49
6.9	Example of map generated by the transformation of metadata file of class <i>sample info</i>	49
6.10	Selected attributes from class <i>sequence index</i>	50
6.11	Example of dictionary generated from the <i>population</i> metadata	52
6.12	List of manually curated metadata.	53
6.13	Manually curated metadata added for each source region file contributing to the sample.	54
6.14	Statistics on the input datasets.	56
6.15	Statistics on the output datasets.	56

8.1	Endpoints available in Data-Summarization-API.	88
8.2	Example result of function variant_occurrence from class VariationSource	89
8.3	Mapping of the ethnicity over population values from 1000 Genomes Project.	95
8.4	Filters available for TCGA and 1000 Genomes Project in Data-Summarization-API	96
A.1	Ensembl genome browsers	118
A.2	Ensembl database servers	121
A.3	Ensembl pilot and phase 1 database server	121
A.4	Endpoints of the Variation RESTful API from Ensembl . . .	124
A.5	Comparison of methods for accessing the 1000 Genomes Project source data and metadata	136
B.1	Examples of original mutations from 1000 Genomes Project .	140
B.2	Examples of mutations from 1000 Genomes Project after the transformation in GDM.	140
B.3	List of 1000 Genomes Project metadata attributes after the transformation	141
C.1	Rules of Mapper module for Metadata-Manager	144

Listings

6.1	Graphical representation of a SNP with respect to 0-based and 1-based coordinate systems.	36
6.2	Example of SNP in 0-based coordinates	37
6.3	Example of insertion in 1-based notation	37
6.4	Effect of insertion inside a reference sequence	37
6.5	Example of insertion in 0-based notation	38
6.6	Multiallelic mutation with redundant bases.	39
6.7	Example of non-minimal variant representation	39
6.8	Minimal representation of a SNP	39
6.9	Complex multiallelic mutation of chromosome MT involving 2 SNPs and 3 INDELs.	40
6.10	Graphical representation of equivalent possible alignments resulting from a non-minimal deletion in VCF.	40
6.11	Values of genotype string in biallelic variants	43
6.12	Output region schema	47
7.1	Excerpt from the XML configuration file used in to map meta-data pairs to GCM relational rows.	75
8.1	Definition of the method <code>variant_occurrence</code> in class <code>VariationSource</code>	88
8.2	Snippet of code calling the endpoint <code>most_common_variants</code>	91
8.3	Transformation of the output of Data-Summarization-API into a Pandas <code>DataFrame</code>	91
8.4	Excerpt of response from the endpoint <code>rarest_variants</code> showing a warning message	99
8.5	Snippet of code checking the violation of a constraint to prevent identification and discrimination of the donors	100
8.6	Enforcement of privacy constraints through prepared ready-made components	101
8.7	Example of response when a privacy policy is violated	102

8.8	Example of request to know the variants falling inside the SERPINE1 gene	106
8.9	Example of request to know the donors owning a variant . . .	107
A.1	Snippet of code showing the usage of Ensembl's PERL API to fetch a variant from Ensembl variation database.	122
A.2	Chunk of response from the Ensembl RESTful API	125
A.3	Snippet of code showing how to query an Amazon S3 bucket with the Java SDK.	131

Chapter 1

Introduction

More than three billion pairs of nucleotides that can have four possible values each. It is an incredibly high number of combinations to code every single function of our cell and ultimately, describe our body. An impressive amount of information, especially when compared to the storage device, a living cell of about $100\mu\text{m}$ in diameter¹ that, by the way, contains also – if you will indulge the term – the hardware necessary to read, interpret, and execute these instructions. As of today, not even our most performing piece of technology can cope with it, neither we can achieve a comparable compression ratio when dealing with normal files. Given such complexity, it was not possible to delve into the details of genetics until the 1960s, despite the genome was known since the previous century [5]. The first progress in genomics was possible by advancements in the computer industry and information technologies that lead to the sequencing of bacteriophage RNAs [30]. Since then, a large number of projects started with the aim of unravelling the unknown aspects of our cells, including the 1000 Genomes Project [22], which is one of the most ambitious initiatives of the past decade, i.e. to fully sequence the DNA of a thousand individuals from all over the world representative of the human diversity. Thanks to the rise of Next Generation Sequencing (NGS, [25])² in the mid-2000s, which determined a rapid growth of the volume and quality of the data, the project surpassed its initial goal, resulting in over 2.5 thousands of individuals' genomes sequenced. Not only NGS allowed sequencing methods to become cheaper and to complete faster

¹ The volume of a eukaryotic cell can vary widely between tissues and different levels of maturity. Also, some cells have different shapes for which the diameter may not necessarily represent a meaningful measure. Still, for the sake of simplicity and the purpose of our speech, we can consider on average the diameter of $100\mu\text{m}$ as a valid representation of cell size.

than before, but it also made the requirement for processing tools that are up to the task even more relevant. Since that moment, the main challenge changed from obtaining the data to making sense of it, and the difference between data and information became tangible – a very well-known issue when dealing with Big Data. One of the key principles while dealing with voluminous and incremental datasets is that the amount of useful knowledge one can get from the fusion of several data sources is always greater than the sum of the information achievable by the sources themselves. This concept applies very well to genomics and bioinformatics, where many applications require to understand the cause-effect relations between the components that constitute a living organism. Besides the biological question to solve, this task puts far from trivial challenges regarding the interoperability of the data sources. On the one hand, this is due to the extent of the research area (RNA sequencing, DNA sequencing, epigenomics, variant calling and genomic annotations, just to name a few). On the other hand, it has to do with the fact that data are produced by laboratories that operate without prior coordination and are collected through different instruments and pipelines. Even when dealing with similar entities, it may not be possible to reconcile the data coming from two sources because of the significant differences in accuracy, coverage and documentation that characterise them.

Therefore, data integration is fundamental to relate the large quantity of available data which are largely heterogeneous both in the subject and in the representation format. However, it is equally important to provide convenient and efficient instruments to analyse and describe such data through meaningful measures. In that respect, few remarkable solutions to achieve summarisation of genomic populations have been proposed, although, at present, they exhibit substantial limitations, either from the functional or the technical perspective. Such limitations make it more difficult to exploit the richness of the data or the integration into the already existing processing pipelines. Given such context, the purpose of the thesis is to integrate the 1000 Genomes Project’s data into a repository of interoperable genomic resources and to build an API providing descriptive statistics on user-defined populations i.e., having precise region and/or metadata characteristics, overcoming some of the shortcomings observed in similar tools.

Before describing how we achieve the aforementioned goals, in Chapter 2 we provide a quick introduction to the Genomic Data Model [20] and the Genomic Conceptual Model [1], which together allow a uniform representa-

² NGS is a technology that superseded the original Sanger sequencing, producing a dramatic increase of the throughput and at the same time lowering costs and time.

tion and a consistent semantics of genomic data across diverse sources. We explain the workflow of Metadata-Manager, i.e., the integration framework used to transform data into GDM and import it into a human genomic data repository. Finally, we present the GenoMetric Query Language, a system developed to access a GDM dataset and extract information in an efficient way. In Chapter 3, we briefly discuss the state of the art, analysing the features of two well-known instruments to summarise genomic populations. Chapter 4 summarizes the goals of the thesis.

Then, we address the data integration problem for 1000 Genomes Project, starting with the analysis of the source in Chapter 5, where we list in detail the available data and metadata.

Chapter 6 describes the transformation applied to the original data to produce GDM compliant regions and metadata files. This chapter contributes to the definition of a method of general validity to convert Variant Call Format (VCF) regions into BED-like³ format and between genomic coordinate reference systems.

Chapter 7 illustrates the implementation of the modules for the framework Metadata-Manager highlighting the core components and the strategies helping to improve the efficiency of the download and transform phases. Moreover, it describes the changes applied to the framework to make it more flexible and capable of integrating with a larger number of sources.

Chapter 8 instead, shows the architecture of Data-Summarization API and addresses privacy constraints and other issues with appropriate examples.

In the last chapter, we conclude by summarizing the achievements and discussing future development options for the Data-Summarization API.

Finally, Appendix A describes and evaluates all the methods considered for accessing the 1000 Genomes Project data according to the metrics of completeness and suitability to the purpose. Appendix B compares the form of some genetic variants between the original format and the one produced by our transformation to facilitate their integration. Lastly, Appendix C shows the syntactic rules used during the integration process to map the attributes of data source 1000 Genomes Project to the Genomic Conceptual Model.

³ BED is a file format for describing genomic regions in a simple and expandable way.

Chapter 2

Background

In this chapter, we give an introduction to the concepts needed to understand this document and the software solutions supporting the achievement of the thesis' goals. It is not meant to be a complete and rigorous description, for which we encourage the reader to see the references [20] [1] [3] instead. .

2.1 Genomic Data Model

The Genomic Data Model (GDM) [20] is a data format designed for representing a generic genomic entity in a very flexible fashion, providing a single abstraction to represent largely heterogeneous data.

In GDM the main entity is the sample, to which genomic regions and meta-data are associated using two simple data models. Genomic regions correspond to portions of the genome and are identified by 4 mandatory attributes which completely describe the location of the feature described. In order, they are: chromosome, start (or left extremity), stop (or right extremity) and strand, having data types string, long, long and char respectively. Coordinates are expressed in the 0-based notation and the strand accept 3 values: "+", "-" and "*" when unknown. Other attributes peculiar to a region can be specified afterwards with any number of values and using the types boolean, char, string, int, long or double, depending on the needs.

Metadata are key-value pairs used to describe a property of the sample, like the information related to the clinical relevance, biological aspects and the experimental conditions in which the sample was taken. Inside each pair, the key and the value should be separated by a blank space and each key-value pair must occur only once. Still, to represent multi-valued properties, the same key can be repeated multiple times inside each sample. So, in GDM, a sample is entirely described as the set of regions and the set of

metadata (key-value pairs) bound to it.

A dataset is just a collection of samples. However, while metadata can have different or same key-value pairs for each sample, all the samples' regions must adhere to the same dataset schema, i.e. same number, order, type and semantics of the attributes. Still, since a dataset is aimed at representing semantically homogeneous regions, this is not a limitation at all.

Thanks to this high flexibility, the GDM format has been successfully used to represent an incredibly high number of different genomic entities, generating a repository of GDM datasets that includes gene expression data, mutations, annotations, binding peaks and more from open sources like ENCODE, Roadmap Epigenomics and TCGA.

2.2 Genomic Conceptual Model

The GCM [1] describes the relation between some general and commonly adopted metadata properties of a sample. This relational model gives a logical structure to the attributes, increasing their semantics beyond just the attribute meaning. As a result, it helps the data integration process by enforcing a common metadata representation for multiple sources. Moreover, GCM brings all the benefits of a relational data model: simplified usage from other applications, minimal representation and allows integrity checks to be implemented. The GCM offers three views on metadata:

- the Biological View describes the biological process leading from the donor to the measured entity, i.e. a genomic variation, a measured peak of binding, etc.
- the Technology View details some of the aspects related to the instrumentation and techniques used in the experiment that leads to the production of the measured entity, i.e. the software pipeline, the sequencing machine, the experiment target and so on.
- the Management View illustrates the project and the case study that originated the measured entity.

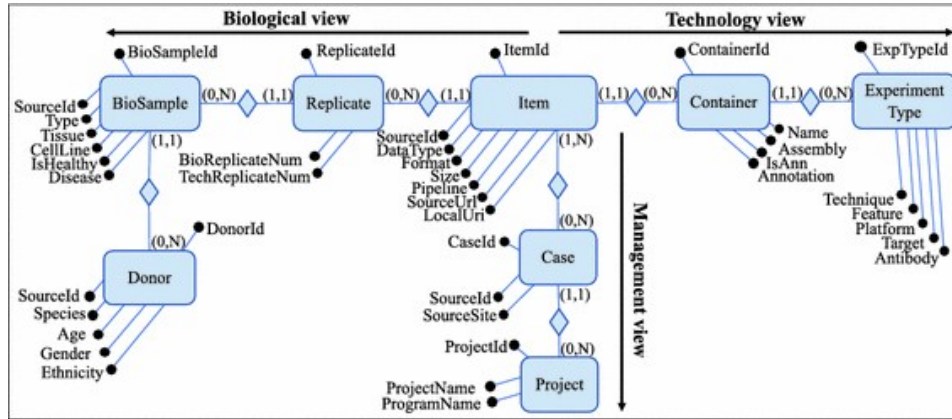
The same partition is illustrated in Figure 2.1.

2.3 GenoMetric Query Language

Tertiary data analysis means to study the relationship between genomic entities in order to entail new knowledge. This process requires to join infor-

2.3. GenoMetric Query Language

Figure 2.1: The picture illustrates metadata attributes included in the Genomic Conceptual Model organized into three branches. Source of image: [1]



mation coming from diverse origins, so besides the biological question, other additional challenges may arise:

- Working with voluminous datasets that require large storage capacity and high computational power.
- Applying transformations both to data and metadata that are scattered in different files.
- Merging information from multiple data formats
- Create ad-hoc scripts that invoke many purpose-specific software tool and converting input and output as necessary.

GMQL [19] is at the same time a high-level language and a system designed to simplify all these tasks providing a uniform way to query large heterogeneous genomic databanks. As a language, it provides simple instructions to perform complex operations on genomic data and metadata. This enables bioinformaticians to focus on the biological question rather than on the computational steps involved while working with multiple tools and different data formats. GMQL brings to bioinformaticians advantages either by replacing long and complex chains of operations, either by providing a uniform language to perform any operation on heterogeneous datasets, independently from the source data format. Indeed, GMQL queries run on already integrated GDM datasets, eliminating the most common complexities generated by the use of incompatible data formats. Moreover, the language is declarative, so the user has not to care about the optimization of the code, which instead is performed at compile-time by a dedicated engine.

In GMQL, queries are expressed with the following syntax:

$$< output\ dataset > = < command > (< parameters >) < input\ dataset >$$

Though, several APIs exists for common programming languages using a slightly different syntax.

As a system, GMQL is organized according to a four-layer architecture:

- Access layer: this layer abstracts the GMQL commands from the interface used to issue them (APIs, shell interface or GMQL web interface).
- Engine layer: compiles the code, generates an optimized Direct Acyclic Graph (DAG) schematizing the code instructions and coordinates the other layers.
- DAG implementation layer: includes the implementation of the DAG for different cloud computing engines.
- repository implementation layer: provides the GDM data, whether they are stored locally or remotely, in a centralized or in a distributed repository, through different file systems.

All in all, this architecture provides fast and efficient computing tools for working with diverse genomic sources homogeneously.

2.4 Metadata-Manager

Metadata Manager [3] is a framework for the integration of genomic datasets. It is designed to guide the transformation of any genomic data source into an integrated GDM [20] dataset through seven progressive steps and it is extensible by means of self-contained and source-specific modules. Thanks to Metadata-Manager, a whole, fully-integrated GDM repository can be generated incrementally as the result of subsequent runs that download, transform and refine each genomic data source. In detail, the integration of a source is achieved through these steps:

1. Download: The most recent version of the source's data and metadata are downloaded into a local repository. Moreover, the local repository is used to detect changes in the remote source and decide to repeat the following stages, if necessary.

2.4. Metadata-Manager

2. Transform: A dedicated module reads the source files and determines which are the samples to be transformed. Then the framework guides the transformation between each pair of input and output files. At the end of the process, a GDM representation of the source is generated.
3. Cleaner: This step prepares the mapping of metadata keys into the corresponding entities of the Genomic Conceptual Module (GCM) by simplifying the keys for the imported dataset. Typically, the Transform module can return long, hard-to-read keys depending on the complexity of the source's metadata structure, so the Cleaner module shorten and simplifies them by removing deeply nested hierarchies in the attribute names.
4. Mapper: The Mapper module matches metadata attributes across different source by mapping them to a uniform metadata structure, the Genomic Conceptual Model. Because of the global validity of the GCM schema, this step is at the core of the integration process between genomic sources, which is completed with the following two steps.
5. Normalizer/Enricher: this stage makes use of a local knowledge base, as well as multiple selected ontologies, to cluster synonyms in the integrated data repository and suggest replacement names. Moreover, in some cases the ontologies can be used to entail possibly missing attributes. This step has the potential to improve the interoperability between the integrated sources and simplify tertiary analysis.
6. Integrity Checker: this step is complementary to the Normalizer/Enricher and the Mapper stages as it verifies the validity of certain assertions limiting the values acceptable for a given attribute. This is accomplished by forcing integrity constraints either on individual attributes, either validating their reciprocal relation.
7. Flattener: Mapping, Normalizer and Integrity Checker stages are database supported. This stage selects the output of the previous stages to convert it back to a file representation, useful for the next stage.
8. Loader: This last stage uploads the newly obtained GDM dataset to a target GDM repository, where it can be then queried through any of the GDM-compatible tools. Depending on the needs, the Loader can be configured for uploading the metadata either right after the Transformation, or after the cleaning stage or after the Integrity Checker.

The execution of the integration steps can be fully configured through an XML configuration file, whose parameters define which modules to use in any stage and in which mode to run the Loader module.

Chapter 3

State of the art

In this chapter, we give an overview of the two most relevant solutions currently adopted to analyse large-scale variation datasets. Of each, we describe the main features, as well as the shortcomings that we intend to address as part of the goals of this thesis.

3.1 gnomAD

The Genome Aggregation Database (gnomAD) [11] is the result of an international coalition of investigators for aggregating and harmonizing exome and genome data from a variety of large-scale sequencing projects. To provide summary data about variants, the raw data from 63 projects, the 1000 Genomes Project among them, were reprocessed through equivalent pipelines and jointly variant-called to obtain a consistent representation across sources. The result was the creation of two very-large data collection known as gnomAD v2 and gnomAD v3. Currently, gnomAD v2 contains 125784 exomes and 15708 genomes aligned on the reference assembly hg19, while gnomAD v3 contains 71702 genomes aligned on grch38. The former is available also in 4 variants with a reduced number of genomes, obtained according to the categorization of the samples into control, non-cancer, non-neuro and non-TOPMed.¹ Browsing a variant into the gnomAD browser returns a large amount of information, detailing:

- gene, intron and exon annotations

¹ Control samples correspond to the samples that were not selected as a case in a case/-control study of common disease. Non-cancer and non-neuro are the samples from individuals who were not ascertained for having, respectively, cancer in a cancer study or a neurological condition in a neurological case/control study. Non-TOPMed, instead are the samples that are not present in the Trans-Omics for Precision Medicine (TOPMed)/BRAVO release.

- multi-allelic sites
- age distribution of the donors analysed
- genotype and site quality metrics
- global allele count, number of occurrences and frequency of a variant
- allele count, number of occurrences and frequency of a variant grouped either by ethnicity or population or gender

It is also possible to browse a region or a gene into the gnomAD browser to view the variants falling inside the corresponding region, their global frequency, allele total count and allele number, their associated annotations (e.g. intron, missense, synonymous), and, only for genes, the expression status of exonic regions across tissues.

Thanks to the high level of maturity reached and the comprehensive analysis offered both on variants and transcripts, gnomAD proves itself to be a helpful and powerful analysis tool for genomics. However, in order to output the requested measures, it mostly relies on pre-computed analyses. Total allele count, allele number and frequency, along with other values, are indeed saved into multiple VCF files, compressed and indexed using tabix [17], so that, when a variant is requested, the gnomAD browser can easily fetch and display the related statistics. While this approach can bring important advantages in terms of performance (storage space and response time), on the other side it brings some limitations. As the measures are fixed, the user cannot obtain statistics on a dynamically chosen sample set, other than the ones already proposed for the control, non-cancer, non-neuro and non-TOPMed sets. Even if one of the pre-built sets of samples matches the requirements for the population of interest, then it is not possible to restrict them to a specific combination of metadata attributes only (e.g. males samples from Italy), neither is possible to include into the population only the samples having precise variants or multi-valued metadata attributes.

Lastly, gnomAD does not implement an API or alternative interface to the gnomAD browser, therefore the integration of this instrument into a processing pipeline is hardly achievable.

With this thesis, we want to demonstrate how it is possible to create a flexible genomic data analysis tool for studying a user-defined sample set having arbitrary meta attributes and fine-grained region attributes, for example by including in the population only the samples with one or more















3.2. Ensembl analysis tools

particular variants², or the samples that showed a mutation in a genomic area.

3.2 Ensembl analysis tools

Even Ensembl, mostly known for its genome browser, provides web tools to study variants and populations. Similarly to gnomAD, it is possible to compare the frequency of a variant or haplotype in samples with different origins (and the corresponding super-groups) by selecting a variant or a gene transcript into the Ensembl web interface. In general, the returned statistics are provided by external services, gnomAD among them, but if the target variant belongs to the 1000 Genomes Project study, then a richer analysis is available, showing the allele frequency and count in all the 26 population groups taking part to the original study. Besides them, the interface can report the list of sample names included in each group and the genotype frequencies and counts.³

Figure 3.1: Ensemble analysis tool. Detail of variant genotype frequencies.

Population	Allele: frequency (count)		Genotype: frequency (count)			Genotypes
ALL		G: 0.810 (4054) T: 0.190 (954)	G G: 0.672 (1682)	G T: 0.276 (690)	T T: 0.053 (132)	Show
AFR		G: 0.930 (1229) T: 0.070 (93)	G G: 0.868 (574)	G T: 0.123 (81)	T T: 0.009 (6)	Show
ACB		G: 0.896 (172) T: 0.104 (20)	G G: 0.812 (78)	G T: 0.167 (16)	T T: 0.021 (2)	Hide
ASW		G: 0.877 (107) T: 0.123 (15)	G G: 0.754 (46)	G T: 0.246 (15)		Show
ESN		G: 0.944 (187) T: 0.056 (11)	G G: 0.899 (89)	G T: 0.091 (9)	T T: 0.010 (1)	Show
GWD		G: 0.951 (215) T: 0.049 (11)	G G: 0.912 (103)	G T: 0.080 (9)	T T: 0.009 (1)	Show
LWK		G: 0.904 (179) T: 0.096 (19)	G G: 0.828 (82)	G T: 0.152 (15)	T T: 0.020 (2)	Show
MSL		G: 0.988 (168) T: 0.012 (2)	G G: 0.976 (83)	G T: 0.024 (2)		Show
YRI		G: 0.931 (201) T: 0.069 (15)	G G: 0.861 (93)	G T: 0.139 (15)		Show
AMR		G: 0.811 (563) T: 0.189 (131)	G G: 0.674 (234)	G T: 0.274 (95)	T T: 0.052 (18)	Show
CLM		G: 0.755 (142) T: 0.245 (46)	G G: 0.564 (53)	G T: 0.383 (36)	T T: 0.053 (5)	Show
MXL		G: 0.875 (112) T: 0.125 (16)	G G: 0.797 (51)	G T: 0.156 (10)	T T: 0.047 (3)	Show
PEL		G: 0.912 (155) T: 0.088 (15)	G G: 0.847 (72)	G T: 0.129 (11)	T T: 0.024 (2)	Show
PUR		G: 0.740 (154) T: 0.260 (54)	G G: 0.558 (58)	G T: 0.365 (38)	T T: 0.077 (8)	Show

For a gene or transcript, the web interface shows a table and a visual overview of the variants falling into the region, integrating the data from multiple sources. Together, the set of tools provided by Ensembl provide a good analysis of a genomic population, although they present two important limitations:

1. Available populations correspond to sets of samples grouped by country or continent only. It is not possible to filter the population by gender or other attributes.

² By extension, also variant haplotypes.

³ The frequency of a genotype differs from the frequency of an allele, since the former considers the occurrence of the pair of alleles found on each chromatid. So for a diploid organism and a simple SNP, three pairs can be formed: (ref,ref), (ref, alt) and (alt, alt) where ref and alt are respectively the reference and alternative alleles.

2. The haplotypes are available only for 1000 Genomes Project samples. So, user-defined filters on samples' regions are not possible or they are limited to predefined options in a restricted set of samples.

Chapter 4

Thesis Goals

In Chapter 3 we reviewed two of the most important tools providing data summarization for genomic variation data sources. However, from our analysis, several limitations emerged that we intend to overcome by developing a new software. This novel architecture must be general enough to be used with multiple data sources and merge their information in order to increase the quantity of usable data. It must allow the definition of populations according to fine-grained region and metadata attributes, for example by considering only the samples having a tuple of arbitrary metadata parameters and specific groups of variants. Moreover, we want this tool to provide an interface that should be easy to integrate into the already existing processing pipelines and easy to use.

To demonstrate the effectiveness and extensibility properties of this instrument, we plan to use it with the data from two well-known variation data sources: the 1000 Genomes Project and The Cancer Genome Atlas Program (TCGA). Our software should be widely flexible on a per-source and per-request level, allowing to meet source-specific requirements. Indeed, analysing whole-genome data brings inevitable concerns on privacy and security aspects, not to mention the possible ethical implications. Being 1000 Genomes Project the last extensive whole-genome data repository that have been released without embargo, we want to provide an instrument ready for future integration with other genomic resources as well, by making possible to restrict the result offered from each source, according to the privacy policy in use and without limiting the other integrated sources.

To ease the analysis of diverse data sources and allow future expansions, we are going to leverage the potentiality offered by the Genomic Data Model and the Genomic Conceptual Model. So, the first part of the work is dedicated to the integration of the 1000 Genomes Project data into the GDM

repository maintained by the Genomic Computing group of Politecnico di Milano. Once completed, we use the integrated repository, which already includes TCGA's data, as data structure for our novel variant analysis software, which is discussed in the second part of the thesis work.

Chapter 5

1000 Genomes Project data source

The 1000 Genomes is an international effort for sequencing and collecting the largest dataset of human variation genotypes, from a sample population of at least 1000 individuals. The goal is to provide a deep characterisation of human genome sequence variation, as a foundation for investigating the relationship between genotype and phenotype.

The project began in 2008 and evolved through three pilot phases and three main phases. The three pilot phases [22] focused on developing and assessing protocols and algorithms for the main phases of the project, in particular (i) to generate sequence data, (ii) to detect variants from these and (iii) for defining an efficient approach to measure common genome-wide variants, as well as rare variants in regions of functional interest, such as exons. Once the method was validated, the main project could start. The phase 1 [23] evolved the inherited framework and developed methods to integrate information across multiple algorithms and diverse data sources, leading to improved sequencing quality across 1092 individuals from 14 populations, and provided a validated haplotype¹ map of 38 million single nucleotide polymorphisms (SNPs), 1.4 million short insertions and deletions², and more than 14000 larger deletions.

Phase 2 of the project focused mainly on technical development and the 1000 Genomes Project Consortium did not release any documentation about that.

The initial goal of the project was largely exceeded in May 2013, when 2535 genomes from 26 populations were completely sequenced with a com-

¹ A haplotype is a group of variants that are commonly inherited together.

² Often simply called "indels".

bination of low-coverage whole-genome sequencing and deep exome sequencing. These data were the object of study for the third and final phase of the project [24], lasting about two years, during which over 88 million (comprising 84.7 million SNPs, 3.6 million short indels and 60000 structural variants) variants were detected. Further analyses were conducted separately by the Structural Variation Analysis Group inside the 1000 Genomes Project Consortium [26] in order to enrich the dataset with the genotypes of structural variations, resulting in the removal of 457 previously catalogued variants and the identification of 9848 more.

Later, the International Genome Sample Resource (IGSR) [4] was established at the European Molecular Biology Laboratory's - European Bioinformatics Institute (EMBL-EBI) to maintain and expand these resources. Since then, the IGSR provides access to several intermediate releases, the three datasets corresponding to the data available at the end of the pilot phase, main phase 1, main phase 3 and further releases generated by IGSR, containing the remapping of the final dataset over the more recent GRCh38 reference genome. [18]

Currently, the project's data and metadata available through several means, including various browser interfaces, APIs, databases and file storage services, although with some significant differences. In a careful assessment of the known access methods, we evaluated each method comparing the completeness of the data available through it, ease of access and expected future maintenance of the software in charge of collecting the data. Through this analysis, reported in Appendix A, we concluded to use the FTP protocol as our preferred access method. However, because of the varying detail with which the access methods present the source's data, in the following work, we assume to use one of the two FTP servers:

- `ftp.1000genomes.ebi.ac.uk/vol1/ftp/`
- `ftptrace.ncbi.nih.gov/1000genomes/ftp/`

5.1 Source data

The first kind of data we want to import from the source is the set of variants discovered, i.e. the primary result of the whole project which we simply call "data". 1KG (acronym for 1000 Genomes) provides such information in the form of a set of Variant Call Format ³ (VCF) files organized by assembly.

³ VCF is a file type defined during the course of the project by the same consortium responsible for the 1000 Genomes Project and it is maintained by an independent group

5.1. Source data

In Tables 5.1 and 5.2 are listed the latest version of the data respectively located inside the directories⁴:

- `ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/` for the assembly hg19
- `ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000_genomes_project/release/20190312_biallelic_SNV_and_INDEL/` for the assembly GRCh38

Table 5.1: List of the region files for the assembly hg19 available via FTP.

Filename	Approximate size (MB)
ALL.chr1.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	1126.4
ALL.chr2.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	1228.8
ALL.chr3.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	1024
ALL.chr4.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	1024
ALL.chr5.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	943
ALL.chr6.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	953
ALL.chr7.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	866
ALL.chr8.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	821
ALL.chr9.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	642
ALL.chr10.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	738
ALL.chr11.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	732
ALL.chr12.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	706
ALL.chr13.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	531
ALL.chr14.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	483
ALL.chr15.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	437
ALL.chr16.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	472
ALL.chr17.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	414
ALL.chr18.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	416
ALL.chr19.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	343
ALL.chr20.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	326
ALL.chr21.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	208
ALL.chr22.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz	205
ALL.chrX.phase3_shapeit2_mvncall_integrated_v1b.20130502.genotypes.vcf.gz	1843.2
ALL.chrY.phase3_shapeit2_mvncall_integrated_v2a.20130502.genotypes.vcf.gz	5.5
ALL.chrMT.phase3_shapeit2_mvncall_integrated_v0.4.20130502.genotypes.vcf.gz	0.19

since the publication of the project's results. The file format specification is available at <https://samtools.github.io/hts-specs/>.

⁴ Note that the same directories can be found also on the NCBI server at `ftptrace.ncbi.nih.gov/1000genomes/ftp/`

Chapter 5. 1000 Genomes Project data source

Table 5.2: List of the region files for the assembly GRCh38 available via FTP.

Filename	Approximate size (MB)
ALL.chr1.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	992
ALL.chr2.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	1024
ALL.chr3.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	909
ALL.chr4.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	914
ALL.chr5.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	821
ALL.chr6.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	824
ALL.chr7.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	743
ALL.chr8.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	705
ALL.chr9.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	548
ALL.chr10.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	641
ALL.chr11.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	627
ALL.chr12.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	610
ALL.chr13.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	461
ALL.chr14.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	415
ALL.chr15.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	371
ALL.chr16.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	398
ALL.chr17.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	351
ALL.chr18.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	361
ALL.chr19.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	294
ALL.chr20.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	293
ALL.chr21.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	179
ALL.chr22.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	177
ALL.chrX.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz	22.9

The VCF standard expects that every mutation is described with a set of mandatory attributes formatted in tab-separated columns:

- CHR: TEXT - The chromosome where the mutation is located
- POS: NUMERIC - The position of the first nucleotide in the reference allele expressed in 1-based coordinates
- ID: TEXT - The ID associated to this mutation. It can be either a new ID defined by the provider of the file or an already existing ID from an external databank. 1KG data is often enriched with mutation IDs from databases dbSNP and dbVar but when this value is missing, a point (the punctuation symbol) is specified instead.
- REF: TEXT - The reference allele expressed as a sequence of one or more letters (A, C, T, G) referring to the nucleotide bases found on the reference genome for the assembly in use.
- ALT: TEXT - The alternative allele reports the sequence of bases found on the sample sequenced in place of the reference. Other than T,C,G,A, it is possible to find other values in this attribute, indeed most of the structural variants sequenced during the project are encoded as symbolic alleles. As an example, "<CN0>" is a long deletion, "<CN2>"

5.1. Source data

is a copy-number 2 variant, while "<INS:ME:ALU>" is the insertion of an ALU transposon.

- **FILTER: TEXT** - A string indicating if the called variant has been accepted or rejected by the filtering process. When the value is not "PASS", this string contains one or more codes to indicate that the variant did not pass all the filters and what are the causes. The codes are defined by the tools used to produce the VCFs and filter the variants, as such it is possible to find any kind of codes.
- **QUALITY: NUMERIC** - It is the Phred-scaled quality score. As for sequencing, it is computed with the usual formula $-10 \cdot \log(E)$ where E is the probability of the error. In the context of variant calling and in particular of 1000 Genomes' VCFs, E assumes the meaning of "probability that the assertion made in ALT is wrong". So the negative logarithm of E represents the estimate of how confident we are that the variant has been identified correctly. High values of QUALITY indicate a high probability that the variant stated is correct, while conversely low values indicate a high probability that the called variant is wrong. Since the scale is unbounded, it is possible to have any quality score from minus infinite to plus infinite.

Table 5.3: Accuracy and error probability for common Phred quality scores.

Quality	Error	Accuracy (1 - Error)
10	1/10 = 10%	90%
20	1/100 = 1%	99%
30	1/1000 = 0.1%	99.9%
40	1/10000 = 0.01%	99.99%
50	1/100000 = 0.001%	99.999%
60	1/1000000 = 0.0001%	99.9999%

The format is also largely extensible by the source which can define and use additional attributes. The 1KG VCFs are enriched with a wide set of additional information encoded under the column **INFO** (8th column) into *<key>=<value>* pairs separated by semi-colons. It follows a list describing all the attributes appearing within 1KG VCFs and the abbreviations in use.

- **AA: TEXT** - In opposition to a derived allele, the ancestral allele is an allele that did not arise because of a mutation during the evolution of the specie and for this reason it is believed to be the original value of a genetic trait. This information can be obtained by inferring the bases that are conserved over a phylogenetic tree and more practically, in the case of humans, by aligning the target sequence with the ones from multiple primates like apes, orangutan or chimpanzee and looking at the conserved bases. It is not possible to infer the ancestral allele for all bases, and also 1000 Genomes does not provide such information for structural variants, MNPs and ME insertions; when this is the case it is simply NULL or ".|||" or variations of it (e.g. "||| unknown(NO_COVERAGE)", "|||unknown(LONG_INSERTION)"). When instead it is available, it assumes the value AA||| (with AA the residual trait) for SNPs or AA|REF|ALT|insertion for INDELs (only insertions).
- **AC: NUMERIC** - It is the number of times this allelic variant occurred across the population studied, indeed it is used to compute the corresponding allele frequency. Note that this number takes into account the possibility that a mutation pattern repeats on both copies of a chromosome and for this reason each sample exhibiting a mutation can contribute to this value by 1 or 2 units.
- **NS: NUMERIC** - The number of Samples participating to the study. See Table 5.4 for further details.
- **AN: NUMERIC** - It is the number of alleles in the considered population. The Allelic frequencies of 1000 Genomes variants are computed over multiple sets of samples, which means that depending on the considered variant, the total number of alleles may vary widely as shown in Table 5.4.
- **AF: FLOATING POINT NUMBER** - The allele frequency is an indicator of the frequency of occurrence of a variant across the population. It ranges between 0 and 1 and it is computed as the ratio between the number of occurrences of the allele carried by a mutation (AC), and the number of alleles of the entire population on the same genomic coordinates, or locus of variation (AN).
- **AFR_AF: FLOATING POINT NUMBER** - Allele Frequency in the African population.

5.1. Source data

Table 5.4: Number of samples and alleles participating to the study.

Assembly	Chromosome where the mutation is located	Number of individuals	Total number of alleles
hg19	Autosomal chromosomes	2504	5008
hg19	X	2504	5008
hg19	Y	1233	1233
hg19	Mitochondrial chromosome	2534	2534
GRCh38	any	2548	5096

- AMR_AF: FLOATING POINT NUMBER - Allele Frequency in the American population.
- EAS_AF: FLOATING POINT NUMBER - Allele Frequency in the East Asian population.
- SAS_AF: FLOATING POINT NUMBER - Allele Frequency in the South Asian population.
- EUR_AF: FLOATING POINT NUMBER - Allele Frequency in the European population.
- END: NUMERIC - Defined only for CNVs⁵, it is the right breakpoint of the variant.
- IMPRECISE - A flag indicating that the mutation parameters END, SVLEN and POS do not respect the equation $|SVLEN| = END - POS + 1$, thus indicating they are approximated values. More generally, imprecise structural variants are identifiable anyway by the presence of the attributes CIPOS/CIEND (confidence interval on POS or END coordinates), while the flag IMPRECISE is reported only in few cases, i.e. when the structural variation is detailed with both SVLEN and the END coordinate.
- CIPOS: TEXT - If not NULL, this attribute implies that the left breakpoint of this mutation is described as a probability curve where the value in POS represents the highest point. The peak value is associated also to a confidence interval expressed in this attribute and formatted as a comma-separated pair of integer numbers where *first_number* \leq

⁵ Copy Number Variants comprise deletion and repetitions of long genomic sequences.

0 and *second_number* \geq 0. Thus the interval of acceptable positions ranges from *POS* + *first_number* to *POS* + *second_number*.

- CIEND: TEXT - This is the equivalent of CIPOS but applied on the coordinate defined in END.
- CS: TEXT - Even if we usually refer to the 1000 Genomes mutations as simply "data", they are actually heterogeneous variant call sets generated individually and finally merged together. The set of provenance of a mutation, if defined, is indicated in this attribute.
- MC: TEXT - Since different structural variant callers have been used in the project, a large number of variant sites have been merged in the final release, so the collapsed variant site IDs are listed in this field.
- DP: INTEGER NUMBER - This is the total read depth, i.e. it is the sum of all the aligned sequences landing in the genomic interval of the mutation considered. Usually during a base calling experiment, segments of DNA are replicated and sequenced thousands to millions of times to increase the accuracy in average. The more a segment is read, the more confident we get about the sequence identified and this number contributes to the increase of the quality score of course.
- EX_TARGET: TEXT - A flag indicating whether the variant is targeting one of the 1000 exons analysed during the pilot study of the 1000 Genomes Project which involved 700 individuals. The complete list of targets is available at [ftp.1000genomes.ebi.ac.uk/vol1/ftp/pilot_data/technical/reference/P3_gene_list.txt](ftp://1000genomes.ebi.ac.uk/vol1/ftp/pilot_data/technical/reference/P3_gene_list.txt). This differs from the whole exome study conducted in phases 1 and 3 which target the entirety of the CCDS (Consensus Coding Sequence) genes, i.e. protein-coding regions that are identically annotated on the human and mouse reference genome assemblies.
- MEINFO: TEXT - Defined only for insertions of mobile sequences, it contains additional details on the specific transposon sequenced
- MULTI_ALLELIC: TEXT - A flag signaling a variant with multiple alternative alleles occurring at the same locus. The alternative values are listed in ALT as a comma-separated list of values.
- VT: TEXT - This attribute is defined for any mutation and categorizes the type of variant in SNP, MNP, INDEL and SV⁶. Instead a more

5.1. Source data

detailed classification is made for structural variants in the attribute SVTYPE.

- SVTYPE: TEXT - If the mutation is a structural variant (VT=SV), then this attribute specifies further the kind of variant as CNV or DUP or one of the transposons (SVA, LINE1, ALU).
- SVLEN: NUMERIC - It reports the length of the structural variation. Unless IMPRECISE is true, this value can be computed as result of the equation $|SVLEN| = END - POS + 1$. Sometimes SVLEN can have a negative value to represent the deletion of a long sequence.
- TSD: TEXT - This attribute refers to mutations involving transposons and specifies a sequence of nucleotides. The TSD is inserted equally on opposite strands at the extremities of the inserted mobile element by the polymerase enzyme in order to repair the DNA which has been excised during the placement of the mobile element by the transposase enzyme. Not all ME insertions specify a TSD.
- OLD_VARIANT: TEXT - The variants released in the final datasets of the 1000 Genomes Project are the result of the integration of different call sets produced by a multitude of analysis tools. This generates discrepancies on the representation of the VCF format since the same genetic variant can be represented in different ways in a VCF file, making difficult to compare variants across call sets. The picture 5.1 exemplifies this observation. *Vt normalize* is a tool from the University of Michigan and applied by the 1000 Genomes Project Consortium to unify the representation of variants and simplifying filtering and duplicate removal [27]. This attribute represents the original variant before *vt normalize* was run as a colon-separated list of string detailing chromosome, position, reference and alternative alleles.

The optional attributes mentioned here are only partially adopted across the two datasets identified by the mutations correspondingly aligned to assemblies hg19 and GRCh38. The Table 5.5 shows the usage of the attributes in the two datasets.

⁶ The variants called in the mitochondrial chromosome of assembly hg19 use instead the keywords S, M and I as abbreviations of SNP, MNP and INDEL.

Figure 5.1: Four different ways to represent the deletion of a CA group. Source of image: [27]

Variant:		Reference Sequence	GGGCACACACAGGG
		Alternate Sequence	GGGCACACAGGG
Genome Reference			Variant Call Format
GGGCACACACAGGG			POS REF ALT
(A)	REF CAC	CAC	6 CAC C
	ALT C	C	
(B)	REF GCACA	GCACA	3 GCACA GCA
	ALT GCA	GCA	
(C)	REF GGCA	GGCA	2 GGCA GG
	ALT GG	GG	
(D)	REF GCA	GCA	3 GCA G
	ALT G	G	

Table 5.5: This Table shows the availability of the region attributes in the two datasets hg19 and GRCh38 from the source. The former has been further classified in three subsets corresponding to the sets of samples used for generating the VCFs. The letter x is used to mark the usage of an attribute.

Attribute	hg19 region data			GRCh38 region data
	chr1-22+X	chrY	chrMT	
AA	x	x		
AC	x	x	x	x
NS	x	x		x
AN	x	x		x
AF	x	x		x
AFR_AF	x	x		x
AMR_AF	x	x		x
EAS_AF	x	x		x
SAS_AF	x	x		x
EUR_AF	x	x		x
END	x	x		
IMPRECISE	x			
CIPOS	x			
CIEND	x			
CS	x			

5.1. Source data

Attribute	hg19 region data			GRCh38 region data
	chr1-22+X	chrY	chrMT	
DP	x ⁷	x		x ⁸
EX_TARGET	x	x		x
MEINFO	x			
MULTI_ALLELIC	x	x		
VT	x	x	x	x
SVTYPE	x			
SVLEN	x	x		
TSD	x			
OLD_VARIANT	x			

Contrary to what may seem from Table 5.5, the data aligned on GRCh38 are as detailed as that aligned on hg19. The lack of many attributes is justified by the fact that the GRCh38 dataset is still under development and, at the time of writing, it reports only SNPs and INDELs, while the other dataset includes also MNPs and the many kinds of SVs previously mentioned.

Finally we know on which sample a mutation occurs by looking at the "genotype". The genotype is a string appearing below each sample column and encoded as $x|x$ or x or x/x where x is a semi-positive integer number denoting the presence in a sample of the x -th allele between the ones listed in the **ALT** column⁹. Intuitively, haploid calls have a genotype string of the second type because the variant can be sequenced only on one chromosome. Whereas the other two types are used in diploid calls to identify on which chromosome copy the variant occurs. The two different expressions used for diploid calls (1st and 3rd forms) refers to the notion of phasing, which is the ability to tell if two or more mutations occurs on the same chromosome copy. Phased genotypes are encoded with the form $x|x$. Instead unphased genotypes can only tell the number of repetitions of the variant in the sample, 0, 1 or 2 times, and it is encoded as x/x . 1000 Genomes Project variants are haploid on chromosomes Y and MT, and diploid phased on the other chromosomes. The genotype string is optional in VCF so the string "GT" is used in the column **FORMAT** to denote the availability of such information.

⁷ Available only for low coverage data

⁸ The read depth is approximated in this dataset

⁹ 0 means that the sample does not carry the mutation.

⁹ mitochondrial

Below, an example reporting a haploid call found on two samples, and a diploid call found once for each sample.

CHROM	POS	REF	ALT	FORMAT	HG00096	HG00097
MT	1438	A	G	GT	1	1
22	17346884	A	G	GT	0 1	1 0

5.2 Source metadata

The metadata available for 1000 Genomes are scarce and mostly contained in files that are valid for both assemblies. All metadata are TSV files without a rigorous structure. For ease of explanation we divided the source metadata files into the classes *sequence index*, *population*, *individual details* and *sample info*. We give an overview of the information contained within each class and the location¹⁰ where to find these files in the following sections.

Sequence index files

These files contain the index of the alignment files and to each it is associated the following list of attributes:

- FASTQ_FILE
- MD5
- RUN_ID
- STUDY_ID
- STUDY_NAME
- CENTER_NAME
- SUBMISSION_ID
- SUBMISSION_DATE
- SAMPLE_ID
- SAMPLE_NAME

¹⁰As usual, files are equally present at EBI and NCBI FTP servers.

5.2. Source metadata

- POPULATION
- EXPERIMENT_ID
- INSTRUMENT_PLATFORM
- INSTRUMENT_MODEL
- LIBRARY_NAME
- RUN_NAME
- RUN_BLOCK_NAME
- INSERT_SIZE
- LIBRARY_LAYOUT
- PAIRED_FASTQ
- WITHDRAWN
- COMMENT
- READ_COUNT
- BASE_COUNT
- ANALYSIS_GROUP

Also, this is the only class of metadata file that differs in the two datasets hg19 and GRCh38, not only in terms of content (since they refer to different alignments), but also of formatting because the latter uses a slightly different syntax and provides also a header description which is absent in the other file. The sequence index file is located at

- `ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/20130502.phase3.sequence.index` for the hg19 aligned dataset.
- `ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000_genomes_project/1000genomes.sequence.index` for the GRCh38 aligned dataset.

The following classes of metadata instead make use of the same files for both assemblies.

Population file

The population file gives summary statistics about the 26 populations participating to the project. For each population it is reported:

- The description of the population
- The population code
- The super-population group to which it belongs
- A Boolean value telling if the population has some blood samples
- A Boolean value telling if the population contains samples from individuals in a father/mother/child relationship
- The number of samples collected in the pilot phase of the project
- The number of samples collected in the phase 1 of the project
- The number of samples collected in the final phase 3 of the project
- The total number of samples in the population

The file is hosted at address `tp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/20131219.populations.tsv`.

Individual details

This metadata file is a detailed report on the characteristics of the sample donors with a special focus on father-mother-child trios. For each sample, this meta indicates:

- Family ID
- Individual ID
- Paternal ID
- Maternal ID
- Gender ("1" for males and "2" for females)
- Phenotype (having always value 0)
- Population code of origin
- Relationship (one of "mother", "father", "child" or "unrel")

5.2. Source metadata

- The sibling IDs
- The second order ancestor IDs
- The third order ancestor IDs
- The children individual IDs
- Comments (e.g. "Sequenced Trio", "possibly contaminated")

Finally it follows a set of Boolean values specifying:

- Sample genotyped during phase 3
- The sample has related genotypes
- Genotypes obtained through Illumina Omni platform
- Genotype obtained by Affymetric chips

The file is located at `tp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/integrated_call_samples_v2.20130502.ALL.ped`.

Sample info

The *sample info* repeats some of the items already seen in the file *individual details* and provides some more, mostly technical aspects related to the sequencing strategy used. Of all 61 attributes contained in this file, we are interested in the one named "DNA Source from Coriell", which tells the origin of the sample distinguishing between blood or lymphoblastoid cell culture¹¹. In this document we do not report all the other attributes available within *sample info* but the file and the complete list of values, is available at `ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/working/20130606_sample_info/20130606_sample_info.txt`.

¹¹ Coriell Institute for Medical Research is an international cell repository providing samples of tissue and cells.

Chapter 6

1000 Genomes Project Data Transformation

In this chapter we delve into the details of how to obtain a GDM representation of the data source 1000 Genomes from a logical standpoint. For all classes of files, we describe the transformations applied, highlighting the key points and comparing each input before and after the transformation. We see, among the others, a novel strategy for converting the genomic coordinates of the regions from 1-based to 0-based, the final data schema and the metadata attributes generated for each sample. Finally, the last part of the Chapter gives some statistics about the dataset generated from the transformation process.

6.1 1000 Genomes Project region data transformation

The transformation of the source region data go through several intermediate steps which are well-detailed in the following sections. The explanation of the methods is interleaved with examples of mutations inspired from real VCFs and eventually shortened by removing the attributes which are not relevant for the discussion.

6.1.1 Overview

The first thing to notice about the transformation of the region data is the difference in the cardinality of the input dataset with respect to the expected output. We know that in a GDM dataset, the region data is grouped by sample, therefore we expect a number of region files equivalent to the number

of samples in the source. Each dataset originating from the 1000 Genomes Project contains the mutations of approximately 2.5 thousands of samples distributed in 23/25 files¹. Given that, the transformation of the source 1000 Genomes is a so-called "many to many transformation"². Indeed from a region source file we obtain 2.5 thousands output files, but each output file is originated from the union of the data contained in all the source region files.

Such characteristic has important consequences on the implementation of the transformation, as we will discuss in Section 7.3, on the method used for writing the output files and also it implies that we must convert the whole dataset in order to get the output regions of any sample³. Similarly we will need to convert the entire dataset before we can compute any meta information related to the samples, such as the size of the region file, the total number of mutations per-sample et simila.

6.1.2 Splitting of multiallelic mutations

The VCF format allows the expression of multiallelic sites in a single row, thus combining together different alternative alleles and their respective properties. Considering one of the simplest cases from the hg19 dataset, the example in Table 6.1 exemplifies two simple nucleic polymorphisms occurring at position 66 of the mitochondrial chromosome.

Table 6.1: Simplified example of multiallelic variant in mitochondrial chromosome without the columns ID and FORMAT. Columns are separated by an indentation space, INFO and FORMAT attributes are separated by a semi-colon, alternative alleles and multi-valued attributes use instead a comma as separator.

#CHROM	POS	REF	ALT	INFO	HG02028	NA20888
MT	66	G	T,A	VT=S,S;AC=4,1	2	1

In the first case, a Guanine is substituted with a Thymine, while in the second case the alternative allele is instead Adenine. Such duality of event is noticeable also in the attributes VT (variant type) and AC (allele count): the former tells that both events are of type S (abbreviation of SNP) and the latter informs that the first event has been observed 4 times in the set of samples analysed, while the second event just once. Finally, the genotype

¹ Depending on the considered dataset, we have 23 VCFs of mutations aligned on the assembly grch38 and 25 VCFs aligned on hg19.

² We will refer to this concept also later in this document with the term "many to many transform" or similar.

³ Which being a very high computational demanding process, it may take days.

6.1. 1000 Genomes Project region data transformation

information indicates that the first event appears in the sample NA20888 and the second appears in the sample HG02028.

In principle the above example can be rewritten equivalently as a couple of biallelic variants like follows:

Table 6.2: Biallelic representation of the Variant from Table 6.1

#CHROM	POS	REF	ALT	INFO	HG02028	NA20888
MT	66	G	T	VT=S;AC=4	0	1

#CHROM	POS	REF	ALT	INFO	HG02028	NA20888
MT	66	G	A	VT=S;AC=1	1	0

After splitting we can treat derived mutations (like the ones in Table 6.2) individually, allowing a 1-to-1 transformation from a mutation in the source to a GDM region.

Apart this simple example, things are actually more complicated in practice because the method for splitting can vary on a per-attribute basis and depends on the cardinality defined in the Meta-information section⁴, whose options are illustrated in Table 6.3.

Table 6.3: Descriptors of the cardinality of attributes. The cardinality "unbounded" can possibly mimic the rules expressed in A, R or G, therefore it is not possible to split multi-valued attributes correctly without knowing the format specification in advance.

Value	Meaning	Handling rationale
<i>number</i>	Cardinality fixed to the value of <i>number</i>	The attribute value is replicated in all derived mutations without changes
A	One value per alternative allele	Each derived mutation takes one value
R	One value for each allele (includes alternatives and reference)	Each derived mutation takes both the value associated to the reference allele and the value associated to the current alternative allele

⁴ This part contains optional information about the VCF itself (like date and format version), the description of the entries used in the INFO, FILTER and FORMAT columns as well as other details. In every VCF it occupies the very first lines, identified by the prefix ##.

Value	Meaning	Handling rationale
G	One value for each genotype (more relevant to the attributes of class FORMAT) ⁵	The value corresponding to the sample is replicated in the output mutation for this sample
.	Unknown/Unbounded	The attribute value is replicated in all derived mutations without changes
0	Attribute used as Boolean flag. If present it is true, false otherwise	The attribute value is replicated in all derived mutations without changes

The rules in Table 6.3 are valid for all the attributes in columns **INFO** and **FORMAT**. The ID value (3rd column in any VCF) has not a cardinality definition in the Meta-information section, however it is unbounded by specification⁶, so it is not possible to split the IDs in the derived mutations. For this reason, its value is simply replicated in all derived mutations.

6.1.3 Conversion of genomic coordinates

During the transformation of the source regions, two genomic coordinate systems are used, indeed mutation coordinates are 0-based in GDM, and 1-based in VCF. The Listing 6.1 shows the conceptual difference between the two systems by representing the SNP of a Cytosine into Adenine:

Listing 6.1: Graphical representation of a SNP with respect to 0-based and 1-based coordinate systems.

Position 0-based	0	1	2	3	4	5
Position 1-based		1	2	3	4	5
Reference	A	G	T	C	T	
Mutation	A	G	T	A	T	

The coordinate system used is reflected in the textual representation of a mutation, as exemplified in the Listing 6.2 which compares the two forms.

⁵ Considering a haploid call, the cardinality of G is the same of R because it is equal to the number of alleles (including the REF). If we consider a diploid call, then an attribute of cardinality G will have one value for each combination of alleles (3 values). For more general diploid calls, G has cardinality 3 to the power of the number of alleles.

⁶ The latest release of the format is the 4.3 and it is maintained by samtools since version 4.1. Full format specifications are available at <https://samtools.github.io/hts-specs/>.

6.1. 1000 Genomes Project region data transformation

Listing 6.2: Example of SNP in 0-based coordinates

	START	END	REF	ALT
In 1-based coordinates	4	4	C	A
In 0-based coordinates	3	4	C	A

For SNPs and MNPs, the conversion to 0-based coordinates is as simple as subtracting a unit from the 1-based starting position and leaving the end position equal to the original.

Insertions are special cases tough, both because in 0-based coordinates the start and end position must have the same value and because, in VCFs, INDELs and SVs need to specify a "padding base", i.e. a nucleotide common to both REF and ALT alleles and located before or after the mutated nucleotides. Consider indeed the simple example shown in Listings 6.3 and 6.4, the latter showing the alignment of the reference and alternative alleles relatively to the 1-based and 0-based coordinates.

Listing 6.3: Generic mutation showing the insertion of a Thymine after a nucleotide Adenine in 1-based coordinates.

CHR	POS	REF	ALT
2	19	A	AT

The example shows that, being the coordinate in POS always relative to the reference genome, in VCF there would be not any other way to express an insertion without the padding base (the T has not a corresponding position in the reference genome).

Listing 6.4: Alignment of the reference sequence with the sampled one showing the insertion of a Thymine in 1-based coordinates.

Coordinates (0-based)	18	19	20	21
Coordinates (1-based)	18	19	20	21
	x	A	x	x
	x	A T	x	x

In 0-based coordinates, however there is no such ambiguity in the representation: the Thymine is inserted precisely at the 19th position. So the padding base becomes a source of error, other than unnecessary. To solve this is-

Listing 6.5: Insertion of a Thymine after a nucleotide Adenine in a generic 0-based coordinate format.

CHR	START	END	REF	ALT
2	19	19	—	T

sue, the same mutation in 0-based coordinates must appear as in Listing 6.5 where the padding base has been removed, coordinates have been converted and the right end has been set equal to the left end.

Coordinates in deletions are obtained similarly to insertions, with the only difference being that the right end matches the end of the original mutation.

Structural variants are treated differently depending on the alternative allele type. If the alternative allele is a symbolic allele⁷, then only the REF allele contains a padding base. In this case, the extremities of the transformed variant are set equal to the original, and the padding base is removed. Otherwise, the structural variant represents a long insertion or a long deletion and it is treated like a normal insertion or deletion. This strategy is applied also to structural variants representing mobile element insertions using symbolic alternative alleles. The Table 6.4 shows an example of genomic coordinates conversion from 1-based to 0-based in each of these cases.

Table 6.4: At the top a SV with a symbolic alternative allele, in the middle a "normal" SV where "the string . . ." symbolizes a long sequence of nucleotides, and at the bottom the insertion of a transposon element.

In 1-based coordinates				In 0-based coordinates			
START	END	REF	ALT	START	END	REF	ALT
23	1823	C	<CN2>	23	1823	-	<CN2>
641	695	GT...TA	G	641	695	T...TA	-
351	670	C	<INS:ALU>	351	351	-	<INS:ALU>

The correctness of the method, considering both the transformation of genomic coordinates and the removal of the padding base, is validated by

⁷ Symbolic alleles use an alias to represent longer variants instead of writing the sequence of nucleotides. They are recognisable by the trailing and leading < > symbols in the ALT column. Very often, symbolic alleles are used to express copy number variants. For example <CN2> symbolizes a copy number variation of type 2.

6.1. 1000 Genomes Project region data transformation

a comparison with the well known UCSC Genome Browser [12]. The application developed by the University of California, Santa Cruz were tested on a set of test cases and the result shows that our method computes the same coordinates and the same set of ALT and REF values for all kinds of mutations.

6.1.4 Reduction to minimal-form

Sometimes, mutation splitting has the side effect of generating redundancy in the fields ALT and REF, especially in short mutations due to unnecessary padding bases that are typically used in VCF. Consider for example the mutation reported in Listing 6.6 expressed in 1-based coordinates.

Listing 6.6: Multiallelic mutation with redundant bases.

CHR	START	REF	ALT
MT	42	TCC	CCC,T

The first mutation represents the substitution of a single nucleotide (SNP), however the reference and alternative alleles are longer than usual because in the same locus we have also the deletion of two Cytosines, as represented in the second mutation. After splitting however, the SNP is no more in minimal form and it also generates an inconsistent representation, especially when the right end and length⁸ values are made explicit (see Listing 6.7).

Listing 6.7: This mutation is not minimal because it uses three bases to represent the substitution of a single nucleotide. Also it is inconsistent because a SNP has length 1 by definition while in this example it has length 3.

CHR	START	END	LENGTH	REF	ALT
MT	41	43	3	TCC	CCC

The correct and minimal representation of the above SNP is instead shown in Listing 6.8.

Listing 6.8: Minimal representation of a SNP. It has length 1 and both REF and ALT alleles has only one base.

CHR	START	END	LENGTH	REF	ALT
MT	41	41	1	T	C

⁸ In 1-based coordinates, the length value is computed as $|LENGTH| = END - START + 1$.

Chapter 6. 1000 Genomes Project Data Transformation

Non minimal forms are found in every kind of mutation, not just SNPs. The Listing 6.9 shows the split and minimal form reduction of a multiallelic mutation with 5 alleles of heterogeneous types.

Listing 6.9: Complex multiallelic mutation of chromosome MT involving 2 SNPs and 3 INDELs.

```
#INPUT
CHR      START  END      REF      ALT
MT        55     60      TATTTT   T,CATTTT,AATTTT,TTT,TTTTT
                                   (1), (2), (3) (4), (5)

#OUTPUT
CHR      START  END      REF      ALT
MT        56     60      ATTTT    -        (1)
MT        55     55      T        C        (2)
MT        55     55      T        A        (3)
MT        56     60      ATTTT    TT       (4)
MT        56     60      ATTTT    TTTT     (5)
```

The previous example highlights also a limitation. Even if the output mutations with numbers 4 and 5 have **REF** and **ALT** alleles terminating with a sequence of Thymines, they are anyway considered minimal because VCF does not provide the equivalence among bases [6], meaning that in certain cases we cannot distinguish precisely which base from the reference has a correspondence in the alternative sequence. This concept is illustrated in Listing 6.10 where the mutation number 5 is represented graphically in 5 valid equivalent alignments.

Listing 6.10: Graphical representation of equivalent possible alignments resulting from a non-minimal deletion in VCF.

```
padding
      55  |   56  57  58  59  60  Event
REF  T   |   A   T   T   T   T
ALT  T   |   -   T   T   T   T   deletion of A
      T   |   T   -   T   T   T   SNP A->T & deletion of T
      T   |   T   T   -   T   T   SNP A->T & deletion of T
      T   |   T   T   T   -   T   SNP A->T & deletion of T
      T   |   T   T   T   T   -   SNP A->T & deletion of T
```

6.1. 1000 Genomes Project region data transformation

For this reason, we preferred to keep the most conservative approach, avoiding to minimize further the sequences.

This ambiguity is reflected in mutations number 4 and 5 where we avoid to minimize further the sequences and instead adopt the a conservative approach which reflects in sticking to the original as much as possible but also maintaining a uniform treatment of INDELs which consists in the removal of only the padding base.

6.1.5 Computing the length of a mutation

The main challenge related to computing the length for the output regions derives for the most part either by the variety of mutation kinds, either by the uncertainty in the alignment of some variants. All these considerations reflect in the possibility to compute multiple values of length, all of them still correct depending on the interpretation⁹. The final strategy used to compute the length is a trade-off between the will to maximize the informative content and the need for an easy, intuitive rule which supports the consistency between the concepts of genomic coordinates and length of a mutation. In practice, this is achieved by separating the variants into two categories, **non-insertions** and **insertions**.

Since we want to compute the length for the output regions, in this section we make use of examples and equations expressed in 0-based coordinates as in the output.

The first category (**non-insertions**) is very broad and includes SNPs, MNPs, deletions, non-symbolic SVs and CNVs. This category is the most-consistent from a purely metric standpoint as the length is the result of the simple equation $length = end - start$.

The second category (**insertions**), as the name suggests, includes only short insertions and the group of ME insertions. For the variants in this category we consider the length as a separate concept with respect to that of genomic coordinates, allowing us to compute the length as the number of bases inserted, while representing the insertion event with a pair of genomic coordinates of equal value. In this way, insertions are expressed with an unambiguous code, but still we provide the real length independently from the value of the start or end coordinates.

Example cases for the two groups are illustrated in Table 6.5 where the mutations are represented with respect to the 0-based coordinate system,

⁹ The interpretation of the length can vary from a mutation type to another. For example one thinks of what is the length of a MNP and that of an insertion comparing the algorithms and the distance between the genomic coordinates.

Chapter 6. 1000 Genomes Project Data Transformation

as in the output regions, while the attributes relevant for the discussion are copied as in the original mutation into the **ATTRIBUTES** column.

Table 6.5: Application of the method computing the length for mutations belonging to the non-insertions and insertions groups. Further explanations are given in the COMMENT column for the edge cases. Note that the mutations are listed in 0-based coordinates, except for the values in column ATTRIBUTES which are copied from the origin.

START	END	REF	ALT	ATTRIBUTES	COMMENT	LENGTH
60071	60072	G	C		SNP. The length is always 1.	1
60034	60036	CC	-		Simple deletion of length 2.	2
60034	60036	CC	C		A one base deletion spanning an interval of 2 bases. We consider the length of the interested genomic area, so the length of REF.	2
187604	218561	-	<CN0>	END=218561	Deletion of large genomic area. We use END to compute the length as END - START.	30957
16942602	16945851	-	<CN0>	SVLEN=3483 END=16945851 CIPOS=-225,0 CIEND=0,367	The mutation is imprecise (even if no IMPRECISE flag is present). Consequently SVLEN and END are approximations. If we compute the length as END - START, it is still an approximation, but it is also methodologically coherent with the other SVs. Plus, by including CIPOS and CIEND in the output region, there is no information loss as it is always possible to compute other estimations of the value.	3249
60020	60020	-	AAC		Insertion of 3 bases.	3
16918023	16918023	-	<INS:ME:SVA>	SVLEN=1312	The original mutation does not provide the STOP coordinate, instead it has only the length in 1-based coordinates which must be converted considering the removal of the padding base.	1311
471635	471635	-	<INS:ME:ALU>		The original mutation does not provide neither the length, neither the STOP coordinate.	NULL

6.1. 1000 Genomes Project region data transformation

6.1.6 Decoding of the genotype string

The genotype string¹⁰ tells how many times a mutation is present in a sample (at most 2 in diploids, and at most 1 in haploids). Moreover, since genotypes in 1000 Genomes are so-called "phased", we can also know whether two distinct mutations occur on the same chromosome copy. All of this is encoded in a string whose possible values¹¹ are shown in Listing 6.11:

Listing 6.11: Values of genotype string in biallelic variants

In diploid variant calls:

```
0|0    -> no mutation found
1|0    -> mutation found only on the chromosome copy #1
0|1    -> mutation found only on the chromosome copy #2
1|1    -> mutation found on both chromosome copies
```

In haploid variant calls:

```
0      -> no mutation found
1      -> mutation found
```

The genotype information is conserved with the same values (0 and 1) in the output regions into separate columns **AL1** and **AL2**, corresponding respectively to the chromosome copies #1 and #2. Conventionally haploid calls assign the only available genotype value to the column **AL1** while **AL2** is set to "NULL".

Finally, the examples represented in Listing 6.11 reports only the values of **AL1** and **AL2** for a biallelic mutation. In VCF these attributes can assume any semi-positive integer value. However, as shown before in Section 6.1.2, we always split multiallelic mutations in a set of biallelic ones, so we can reduce all the assignments of **AL1** and **AL2** to those illustrated in Listing 6.11.

6.1.7 Recognition of the type of mutation

Even if the mutation type is already expressed in the original VCF through the attributes¹² **VT** and **SVTYPE**, insertions and deletions are grouped into a

¹⁰ In VCFs the genotype string is the first value of every column corresponding to a sample (10th column and subsequents).

¹¹ Actually there are also other possible values of the genotype string, however we are able to reduce them to the few ones listed in the example. See the end of this section for further details.

single category generically called "INDEL". In order to separate the two events, we need to compute the type manually from the properties of the mutation. As shown in Table 6.6, identifying the the class of a short variant is as simple as comparing the REF and ALT attributes:

Table 6.6: Rationale for identifying the type of a short variant

REF	ALT	TYPE	RATIONALE
C	A	SNP	They both have length 1.
CT	AC	MNP	They have equal length > 1.
ACTG	C	Deletion	REF is longer than ALT.
-	CGC	Insertion	ALT is longer than REF.

To identify structural variants instead we rely on the values of the attributes provided by the original mutation: by default we assign `mut_type` to the value of `SVTYPE`. However, since mutations with alternative allele "<CN0>" in the source regions are sometimes encoded with `SVTYPE=DEL`, sometimes with `SVTYPE=CNV` we eventually replace the value of `mut_type` with "DEL" whenever the ALT matches the string "<CN0>" to increase coherency in the expression form.

6.1.8 Format of output regions

Each single output region is characterized by a set of attributes. The Table 6.7 summarizes the attributes used (2nd column) as well as the corresponding attribute from the source (3rd column) and how the first is obtained from the last one. The selection and ordering of the output fields is driven by the region output schema illustrated in Listing 6.12 which is applied for the transformation of the datasets hg19 and grch38.

Table 6.7: List of all the attributes characterizing the the output transformed regions.

	Output attribute	From input	Transformation details
1	chr	$f(\text{CHR})$	Concatenation of the string "chr" with the value of CHR.

¹²Optional attributes located in the column INFO. More precisely, VT distinguish between SNP, MNP, INDEL and SV. The Sv category is then further classified into CNV, DEL, INS, ALU, SVA or LINE1 only if the variant is a structural variant.

6.1. 1000 Genomes Project region data transformation

	Output attribute	From input	Transformation details
2	left	$f(\text{POS}, \text{REF}, \text{ALT})$	See Sec. 6.1.2 to 6.1.4.
3	right	$f(\text{left}, \text{ref}, \text{alt}, \text{END}, \text{SVLEN}, \text{SVTYPE})$	See Sec. 6.1.2 to 6.1.4.
4	strand		Always set to "+" ¹³ .
5	AL1	$f(\text{GT})$	See Sec. 6.1.6.
6	AL2	$f(\text{GT})$	See Sec. 6.1.6.
7	ref	$f(\text{REF}, \text{ALT}, \text{SVTYPE})$	See Sec. 6.1.3 and 6.1.4.
8	alt	$f(\text{REF}, \text{ALT}, \text{SVTYPE})$	See Sec. 6.1.2 to 6.1.4.
9	mut_type	$f(\text{VT}, \text{SVTYPE}, \text{REF}, \text{ALT})$	See Sec. 6.1.7.
10	length	$f(\text{left}, \text{right}, \text{ref}, \text{alt}, \text{SVLEN}, \text{CIPOS})$	See Sec. 6.1.2 to 6.1.5.
11	id	$f(\text{ID})$	See Sec. 6.1.2.
12	quality	QUALITY	Copied from source.
13	filter	FILTER	Copied from source.
14	DP	$f(\text{DP})$	See Sec. 6.1.2.
15	AF	$f(\text{AF})$	See Sec. 6.1.2.
16	AC	$f(\text{AC})$	See Sec. 6.1.2.
17	AFR_AF	$f(\text{AFR_AF})$	See Sec. 6.1.2.
18	AMR_AF	$f(\text{AMR_AF})$	See Sec. 6.1.2.
19	EUR_AF	$f(\text{EUR_AF})$	See Sec. 6.1.2.
20	EAS_AF	$f(\text{EAS_AF})$	See Sec. 6.1.2.
21	SAS_AF	$f(\text{SAS_AF})$	See Sec. 6.1.2.

¹³ All mutations from 1000 Genomes Project are reported on the positive strand as documented in <https://www.internationalgenome.org/faq/what-strand-are-variants-your-vcf-file/>.

Chapter 6. 1000 Genomes Project Data Transformation

	Output attribute	From input	Transformation details
22	AA	$f(\text{AA})$	See Sec. 6.1.2.
23	IMPRECISE	$f(\text{IMPRECISE})$	See Sec. 6.1.2.
24	CIEND	$f(\text{CIEND})$	See Sec. 6.1.2.
25	CIPOS	$f(\text{CIPOS})$	See Sec. 6.1.2.
26	TSD	$f(\text{TSD})$	See Sec. 6.1.2.
27	MEINFO	$f(\text{MEINFO})$	See Sec. 6.1.2.
28	MC	$f(\text{MC})$	See Sec. 6.1.2.
29	EX_TARGET	$f(\text{EC_TARGET})$	See Sec. 6.1.2.
30	MULTI_ALLELIC	$f(\text{MULTI_ALLELIC})$	See Sec. 6.1.2.
31	OLD_VARAINT	$f(\text{OLD_VARIANT})$	See Sec. 6.1.2.

6.1. 1000 Genomes Project region data transformation

Listing 6.12: Output region schema

```
<?xml version="1.0" encoding="UTF-8"?>
<gmqlSchemaCollection name="one_k_genomes"
  xmlns="http://genomic.elet.polimi.it/entities">
  <gmqlSchema type="TAB" coordinate_system="0-based">
    <field type="STRING">chr</field>
    <field type="LONG">left</field>
    <field type="LONG">right</field>
    <field type="CHAR">strand</field>
    <field type="INTEGER">AL1</field>
    <field type="INTEGER">AL2</field>
    <field type="STRING">ref</field>
    <field type="STRING">alt</field>
    <field type="STRING">mut_type</field>
    <field type="LONG">length</field>
    <field type="STRING">id</field>
    <field type="DOUBLE">quality</field>
    <field type="STRING">filter</field>
    <field type="LONG">DP</field>
    <field type="DOUBLE">AF</field>
    <field type="INTEGER">AC</field>
    <field type="DOUBLE">AFR_AF</field>
    <field type="DOUBLE">AMR_AF</field>
    <field type="DOUBLE">EUR_AF</field>
    <field type="DOUBLE">EAS_AF</field>
    <field type="DOUBLE">SAS_AF</field>
    <field type="STRING">AA</field>
    <field type="STRING">IMPRECISE</field>
    <field type="STRING">CIEND</field>
    <field type="STRING">CIPOS</field>
    <field type="STRING">TSD</field>
    <field type="STRING">MEINFO</field>
    <field type="STRING">MC</field>
    <field type="STRING">EX_TARGET</field>
    <field type="STRING">MULTI_ALLELIC</field>
    <field type="STRING">OLD_VARIANT</field>
  </gmqlSchema>
</gmqlSchemaCollection>
```

Finally when all the values are ready, i.e. computed or transformed, they are concatenated in a single string using a single tab character as value separator, as indicated also in the string `gmqlSchema type="TAB"` in the 4th line of the schema file.

Moreover, the schema controls also how missing values should be treated: if the field is of type string (`field type="STRING"`) an empty string is used in place of the value that is missing; in all the other cases, a string with value "NULL" is used instead.

The Appendix B shows some example mutations transformed according to the method here described, and comparing input and output.

6.2 1000 Genomes Project metadata transformation

As shown in Section 5.2, the metadata from 1000 Genomes are encoded into text files (4 per dataset) whose layout resemble that of a generic TSV¹⁴ format. Moreover, this is a many-to-many transformation, exactly as for region data. In general the transformation requires to filter out headers, if present, and reading the file line by line. Then, using a regular expression to separate the values, is possible to build a map/dictionary of the values, indexed by a key, which is usually the sample name.¹⁵ In the next sections we show how such maps are created for groups of metadata.

6.2.1 Family ID and Gender

The **Family ID** and **Gender** values are located in the metadata file of class *individual details* as the 1st and 5th columns. Being the file organized as a simple TSV, we just need to skip the first line (containing the name of the columns) and decode each line into a list/array of values obtained by dividing the original string at the occurrence of the character `\t`. We can reduce the size of the list/array by ignoring the non-relevant columns, except for the 2nd, named **Individual ID**¹⁶, needed to relate the **Family ID** and **Gender** values to each sample¹⁷. Since the gender value is encoded in the

¹⁴Tab separated values

¹⁵This is not the only possible way, but in the next Chapter, about the modules implementation, we will discuss the benefit of this method.

¹⁶An alias for the sample name.

¹⁷A manual check on the data confirmed that the values in **INDIVIDUAL ID** are unique, therefore it can be used as key for an indexed data structure.

6.2. 1000 Genomes Project metadata transformation

origin file as 1 for males and 2 for females, we transform it into "male" and "female" respectively.

Table 6.8: Example of map generated by the transformation of metadata file of class `individual details`. The 1st column is used as key to find the attributes of the sample. The example shows two individuals belonging to the same family and an unrelated sample.

Sample	Family ID	Gender
HG00144	GBR001	female
HG00124	HG00124	female
HG00155	GBR001	male

6.2.2 DNA Source from Coriell

The attribute **DNA Source from Coriell** is the 60th column from the metadata class `sample_info` which is formatted as a standard TSV file. Again, as for `individual details` we skip the header (first row of the file) and then build a map of the values of **DNA Source from Coriell**. For each value, the key is the sample name, provided in the 1st column of the same file. Missing values do not generate this metadata in the output.

Table 6.9: Example of map generated by the transformation of metadata file of class `sample info`.

Sample	DNA source from Coriell
HG00288	LCL ¹⁸
HG00733	Blood

6.2.3 Experiment, Population, Analysis group and others meta

From a metadata file of class `sequence index` we extract metadata values shown in Table 6.10, indexed by **SAMPLE_NAME**. The process is similar to what already seen for `individual details` with two main differences.

First of all, the file requires different preparatory operations depending on the dataset being transformed. Indeed in dataset hg19, the origin file presents just one extra line at the top for the header, while in grch38 we have 29 lines to skip before reaching the "body" of the file. But more importantly, the data

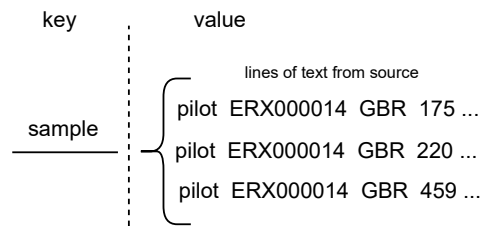
¹⁸LCL stands for Lymphoblastoid Cell Line. Cell lines used by the 1000 Genomes Project are obtained from samples of B cells treated with Epstein-Barr virus.

Table 6.10: List of the selected attributes (or categories).

STUDY_ID	4 th column
STUDY_NAME	5 th column
CENTER_NAME	6 th column
SAMPLE_ID	9 th column
SAMPLE_NAME	10 th column
POPULATION	11 th column
EXPERIMENT_ID	12 th column
INSTRUMENT_PLATFORM	13 th column
INSTRUMENT_MODEL	14 th column
INSERT_SIZE	18 th column
LIBRARY_LAYOUT	19 th column
ANALYSIS_GROUP	26 th column

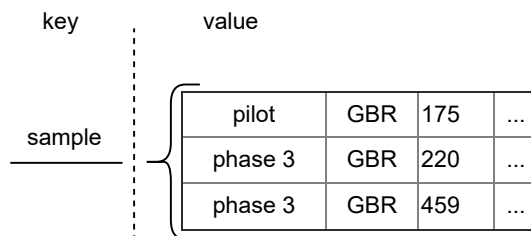
in this file is indexed by sequence file (1st column) instead of **SAMPLE_NAME**. This means that we can have different tuples of the values in Table 6.10 for the same sample. To address this issue we use the following strategy:

1. We build a map indexed by **SAMPLE_NAME** collecting the lines of text related to each sample. This works similarly to grouping the rows of a database. So each sample in the map will have a set of Strings.

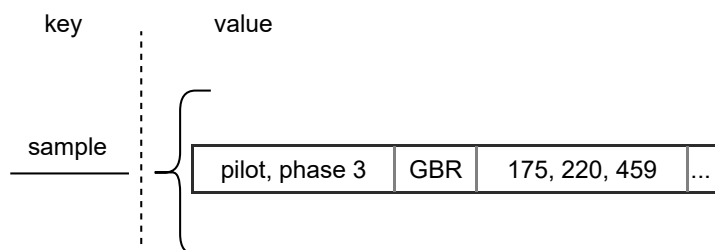


2. Considering a single sample and its lines, we keep only the relevant attributes listed in Table 6.10 and discard the others. We do this by transforming the list of rows (i.e. the value of the map) into a list containing equally-sized lists of Strings, each one containing a single attribute. We repeat this process for each sample into the map.

6.2. 1000 Genomes Project metadata transformation



3. Finally, we transform back the value of the map to a list of strings, where each item is the reduction of many attributes of the same kind contained in all the lists of the sample. In practice, the reduction process operates on values of the same category and removes duplicate values before joining them into a single string of comma-separated values. By repeating this step for each sample, we get a map indexed by `SAMPLE_NAME`, having as value a list of strings where each string is the set of unique values associated to the sample for one category.



We transform all the attributes of Table 6.10 with the above method, except for the attribute `ANALYSIS_GROUP`, for which instead we decided to generate in output as many `<key, value>` pairs as the number of unique values.

6.2.4 Super-population, DNA from Blood

`Super-population` and `DNA from Blood` are part of the metadata of class *population*. To extract the relevant attributes from this metadata source we process the file line by line and keep only the ones matching the regular expression:

```
.*\t(\S+)\t(\S+)\t(yes|no)\t(yes|no)\t\d+\t\d+\t\d+\t\d+
```

Basically, this regular expression matches with the "body" of the file while rejecting the header, some empty rows and a line telling summary statistics

located toward the end of the file. This regular expression already isolates some of the attributes for us, returning a list of strings with only the values in between parenthesis, which are listed below:

Super-population	2 nd pair of parentheses.
DNA from Blood	3 rd pair of parentheses.
Population	1 st pair of parentheses. We use this as index.
Offspring available	4 th pair of parentheses. This group is functional to the correctness of the regular expression but it is not used in practice.

Note that the attributes in this file are indexed by **Population**, so we use this as key for the map that we are going to generate. Finally, before building the map¹⁹, we transform the values of the field **DNA from blood** from a "yes"/"no" to a "true"/"false".

Table 6.11: Example of dictionary generated from the population metadata (without "Offspring available").

Population Code	Super-population	DNA from blood
TSI	EUR	false
GBR	EUR	true
PUR	AMR	true

6.2.5 Manually curated metadata

Not all the information we have on the source comes from data files. Some of the knowledge is context-derived, some computed by algorithms, some come from papers and other material documenting the progress of the 1000 Genomes Project. We group such kind of information into the so called "manually curated" category of metadata and every information is rep-

¹⁹One could propose also to hard-code the values of this file into the program because it have few values which depend only on the population (which are only 26). Of course it is an alternative worthy of consideration because these values are not supposed to change in future, Nevertheless, the development of a method for the automated data extraction for this file requires little effort and is better suited to a program supposed to reflect the source and automatically update the repository.

6.2. 1000 Genomes Project metadata transformation

resented as a pair $\langle key\ value \rangle$ where the *key* has always prefix "manually_curated__".

Precisely, the $\langle key\ value \rangle$ pairs belonging to this category are listed in Table 6.12 (without the prefix for reasons of compactness).

Table 6.12: List of manually curated metadata.

<i>key</i>	<i>value</i>
assembly	"hg19" or "grch38"
data_type	"variant calling"
feature	"variant"
file_format	"bed"
file_name	Name of the sample concatenated with ".gdm"
is_healthy	TRUE
pipeline	Read from the project documentation and saved into a string into the configuration file
project_source	"1000 Genomes"
source_page	List of the source files containing the regions for the sample
species	"Homo Sapiens"
chromosome	List of the chromosomes in which are located the mutations of the sample
local_file_size	The size of the output region file of the sample
local_md5	The md5 checksum of the output region file of the sample
origin_last_modified_date	Last update date of the remote dataset

In addition, for a sample and every source region file related to it, we write the following metadata (Table 6.13).

Table 6.13: Manually curated metadata added for each source region file contributing to the sample.

```
download_date
origin_md5
origin_file_size
data_url
```

The nature of the metadata `source_page`, `chromosome`, `local_file_size`, `local_md5` and the ones in Table 6.13 imposes that such attributes are written only after all the regions of the sample have been transformed.

6.2.6 Format of output metadata

Output metadata files are generated for each sample by collecting and writing all the values available from the in-memory data structures, after being prepared as described in Sections 6.2.1 to 6.2.5.

Most of the metadata attributes are indexed by sample for our obvious convenience, except the metadata expressing the super-population and "DNA from Blood" values which are indexed by population code. Consequently, before writing the metadata for a sample, we need to fetch the population code from the *sequence index* (of course we can just read the corresponding map built in Section 6.2.3) and use it as index into the map of the *population*.

When all the metadata of a sample are available²⁰ we write them one per line into a `<sample>.gdm.meta` file using a tab character as separator between the key and the value. Long multi-word keys are written by replacing any white space with an underscore symbol. The Table B.3 in Appendix B collects all the output metadata attributes discussed and shown throughout this Section (subsections 6.2.1 to 6.2.5) and provide for each an example of value.

6.3 Generated datasets

Applying the method of transformation described in this Chapter, we have successfully converted all the two datasets available for 1000 Genomes Project, one containing the genomes of 2535 donors with variants aligned on the assembly hg19, and one containing the genomes of 2548 donors with variants

²⁰Remember that a part of the manually curated metadata describe properties which are known only after all the region data of the same sample have been transformed and written (See Section 6.2.5).

6.3. Generated datasets

aligned on the assembly grch38. The output of the transformation was the generation of two corresponding GDM datasets:

- HG19_1000GENOMES_2020_01
- GRCh38_1000GENOMES_2020_01

6.3.1 Test performed

The correctness of the transformed data has been verified in three ways:

1. During development, at every step we tested the code generating the output mutations with inputs both sampled from the original source files and with artificially created inputs in order to evaluate the coverage of corner cases. These tests were targeted especially at assessing the correctness of the region and metadata values.
2. For a set of variants (large enough to include some examples of all kinds of mutations observed in the source), we also manually verified the correspondence of the coordinates of the output regions with the UCSC Genome Browser. [12]
3. During development we performed several runs on reduced versions of the two datasets with a small number of input files each and we verified the output log and files to check that the Transform stage executes correctly as a whole.

6.3.2 Statistics

Excluding short biallelic variants, VCF and GDM can be two very different formats to represent genomic variants. Indeed, the steps necessary to apply the conversion are many and quite complex overall, so the transformation took several days despite all the arrangements used to reduce the execution time. Also the size of the generated datasets is considerably greater than the source because the input format offers a greater compression ratio when describing variants that are common to large set of individuals. The details concerning the number of files transformed, the size of the datasets and the execution time are reported in tables 6.14 and 6.15.

The output datasets are accessible at <http://gmql.eu/> in the public datasets folder.

Table 6.14: Statistics on the input datasets.

dataset	input			
	region files	metadata files	size (.gz compressed)	size (extracted)
GRCh38	23	4	13 GB	752 GB
hg19	25	4	17 GB	796 GB

Table 6.15: Statistics on the output datasets.

dataset	region files	metadata files	output	
			size	execution time
GRCh38	2548	2548	1.1 TB	7 days 19 hours 59 min 45 s
hg19	2535	2535	1.5 TB	9 days 1 hours 7 min 40 s

Chapter 7

Implementation of Metadata-Manager modules for 1000 Genomes Project data source

Now that we have defined the files to import and the transformations to apply in order to make the source compliant with the GDM model, we can extend the framework Metadata-Manager and add the required functionalities to import the data from the 1000 Genomes Project. The implementation runs through seven steps: download, transformation, cleaning, mapping, normalization/enrichment, constraint checking, flattening and loading. As for the first stage, we initially developed the module **KGDownloaderA**; this module is responsible for creating a local data repository containing the source data and metadata while keeping the files updated with their most recent version available from the origin. For the next stage of the integration process we developed the module **KGTransformer** which takes as input the local repository and transforms the region data files and metadata files in GDM format. The process is coordinated by the framework which controls the files to process at each stage with the help of a database in which every file is enriched with details regarding the origin, the output dataset and its status. In this occasion, the framework has been minimally changed to address the specificity of the source 1KG and the ones with a similar data structure. The following stages were completed by appropriately configuring the framework to map the source to the GCM metadata model, facilitating its integration with other sources. Finally, the modules **KGDownloaderB** and **KGParallelTransformer** of have been developed subsequently for per-

formance and reliability purposes and can be used interchangeably with the previous ones by simply editing the configuration file.

7.1 Download of the source files

The download module of 1KG is made of three components created to separate the management of the involved resources into isolated containers. **DatasetInfo** is used to obtain a representation of the files needed by each source dataset. **FTPHelper** is a state-full class providing handful methods to explore and reliably download large files from any FTP server. **KGDownloaderA** manages the database representing the local source data repository and coordinates the operations of fetch, update and download of a dataset. The Components involved in this module are presented schematically below.

```
/src/main/scala/it/polimi/genomics/metadata/  
├── downloader_transformer/one_k_genomes/  
│   ├── DatasetInfo.scala  
│   ├── KGDownloaderA.scala  
│   └── KGDownloaderB.scala  
└── util/  
    ├── FileUtil.scala  
    ├── FTPHelper.scala  
    └── PatternMatch.scala
```

DatasetInfo

This Scala object integrates the knowledge of the source FTP server and its functions are fundamental to identify the files that are required for building the local repository but, most of all, to ensure the most recent version of the remote repository is selected on every update check. In order to identify the correct dataset, this component embeds three basic assumptions derived from the observation of the FTP source:

1. The region files hosted on the server are located according to a recurrent folder structure composed of a root FTP address, a dataset-specific relative path¹, a version name², and a file name.

¹ 1KG provides two GDM datasets of mutations, one aligned on the assembly hg19 and the other aligned on the assembly grch38.

² When datasets are still under development, such in the case of grch38, the updates can be distributed as corrections to one or more single files or as a complete new package, generating multiple versions of the same dataset each time.

7.1. Download of the source files

For example, the file containing mutations observed on chromosome 5 is located at path `ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/ALL.chr5.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz` for the dataset aligned on the assembly hg19, or at path `ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000_genomes_project/release/20190312_biallelic_SNV_and_INDEL/ALL.chr5.shapeit2_integrated_snvindels_v2a_27022019.GRCh38.phased.vcf.gz` for the dataset aligned on the assembly grch38. The path begins with the root of the FTP server, which is defined as

`ftp.1000genomes.ebi.ac.uk/vol1/`

It is then followed by a string which points to the root directory of each dataset:

- `ftp/release/` for the dataset hg19
- `ftp/data_collections/1000_genomes_project/release/` for the dataset grch38.

Continuing the analysis of the path, we find respectively the versions `20130502/` and `20190312_biallelic_SNV_and_INDEL/` for the two identified dataset, and the filenames which compose the remaining part of the addresses.

2. Major updates to each dataset are distributed as a completely new release - or version - of the dataset itself, whose files are contained in a folder named in chronological order and located inside the same directory hosting the old release directories.
3. The FTP server provides a record listing the last modified date, the size in bytes, the MD5 checksum, the file type (single file or directory) and the path relative to the root FTP server address for all the hosted files. We call this *tree file* for convenience.

Therefore, the method **latestVariantsRecords** can parse a *tree file* and return the records corresponding to the updated region files from the remote for a given dataset in the form of URL, last modified date and MD5 checksum. Since every record is a string containing heterogeneous information, **DatasetInfo** also defines the class **DatasetPattern** which helps in generating regular expressions to extract the attributes separately and to filter the variants according to the preferences defined in the following XML configuration parameters:

Chapter 7. Implementation of Metadata-Manager modules for 1000 Genomes Project data source

Figure 7.1: Representation of the hierarchical organization of dataset versions and files within each dataset root folder. For the sake of the completeness, the diagram represents also the displacement of other complementary files and also how the files are distributed between the hg19 dataset (in green) and the grch38 dataset (in red).

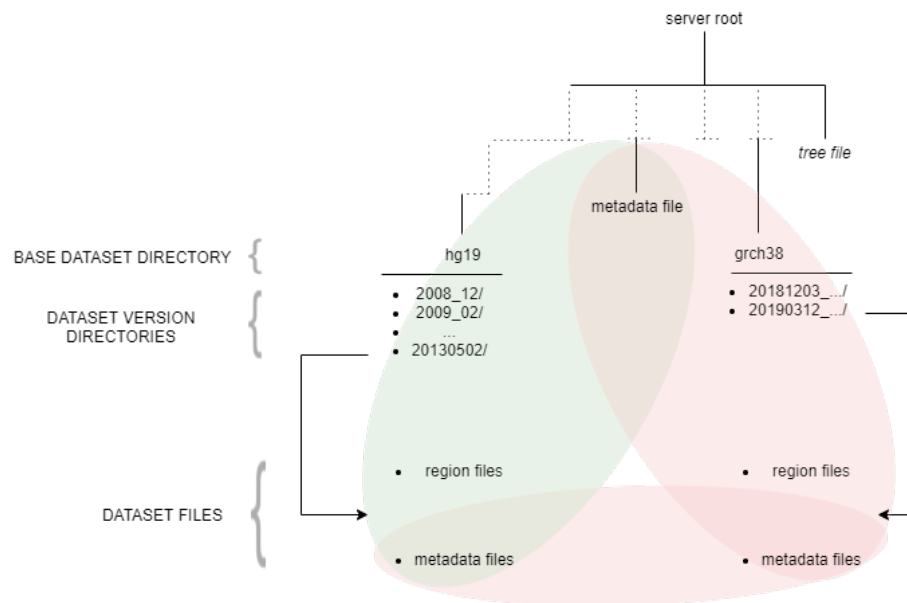


Figure 7.2: Content of the tree file for three example files. Each record shows in order the path relative to the root, the file kind, the size in bytes, the date of the last modification and the MD5 checksum separated by tabs. Directories are reported without the checksum.

```

1 ftp directory 4096 Thu Jun 19 09:04:12 2014
2 ftp/PRIVACY-NOTICE.txt file 414 Thu May 24 10:16:54 2018 75766262f08b901ac92fc751096db7be
3 ftp/CHANGELOG file 293659 Tue Sep 24 17:20:07 2019 44d3c01b8ab2b1fcfaa5a94478c835f7

```

- `filter_variants_starting_characters`

The characters a source region filename begins with.

- `filter_variants_ending_characters`

The characters a source region filename ends with.

- `exclude_subdirs_in_each_dataset_release`

A Boolean condition telling whether to limit the search space of the source region files to the directory of a dataset version.

- `filter_variants_with_custom_path_regex`

7.1. Download of the source files

A custom defined regular expression to be applied to the source region filenames only. If defined, it overrides all the other options.

Records are retrieved also for metadata in the method **metadataRecords** but in a different manner: as shown in the Section 5.2, metadata files for 1KG are few and scattered in different locations without showing a strong logical organization, so it is not possible to use an automated logic for finding them. Instead, we manually provide the path for these metadata files through the XML configuration file, which is in turn given as argument to MetadataManager. When **metadataRecords** is called, the following parameters are fetched from the arguments.

- `population_file_path`
- `individual_details_file_path`
- `samples_origin_file_path`
- `sequence_index_file_path`

Each value provides the URL relative to the server root for a metadata source file, which is used as search term in the *tree file* to get the last modified date, size and MD5 checksum before the download begins. In addition, the download module requires three more attributes to know the URL of the *tree file*, the root path of the FTP server and for each dataset, the base dataset directory (i.e. the directory containing one or more versions of the dataset). These attributes are given respectively as parameters having key

- `tree_file_url`
- `url_prefix_tree_file_records`
- `dataset_remote_base_directory`

FTPHelper

FTPHelper is a container for all the typical operations performed on a FTP server: download, server exploration, file search, file listing and reading the file properties. Of course the most important method of this class is **downloadFile** which in a single command connects to the server, performs the necessary authentication³, finds the desired file on the server and downloads it and makes sure the downloaded file exists in the target path by creating any intermediate directory. If a file with the same name exists at the target location, it is overwritten.

³ The authentication parameters are provided as XML configuration parameters with keys `FTP_username` and `FTP_password`.

KGDownloaderA

This is the coordinator of the download phase for 1KG. It directly extends the trait **Downloader** provided by the framework and coordinates the sequence of operations for downloading and updating the database, maintaining the local repository always in a consistent state. Whenever the **download** method is called by the framework, the flag COMPARE is applied to all the files in the dataset; it is a temporary status indicating that the file still needs to be checked against the remote repository. The *tree file* is downloaded from the remote and its MD5 checksum is compared with the one already stored in the database (if present) with the method **checkIfUpdateFile** from the class **FileDatabase**. If the comparison does not show any difference, the dataset is marked with the status UPDATED as it is already up-to-date and the process is completed. Otherwise, the new *tree file* is passed to the methods **fetchUpdatesForVariants** and **fetchUpdatesForMeta** which, respectively for each region and metadata, parse the MD5 checksum from the records and repeat the comparison on a per-file scale using the methods and classes of **DatasetInfo**. Files that were already present in the dataset but showing a different signature are re-downloaded and marked as UPDATED. New files that were not already part of the local dataset are downloaded too and registered in the database with the status UPDATED. If the download of a file fails, its database record is then marked as FAILED. Files already up-to-dated are simply marked as UPDATED in the database. At the end of this procedure, the ones still having the status COMPARE are the files that were present in a previous version of the remote dataset and eventually removed in a subsequent update; as such, they are marked for deletion with the status OUTDATED. Finally the status UPDATED is put on the *tree file* and a second update attempt is made for the files whose download has failed.

To maintain the database in a consistent state in case of an unexpected error condition, fatal exceptions are caught and left to the parent class only after the working dataset is marked with the status FAILED, thus indicating it should not be used in the next stages of the import process.

The methods shown on the FTP server hosted by EBI equally apply for the mirror FTP site located at NCBI (`ftp-trace.ncbi.nih.gov/1000genomes/ftp`). It is possible to switch from one to the other by just changing the XML configuration parameters referencing the path of metadata, tree file and base dataset folders. Keep in mind however, that the server hosted by NCBI currently lacks the dataset of mutations aligned on the assembly grch38.

7.1. Download of the source files

7.1.1 Resilience to network issues

1KG source region files have an average size of 671.33 MB in the dataset hg19 and 551.34 MB in the grch38 one, reaching up to 1.22 GB for single files. On an stable 20 Mbit/s connection it would require about 9 minutes for the largest file and less than 100 minutes for the entire hg19 dataset, but in practice it is not rare to observe execution times up to 7 hours when high traffic load is registered on the network or on the server; moreover, there is a number of network related issues which become more frequent as the download time increases. For example, sudden delays in the transmission of packets can interrupt a download several times before it is completed. Temporary network unavailability can occur as well and cause the connection to be lost. **FTPHelper** adopts two basic strategies for facing these issues.

First, by default, **downloadFile** makes three download attempts on every file, but the number of attempts increases every 200 MB if the caller can provide the expected size of the file being downloaded; in this way, the probability of a single transmission being interrupted is balanced by the greater number of attempts made. Also, every new attempt set the transmission stream to start from the last byte received, basically resuming the download from where it left off. The step size has been determined from empirical experiments, nevertheless is demonstrated to work in all test cases.

Secondly, the XML parameter **data_connection_timeout** regulates the maximum waiting time in milliseconds for the FTP data exchange socket to receive packets when the stream is interrupted. In networks affected by large and/or varying lag time, this timeout may be not enough and it is automatically increased by 500 ms on each following attempt to accommodate this condition. If instead the transmission is blocked by the host - i.e. when the server replies with the status code 421 - or the connection is lost for other reasons, a new attempt is repeated - if permitted - after a 1 minute delay to prevent the connection from being rejected immediately.

Despite the tuning of the parameters derives from empirical experiments, these techniques always allowed our tests to complete the download of a source datasets in a single run, even when such operation lasted for more than 6 hours under high network traffic conditions.

7.1.2 Reliability aspects

The internal logic of **KGDownloaderA** performs the checksum comparison in order to assess the equality between a file on the remote server and its local version. The presence of the *tree file* has a key role in this process because it allows our module to quickly compare dozens of very large files

without the need to download first the counterparts hosted on the remote. However, the availability of a data structure like the *tree file* is not part of any standard FTP protocol, rather an optional feature supported by the particular implementation running the server at this time. In view of the above, the sibling Scala class **KGDownloaderB** of package **one_k_genomes** can be used in place of **KGDownloaderA** achieving the same result with a different method. When **KGDownloaderB** is used, the client explores the server to find the most recent version of a remote dataset and lists the content of each directory; files and folders are filtered by name with the methods available from **DatasetInfo**⁴ and according to the same configuration parameters used for **KGDownloaderA**. When the files on the remote dataset are compared with the ones registered in the local database, the tuple <name, last-modified-date, file-size> is used as discriminating factor for detecting changes in any file that is already part of the local repository, thus using properties that are supported by all standard FTP protocols.

Overall, the download module for 1KG comes with two alternative strategies to balance the pros and cons of each strategy; indeed FTP was not designed to be used as a file synchronization protocol: it does not provide the file checksum on its own, the last-modified-date attribute could be set manually and the content of a file can be changed without affecting the original file size. However, given the context of application, both strategies represent a good solution to the original problem and they can be applied interchangeably by just modifying the parameter **downloader** in the XML configuration file. In conclusion, checksum comparison is the most reliable solution for detecting modifications and this makes **KGDownloaderA** the preferred choice. Still, the software supporting the 1KG server may be replaced over the time, so **KGDownloaderB** may become better suited if the *tree file* is removed or reformatted.

⁴ The methods **latestVariantsRecords** and **metadataRecords** have their counterparts in **latestVariantsFTPFile** and **metadataFTPFile** which work on **FTPFile** object instances, instead of the strings contained in the *tree file*.

7.2 Data transformation

The transformer module is the complex result of the input processing operations run by three main classes of the package **one_k_genomes** that contribute to the generation of the final GDM dataset by processing the original dataset at different levels of scope. The components involved are presented schematically below. In the following Sections we present how these components interact, their responsibilities and goals. Below is shown the structure of all the Scala classes involved in the transformation.

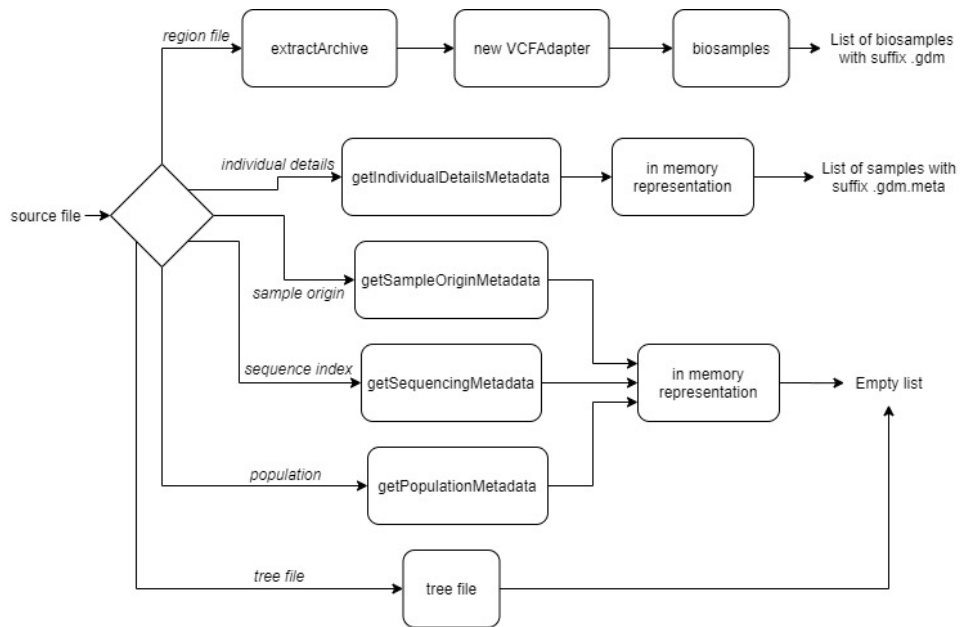
```
/src/main/scala/it/polimi/genomics/metadata/
├── downloader_transformer/one_k_genomes/
│   ├── KGMutation.scala
│   ├── KGTransformer.scala
│   ├── KGParallelTransformer.scala
│   ├── SchemaAdapter.scala ... (includes also MutationPrinterTrait)
│   └── VCFAdapter.scala
├── util/
│   ├── AsyncFilesWriter.scala
│   ├── FileUtil.scala
│   ├── ManyToFewMap.scala
│   ├── PatternMatch.scala
│   ├── XMLHelper.scala
│   └── vcf/
│       ├── HeaderMetaInformation.scala
│       ├── VCFMutationTrait.scala
│       ├── VCFMutation.scala
│       ├── VCFMultiAllelicSplitMutation.scala
│       ├── VCFFormatKeys.scala
│       └── VCFInfoKeys.scala
```

KGTransformer

KGTransformer inherits from the Scala trait **Transformer** two abstract methods, called by the framework and separating the transformation process into two steps, the identification of the output samples first, and the transformation of the source files into a GDM dataset then. The former is done by the method **getCandidateNames** which is called once for every file composing the source dataset. Metadata are expressed through multiple non-standard - but still structured - formats, thus requiring a different method for each class which is chosen accordingly to the name of metadata received as input in **getCandidateNames** (here we reuse the URLs of the metadata files included in the argument XML file). After the class of metadata is

recognised, `getCandidateNames` calls one of `getSequencingMetadata`, `getPopulationMetadata`, `getIndividualDetailsMetadata` or `getSampleOriginMetadata` to immediately parse the file. The parsing is required to read the sample names and during the process a map of the interesting attributes is built and saved in memory as described in Sections 6.2.1 to 6.2.3. Since the metadata files are quite small, caching the relevant information contained inside them has a minimal impact on the memory footprint⁵, but it allows us to avoid a huge number of I/O operations which would cause a performance hit during the transformation of the metadata.

Figure 7.3: Overview of candidate names generation



During the invocation of the above methods, a list of sample names terminated by ".gdm.meta" is returned only if the input file is of class *individual details*, otherwise an empty list is returned. This is an implementation choice which has three important benefits:

1. It avoids an important performance loss, because every time Metadata-Manager receives the candidates, it saves the list on a database and

⁵ The amount of memory used is largely reduced thanks to the usage of a special data structure (**ManyToFewMap** in package `util/`) which externally behaves like a normal map, but internally replaces non-unique values with pointers. This data structure is suited for storing our metadata since many attributes have only few acceptable values (think of the ANALYSIS GROUP in *sequence index* meta for example), so instead of wasting memory for storing the same values multiple times, we use pointers for them using less memory.

7.2. Data transformation

performs a series of inspections to make sure that, after all invocations of **getCandidateNames**, a couple of candidates with extension `.gdm` and `.gdm.meta` have been declared for every sample.

2. It gives the opportunity to cross-reference and transform all the input metadata of each sample in a single instance during the second step of the process. For example, this allows to change the value of a metadata attribute according to the value of an attribute contained into another file.
3. It solves the problem of extracting the sample names from the metadata file `20131219.populations.tsv`, possible only by cross-referencing with other metadata, which however may be still unavailable in this step, because the order in which the files are called in **getCandidateNames** is controlled entirely by the framework.

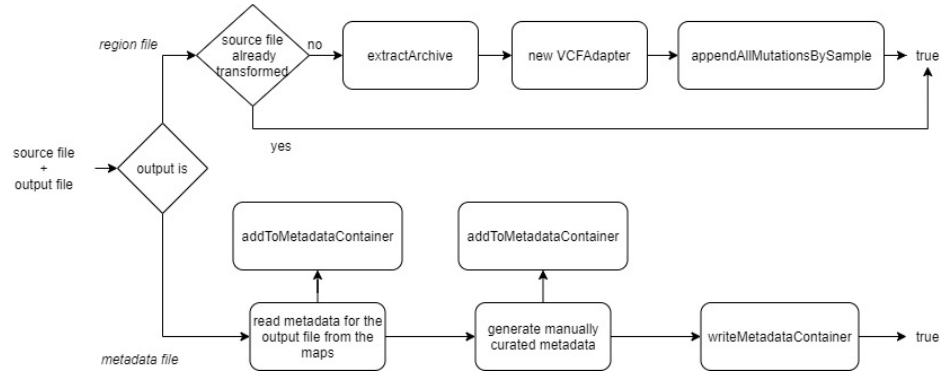
If the source dataset has been downloaded with **KGDownloaderA**, also the *tree file* is part of the source metadata and so it is issued as input for **getCandidateNames** by Metadata-Manager. Of course it contains no samples so an empty lists is returned also in this case.

As for the source regions files, **getCandidateNames** extracts the GZ-compressed files and uses a dictionary to remember the path of the uncompressed files before their processing. The structure of a VCF files is not trivial, so every operation, including obtaining the sample names, is delegated to the Scala class **VCFAdapter**, which can handle properly the format and provides the sample names in the field **biosamples** after the creation of an instance of the class (one per VCF). The list of samples is then transformed inside **getCandidateNames** by adding the extension `".gdm"` to every name in order to transform it into a list of GDM region filenames, as the framework requires.

When the next step of the transformation begins, Metadata-Manager calls the method **transform** of trait **Transformer** for every couple of (source file, candidate name) previously defined, i.e. for every pair (`<compressed VCF file>`, `<sample_name>.gdm`) and (*sequence index*, `<sample_name>.gdm.meta`). Depending on the input, **transformRegion** or **transformMetadata** is executed in turn.

When **transformRegion** is called, the dictionary of the extracted VCF files let it quickly obtain the path of the uncompressed source file corresponding to current input, without having to perform again the extraction. The path is then used to instantiate a **VCFAdapter** object. Since the

Figure 7.4: Overview of the transformation



lookup of the mutations related to a specific sample implies to scan the whole file, instead of transforming separately each pair (<compressed VCF file>, <sample_name>.gdm), **KGMutation** executes the early transformation of all the mutations contained in the origin for all the samples it contains. All the output GDM region files are initiated at the first execution of **transformRegion**, and they are then subsequently integrated with new mutations every time a new source region file is evaluated.

As for the transformation of the metadata, **transformMetadata** searches the sample name inside the maps built on top of the metadata *sequence index*, *sample info* and *individual details*, and finds the related attributes⁶. As discussed in Section 6.2.4, the population value from *sequence index* is used as index in the map of attributes derived from *population* to complete the set of file-derived metadata attributes. All the metadata and the "manually curated" values immediately available⁷ are progressively added to a temporary container, before being written all together in alphabetical order as tab-separated key-value pairs into a file named <sample_name>.gdm.meta.

VCFAdapter

The transformation of VCF files follows the threefold subdivision of the content into the meta-information section, the header and the body section. Before transforming the mutations, **VCFAdapter** scans the meta-information section to know the cardinality of the optional attributes used to describe the mutations. This information is essential to correctly assign the values of special attributes in the output variants when multiallelic mutations are sep-

⁶ See the metadata-specific sections in Chapter 6.

⁷ A part of the metadata attributes can be computed only after all the region files have been transformed. This is detailed in Section 6.2.5

7.2. Data transformation

arated into individual variants⁸. The header is a sequence of tab-separated strings, with the first 8 columns describing the mandatory columns in every VCF file, and the following ones describing the sample names. The whole line is read and the sample names are saved inside an indexed sequence of strings named **biosamples**.

When **appendAllMutationsBySample** or **appendAllMutationsBySampleRunnable** are called, the actual transformation of the region data begins. Considering a single row of the body, a number of operations occur before the corresponding mutation can be written in a GDM region file. First of all, multiallelic mutation are split by calling **splitOnMultipleAlternativeMutations** of class **VCFMutation** and one or more **VCFMutationTrait** objects are returned as a result. These objects are made GDM-compliant by the class **KGMutation**. The list of the target candidate files is derived by the genotype string contained in the columns 9th and onwards, and looking only at the samples affected by the particular **KGMutation** considered. The transformation of the row terminates with each **KGMutation** object being formatted via the class **SchemaAdapter** in a representational string, which is finally GDM-compliant and ready to be appended to the target candidate files. The above procedure repeats for every row of the body until the VCF as been completely scanned. Of course, the writes are buffered in order to minimize the I/O overhead, so **VCFAdapter** maintains a Map of **BufferedWriter** for all its target samples whose streams get closed only when the transformation of the whole VCF is completed.

With respect to its sibling, **appendAllMutationsBySampleRunnable** performs exactly the same transformations but it can reduce the overall time required by the transform stage when multiple instances of **VCFAdapter** are created. This aspect is discussed in detail in Section 7.5.2.

KGMutation

This class takes as constructor parameter a **VCFMutationTrait** object, which is an in-memory representation of a VCF mutation, and converts it into a GDM region. The transformations suitable for the purpose are applied at the moment of creation of a **KGMutation** and they are thoroughly explained in Section 6.1. Basically, it computes the start and stop coordinates in 0-based notation, it adds the length and the strand information, the at-

⁸ VCF allows to define custom attributes with a varying number of values for the ALT, INFO or FORMAT parts. In those cases, the cardinality can be a specific number, undefined, or governed by the cardinality of other attributes too. See Section 6.1.2 for more details.

tribute TYPE is normalized removing synonymous forms and eventually the reference and alternative allele strings are condensed removing the unnecessary nucleotides⁹ and adjusting the length and the coordinates consequently.

It is important to notice that this class implements the trait **VCFMutationTrait** exactly as **VCFMutation**, but they are not interchangeable, indeed they are not even in the same package (the former is from package **one_k_genomes** while the latter from **util.vcf**). The reason for it, is that **KGMutation** provides the same features of the **VCFMutationTrait** originating it in a backward compatible fashion, but it also brings some more attributes whose availability strictly depends on the source of the VCF. For example, the TYPE normalization is possible because 1KG express such information in the attributes VT and SVTYPE of its VCF files, but they're custom-defined attributes. So, another source can publish a VCF where the same information is given through a different set of attributes or it may not be given at all. So **KGMutation** is to be considered an implementation of **VCFMutationTrai** specific to 1000 Genomes.

7.3 Modifications to the framework

The 1000 Genomes data repository has a quite peculiar data structure, also it is the first of its kind to be imported into a GDM repository. A comparison of the GDM model with that used by the source highlights at least two important differences:

1. In the GDM model any region datum is a property of a sample. Instead, VCF files represent samples as attributes of a variant.
2. Considering a GDM dataset, samples and variants are in a one-to-many relation (each sample has a .gdm file listing its variants) while in 1KG, one VCF describe the variants of different samples, but each sample can appear across multiple VCF files, so the relation is of type many-to-many.

Despite Metadata-Manager is a framework designed to help the integration of a multitude of sources through a very general approach, the particular characteristics of 1KG made this an unprecedented case, which demanded

⁹ Since VCF express the variants in 1-based coordinates, usually the nucleotide preceding the mutation is included in the REF and/or ALT columns to help the positioning of INDELs and structural variants. This often happens in SNPs and MNPs multiallelic mutations too. Though, in 0-based coordinates it is possible to tell the exact position of any variant without that prefix so it can be removed.

7.3. Modifications to the framework

the evaluation of new methods by which to guide the transformation stage in order to avoid inefficiencies while keeping the approach backward compatible with the modules that have been already developed for Metadata-Manager.

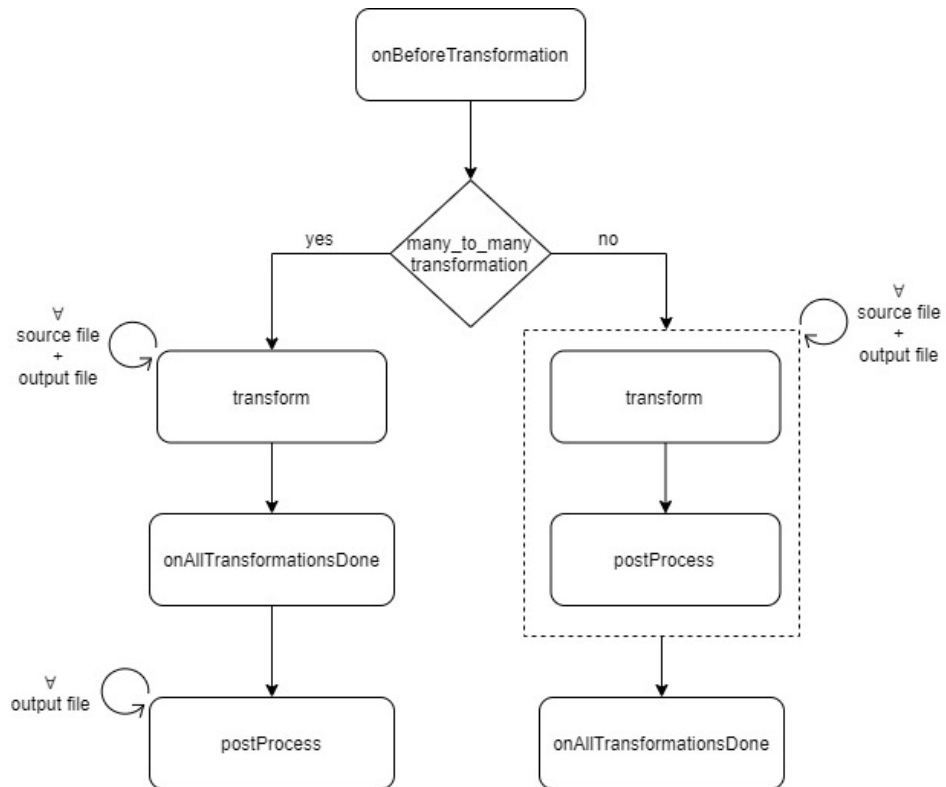
The new approach to the transformation of a source is enabled by the optional configuration parameter *one_to_many_transform* and effectively traduces into in a few changes to the components responsible for this stage, **TransformerStep** and **Transformer**:

- By default, two candidates having the same name but declared into separate runs of **getCandidateNames**, meaning they come from different origin files, are treated as different samples by the framework which appends to their names a copy number in order to distinguish them. When the XML configuration parameter *one_to_many_transform* is true, the two candidates previously described are instead treated as one, so two runs of **transform** can have different origin and same target filename, allowing the module to enhance a candidate progressively multiple times. This effect is accomplished by selectively setting to false the argument *useUrl* of method **FileDatabase.fileId**.
- All the operations involving metadata enrichment, consistency checks and sorting usually carried out for every sample right after a call to **transform** are now collected into the method **postProcess**.
- Normally the post-processing operations would be executed right after each invocation of transform. Post-processing operations include the addition of the remaining "manually curated" attributes to the metadata files, the sort of the regions and the verification of the correspondence between the region files and the schema provided. Since 1KG is a many-to-many transformation, such operations would be repeated fruitlessly many times while the samples are enriched by each source region file. So, when *one_to_many_transform* is true the post-processing operations are delayed until the end of all transformations and they are executed only once when all regions of all the samples are finally complete.
- Now every transform module of Metadata-Manager can optionally override the abstract methods **onBeforeTransformation** and **onAllTransformationsDone** from the parent trait **Transform**. The meaning of the callbacks is obvious, however there is a slight difference in the behaviour of **onAllTransformationsDone** depending on the value of *one_to_many_transform*. If true, the callback is received after all

the source region files have been transformed and before the the post-processing operations applied by **TransformerStep**, otherwise it is received after both all the transformations and the post-processing operations have been completed. This behaviour is necessary to preserve backward compatibility with previous sources, where the transform and the post-processing operations are executed atomically, i.e. as a single instruction for each sample.

Of course this new approach helps the transformation of 1KG source, but most importantly, it gives an important contribution supporting the transformation of all the sources characterized by a many-to-many relation between samples and variants.

Figure 7.5: Callgraph of the transformation and post-processing operations in the modified framework.



7.4 Final integration steps

Although with the transformation described so far we have obtained a representation of the source files in GDM format, these files are not yet ready

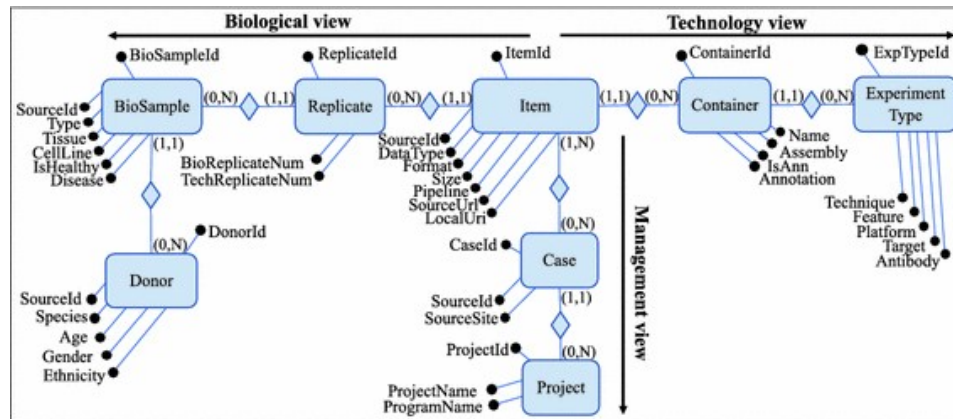
7.4. Final integration steps

for integration with other sources. In fact, the process of data integration through Metadata-Manager also requires the completion of the Cleaning, Mapping, Normalization / Enrichment, Flattening and finally Loading phases. These phases are performed through modules already integrated in Metadata-Manager that require only an adequate configuration to adapt their behaviour to the source used. In this section, we concisely describe the purpose of the remaining phases of the integration process and the steps performed to carry them out.

The goal of the Cleaning phase is of simplifying the names of the metadata keys generated during the Transformation stage. This step is necessary, in particular, for those metadata transformations whose input is in a hierarchical format (e.g. JSON or XML), which thus produce metadata pairs with long and difficult to read keys. However, since 1000 Genomes Project's metadata is in tab-delimited format, the pairs produced generally have simple keys, mostly composed of only one or two words (See Table B.3 for a complete list of the metadata attributes produced by the transformation). For this reason, the 1000 Genomes Project Cleaner module does not affect metadata.

One of the fundamental phases carried out by Metadata-Manager is the mapping phase. In this stage, we map attributes of the source to the entities described into the Genomic Conceptual Model (discussed in Chapter 3). Before executing this process, a database is set up with the tables implementing the GCM relational model shown in Figure 7.6. The execution of this stage is configured for 1000 Genomes Project through an XML file (below we report a snippet of the input used) which, for each destination table, declares mapping rules assembling groups of metadata <key, value> pairs into relational rows. Sometimes special rules are used to concatenate the values of a multi-valued attribute. In Listing 7.1, we report an excerpt of such configuration file, which shows how some values of the PROJECTS and ITEMS tables are assigned. For example, the attribute *projectName* of table PROJECTS is assigned with the value of the metadata *study_name*. Instead the attribute *sourceId* of table ITEMS, is obtained as the result of two consecutive rules. The first rule copies the value of *manually_curated__file_name* and removes the extension ".gdm" from it, then the previously computed string is concatenated with the value of the *manually_curated__assembly*. A list of the rules used to map 1000 Genomes Project attributes to the entities modeled by GCM is illustrated in Table C.1

Figure 7.6: Entities and relations described by the Genomic Conceptual Model. Source of image: [1]



Normalizer/Enricher and Constraint Checker stages are source independent, and the framework Metadata-Manager executes them autonomously. In this stage, the framework automatically find and possibly assign missing values from the previous stage, i.e. columns of the GCM tables that still do not have any valid value. The Flattener module materializes the mapped/enriched metadata into an equivalent file-based representation by transposing the tables into GDM metadata files and pairing them to the corresponding GDM regions generated from the transformation. Finally, the "flattened" dataset is then copied by the Loader module into the integrated GDM repository maintained by the Genomic Computing group of Politecnico di Milano.

7.5 Performance optimizations

Performance is a widely used term in the field of computer science, as it may refer to a broad category of concepts: reliability, execution time, reactivity and start-up time, just to name a few. In this sub-chapter we are concerned only with the aspects involved in the reduction of the overall execution time of the transform stage. The transform module of 1000 Genomes makes use of some expedients in order to gain performance, including the use of *lazy val* to cache long computations, but there are especially two features giving the most important contribution in this sense.

7.5. Performance optimizations

Listing 7.1: Excerpt from the XML configuration file used in to map metadata pairs to GCM relational rows.

```
<table name="PROJECTS">
  <mapping>
    <source_key>study_name</source_key>
    <global_key>projectName</global_key>
  </mapping>
  <mapping>
    <source_key>manually_curated__project_source</
      ↪ source_key>
    <global_key>programName</global_key>
  </mapping>
</table>
<table name="ITEMS">
  <mapping method="REMOVE" rem_character=".gdm">
    <source_key>manually_curated__file_name</source_key>
    <global_key>sourceId</global_key>
  </mapping>
  <mapping method="CONCAT" concat_character="_">
    <source_key>manually_curated__assembly</source_key>
    <global_key>sourceId</global_key>
  </mapping>
  <mapping>
    <source_key>manually_curated__local_md5</source_key>
    <global_key>checksum</global_key>
  </mapping>
</table>
```

7.5.1 Dynamic source code generation and compilation

The region files produced by the transformation of 1KG comply with the schema *one_k_genomes_schema* which declares the quantity, type and order of the region attributes through a sequence of elements `<field type=...>....</field>`¹⁰. The schema file can be changed anytime in the XML configuration file by specifying a new path for the element `<schema_url>`, so for the transform module it is important the capacity to adapt the output to the schema file currently in use. Inside our transform module, this requirement was initially met by getting the list of fields and then running a pattern matching algorithm to fetch the corresponding attribute values for an **KGMutation**. Also, since every mutation is elaborated individually, the list of fields must be evaluated as many times as the number of mutations,

¹⁰ The schema is reported in Listing 6.12 in Section 6.1.8.

i.e. more than 88 million times¹¹.

With **SchemaAdapter** we achieve the same result without pattern-matching nor **if-else** statements.

When the method **fromSchema** of the object **SchemaAdapter** is called, it transforms the sequence of attributes found inside the schema into a sequence of instruction calls. Then the instructions are put inside a prepared block declaring a new instance of a Scala class and implementing the method **formatMutation** from trait **MutationPrinterTrait**. Finally, the Scala compiler package is invoked at run-time and the block of code is compiled into a concrete Scala class named **SchemaAdapter** (in Scala language it is possible to define a class and a companion object with the same name). The benefit of this strategy is a method, **formatMutation**, which takes as argument an **KGMutation** and outputs a string of tab-separated values, only the ones declared in the schema, without evaluating any conditional statement.

7.5.2 Parallel transformation

Initially the transformer module was developed to run sequentially due to its inherent simplicity. As discussed previously¹², since every output file is the result of the elaboration of many ones, in order to concurrently transform multiple VCFs, we need a way to coordinate multiple actors so that they do not interfere each other while writing on the same resources.

This issue was addressed through the development of a set of components: **KGParallelTransformer**, **AsyncFilesWriter**, and the addition of **appendAllMutationsBySampleRunnable** to **VCFAdapter**. Basically, **AsyncFilesWriter** represents a centralized access to the file system, being both a mediator and a buffer for all the writings. The same instance of **AsyncFilesWriter** is shared with the actors, i.e. the **VCFAdapter** objects which in **appendAllMutationsBySampleRunnable** can call the method **write** by passing as argument the target file and the content to write. **AsyncFilesWriter** does not immediately write, instead it appends any write request to a queue of messages to be elaborated later. The queue is consumed every 500 ms by an independent thread running in background which takes each request and perform the writings but it also can block an

¹¹From a pure performance standpoint, the **if...else** construct can give better results, however a **match-case** control statement was used because the code responsible for reading the schema must discriminate across 12 branches, so the reasons of readability, maintainability and compactness become prominent in such a context.

¹²See sub-chapter 7.3.

7.5. Performance optimizations

actor if the queue maximum capacity is reached¹³.

Moreover, a pool of objects is maintained during the life time of this class to recycle the request objects and avoid memory inefficiencies that would trigger the java garbage collector too frequently causing possible latencies.

Before writing to a target file, an actor must declare the path and an alias for the target¹⁴ through the method **addTargetFile** or **addTargetFiles**. This allows to refer to files with a name much shorter than the real path. For example, **VCFAdapter** can write a mutation to the sample HG00096 without the need to build a full path like `/home/gmql_importer_mt/1kGenomes/hg19/Transformations/HG00096.gdm`.

Optionally, it's possible to register the users of **AsyncFilesWriter** with the methods **addJob** et simila and let **AsyncFilesWriter** to automatically stop itself when all the users have terminated the writing operations and called **removeJob**. **AsyncFilesWriter** accepts a configurable maximum number of jobs concurrently running¹⁵, while further attempts to add a job before another one is removed, will temporary block the requesting user's thread until another job is removed. A blocked user, is resumed automatically when the issued job can be accepted. Such mechanism automatically regulates the maximum number of transformations running at every moment without further actions needed: **KGParallelTransformer**, runs as many instances of **VCFAdapter** as permitted and it is temporary blocked by **AsyncFilesWriter** when the threshold is reached. If the remaining region source files are less than the threshold, **KGParallelTransformer** continues its execution and waits in **onAllTransformationsDone** until all jobs have terminated.

Somehow, the parallel transformation of the source files extends also to metadata because their processing is performed by **KGParallelTransformer**, on the main thread, while the last region files are being transformed by their working threads.

Beside the realization of **AsyncFilesWriter**, that of **KGParallelTrans-**

¹³The instant amount of requests in the queue can be observed by running a **QueueObserver** thread with the instance of **AsyncFilesWriter** as argument. The rate at which to observe the queue is configurable too through the XML parameter *observe_writing_queue_size_at_rate*.

¹⁴Aliases are never deleted and **AsyncFilesWriter** does not allow the reassignment of an alias to a different target. It is responsibility of the users of the class to always use new names for new files, or to use the already existing assignments.

¹⁵The maximum number of region files being transformed in parallel can be customized in the XML parameter *max_concurrent_transformations* inside the configuration file. If **KGParallelTransformer** is used without this parameter, it is assumed the default value 3.

former was quicker thanks to inheritance. Most of the implementation of **KGParallelTransformer** is inherited from **KGTransformer** from whom it takes all the logic concerning the production of the candidate names, the metadata transformation as well as auxiliary data structures and helper methods. Consequently **KGParallelTransformer** extends the trait **Transformer** too, so it is possible to enable the parallel transformation of 1KG by changing the class path in the XML source-level parameter *transformer*.

Also **VCFAdapter** required a few changes. In principle, the method **appendAllMutationsBySampleRunnable** applies the same transformations that are performed by **appendAllMutationsBySample**; the main changes regard first the use of **AsyncFilesWriter** which replaced the direct handling of the buffered readers, and then the return type. In fact when this method is called, it returns immediately a Runnable object which is later assigned to a thread by **KGParallelTransformer**, as opposed to running the transformation directly.

Chapter 8

Data-Summarization API

In this chapter we present the Application Programming Interface (API) developed as a conclusive result of this thesis. We describe the design process of our software in Section 8.1; we analyse the 1000 Genomes Project data and understand which ones can provide the most meaningful statistics for the user, according to the available attributes. Sections 8.2 and 8.3 summarise the project's requirements and the assumptions made during its development. Then, we give a high-level description of the software architecture in Section 8.4 and we describe the main components. Section 8.5 lists the endpoints made available by our API and shows the link to the related documentation. Section 8.6 shows an example of how the Data-Summarization-API can compute statistics for a variable number of sources. From Section 8.7 to Section 8.10 we comment on the solutions that have been developed to account for the aspects of usability and scalability, providing also an example of source integration (TCGA)n. Finally, aspects of performance, privacy, as well as source-specific constraints, are discussed in Sections 8.9 and 8.10.

The chapter is focused on the source 1000 Genomes Project as the API have been designed with a focus on the analysis on large datasets of whole-genome sources; example use cases include the study of human diversity and the comparison of the effectiveness of drug treatments in different individuals.

Note on terminology. In this chapter we often refer to a set of donors with the term "population". To avoid repetitions, we also use "individuals", "set of samples" *et similia* understood with the same meaning.

Also note that the terms "samples" and "donors" refer to the same entity in 1000 Genomes Project (we assume that one donor only provides one biological sample, as we do not have information to determine otherwise);

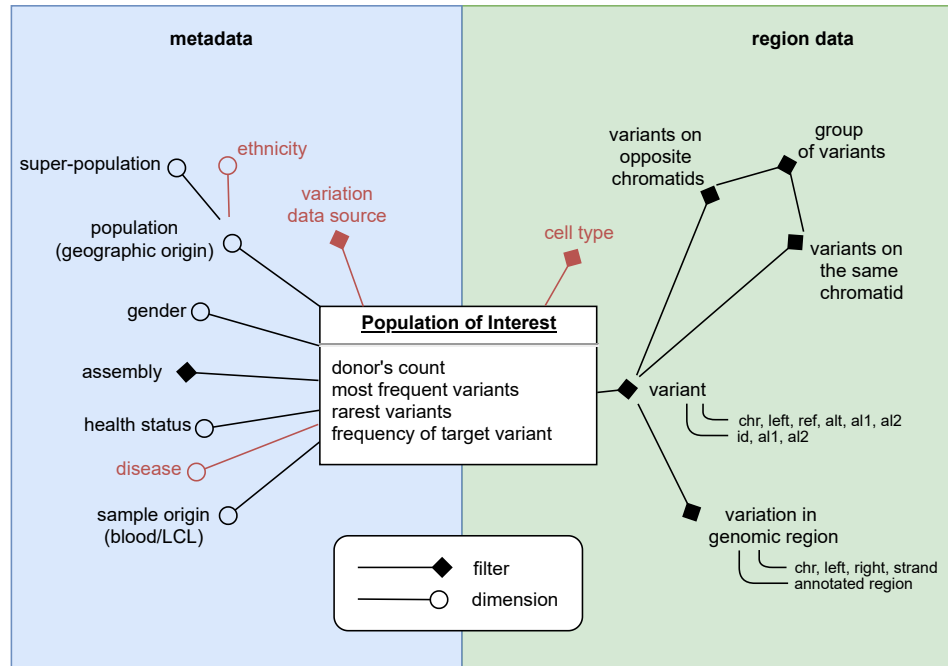
this does not hold in general for all sources.

In 1000 Genomes Project, the term "population" also refers to the specific metadata attribute used to indicate the country of origin of the donor (e.g. TSI - Tuscani in Italy); to avoid confusion, proper annotations will make clear when 'population' is used with this second meaning.

8.1 Design of supporting Data Warehouse

To support the queries of the API we built a data warehouse [2] that presents the available data in a suitable structure. The figure 8.1 is a re-elaborated version of the Fact Schema normally used in data integration which, in this situation, is better suited to the illustrate what we are going to explain in this Section. The parts in red refer to improvements applied after the initial design and discussed in Sections 8.8.

Figure 8.1: Schema of the data warehouse for Data-Summarization-API. The Figure is a reworking of the Fact Schema already known the data integration theory, indeed the central entity refers at the same time to donors and their genomes. These are separate but related concepts, the description of which through formal data integration methods would have required two interconnected Fact Schemas, making the representation less immediate. In red are shown the parts discussed in Sections 8.8.



While designing the data warehouse, we assume that the entity of primary

8.1. Design of supporting Data Warehouse

interest for the user is a set of individuals composing a population. The population has certain characteristics that can be changed directly by the user, and some descriptive properties, properly called measures.

Considering a population defined using arbitrary assigned attributes, we choose the following measures, of which the first one is relative to the population itself, while the others are obtained by considering all the variants carried by the individuals that compose it:

- count of the samples/donors
- variants with the highest frequency in the set of donors
- variants with the lowest frequency in the set of donors

The measures are specific to a single population and can be observed through various "lenses", technically called dimensions. For example, to view a population through the gender dimension means to compute the above measures for all the values of the gender attribute, leaving the other characterising parameters unchanged. At any moment, the dimensions can be either fixed or free. Free dimensions are the ones that are not constrained to assume a precise value, so each possible value of a free dimension defines a partition of the population, on which we can compute the relative measures. Fixed dimensions instead behave like filters by restricting the population to the individuals that comply with the set value. Note that, in this context, the term "dimension" indicates a slightly empowered version of a "filter". Indeed filters too can be fixed or free (or better, set and unset), but, unlike for dimensions, we cannot group the individuals of the population depending on the values that these can assume.

After a review of the metadata attributes imported from the source 1000 Genomes Project in Section 6.2.6, we can extract a subset of candidate dimensions and filters. We show this selection below.

1. gender
2. population
3. super_population
4. dna_source_from_coriell
5. assembly (only as filter)¹
6. is_healthy

On the side of region data, most of the individual attributes are of little or no use as dimensions. However, the tuple of attributes (chrom, left, ref, alt) identifies a variant. Taking into consideration a precise variant (the "target variant"), we can define the additional measure

- frequency of the target variant

Generally, we can neither consider the variants as a dimension due to the high cardinality (over 4 million for each donor on average). Still, they can be certainly employed as a filtering option, i.e., we can select a population as the one whose individuals own at least a precise set of variants into their genome. Expanding further this concept, we can define the following set of filters that apply to all the regions of the individuals composing the population:

- I. presence/absence of a precise variant
- II. presence/absence of a group of precise variants
- III. presence of some variants located into a region of interest (a gene, for example)

Also, we can take advantage of the completeness of the region attributes provided by the source 1000 Genomes Project to define additional filters. As explained in the last paragraph of Section 5.1, the two values AL1 and AL2 tell on which chromosome copy (or chromatid) the associated variant is located. Therefore we can use this information to define two groups:

- IV. individuals owning the specified variants on the same chromatid
- V. individuals owning the specified variants on opposite chromatids

Some of the dimensions and filters mentioned up to here are generalizations of others, e.g. "precise variant" and "group of precise variants". This hierarchical dependency is made explicit in Figure 8.1 with an edge connecting such dimensions/filters.

In this process, we focused on the most salient aspects represented by meta-data and region data. In the future, we may consider adding other dimensions on which to filter the dataset, such as mutation type and quality measures. As to metadata, the 1000 Genomes Project also exposes the information about family relationships among samples. In order to consider such detail, a more complex data structure should be implemented. We consider this as a further extension.

¹ There is no point comparing genomic features occurring on different assemblies, so we should use the assembly only as simple filtering option, rather than as dimension of the sample set.

8.2 Requirements

What presented in Section 8.1 and the analysis of the state-of-the-art in Chapter 3 put the basis for the definition of the following functional requirements:

- I. The system must allow the user to define a population of donors having specific metadata characteristics and optional region constraints, corresponding to the dimensions and filters here reported:
 - sample origin
 - health_status
 - assembly
 - gender
 - population (country of origin)
 - super-population
 - the occurrence of one or more precise variants in the donors' genomes
 - the occurrence on the same chromatid of two or more precise variants in the donors' genomes
 - the occurrence on opposite chromatids of two precise variants in the donors' genomes
 - the occurrence of variants in a genomic region interval
 - the absence of one or more precise variants in the donors' genomes
- II. For a population of interest, the system must be able to compute the measures:
 - count of donors
 - variants with the maximum frequency
 - variants with the minimum frequency
- III. For a population of interest and a target variant, the system must be able to compute the measure:
 - frequency of the target variant
- IV. For a population of interest, the system must return the requested measure computed for every partition generated by the combinations of the dimensions specified by the user.

V. The system should allow the use of multiple genomic variation data sources concurrently, eventually merging the information provided by each.

VI. The system should provide gene annotations when requested.

Additional non-functional requirements:

- The system should be simple to use.
- The system should allow easy integration into the existing workflow pipelines.
- The system should handle an increasing number of sources over time.

8.3 Assumptions

To simplify the development of the software we also make the following assumptions:

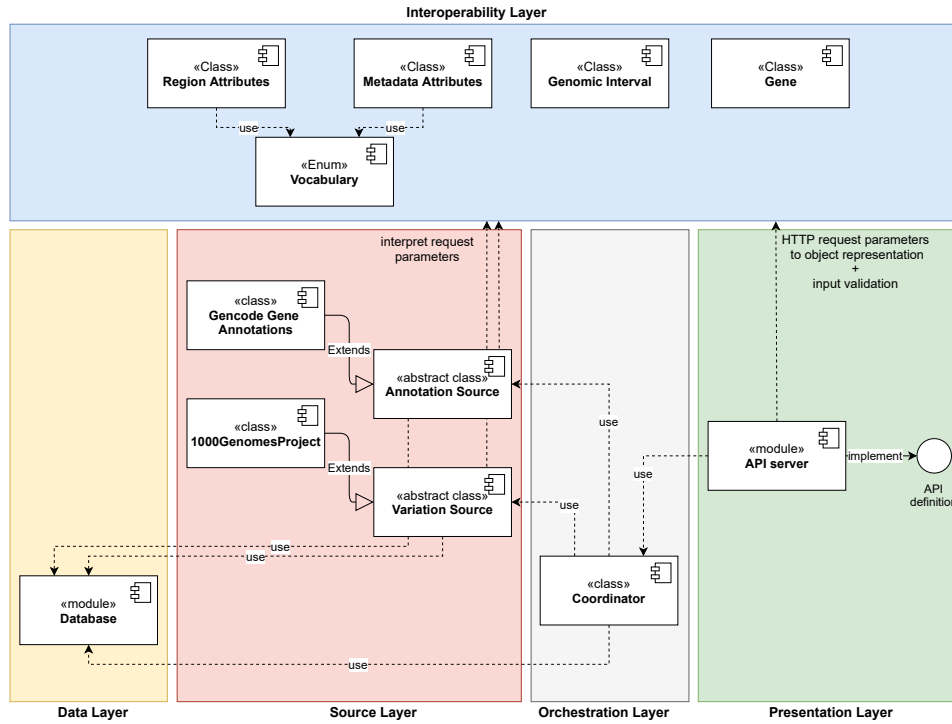
- I. Individuals in the supporting data have each a unique donor identification code.
- II. The same donor cannot be present in multiple sources.
- III. The source data are clean, i.e. no duplicated variants in the same individual, attributes are syntactically correct, etc..
- IV. The source data and metadata are available through a database connection.

8.4 Software design and architecture

The software architecture designed to meet the requirements is a client-server architecture developed with the Python programming language and available to the user as an API. Internally, the server is structured in five layers as shown in Figure 8.2. This subdivision helps in reducing the internal dependencies since each layer accomplishes a single function, and there are few points of contact between them. Here we are going to briefly introduce the layers and their functions.

8.4. Software design and architecture

Figure 8.2: Data-Summarization-API architecture overview



Representation layer

This layer is implemented by a single Python module, which implement the OpenAPI² definition through simple functions corresponding to the declared endpoints. When a request is received, before forwarding it to the Orchestration layer, it traduces the variable parameters into an equivalent object representation using the classes and definitions offered by the Interoperability layer. After the result is received, it is validated (i.e. it checks for nullity and other errors) and packed into a valid HTTP response having a JSON body³.

The characteristics of our API can be well described by the REST [7] principles. REST stands for REpresentational State Transfer and it s an architectural style for applications and protocols. We can identify five architectural constraints that a RESTful application must satisfy, although they

² OpenAPI is an industry-standard specification for building well-formed and vendor-independent API definitions. The specification is open-source, supported by the community and maintained by the OpenAPI-initiative.

³ JavaScript Object Notation (JSON) is a simple format for describing objects as strings. Due to its simplicity it is easy to read and write both for humans and machines, indeed it is usually the preferred way for APIs to exchange data over the web.

were never formally defined:

- Client-server: there is a clear separation of responsibilities between the client, which needs only to know the interface and display the result, and the server, which takes care of the application logic and the data.
- Stateless: every client's request contains all the information necessary for the server to handle it, and the server has not to remember a state. Consequently, the server can be replaced at any time without effect on the client.
- Caching: requests on objects that are read-only should be cacheable. Currently our API does not implement a caching mechanism, but it is certainly possible to implement it in a transparent way by adding a layer between Representation and Orchestration layers. A future expansion should certainly consider this as an important performance improvement.
- Layered system: the internal server layers and their role are transparent to the client.
- Uniform interface: a RESTful application should define resources on which actions can be performed in a consistent way. It is possible to use almost any endpoint of Data-Summarization-API with a simple call to the HTTP method 'POST' and a standard JSON object for the parameters.

Orchestration layer

The Orchestration layer contains the business logic of the application and computes the integration of the results obtained by the individual sources. It is implemented as a Python class, the Coordinator, which internally represents the available sources as instances of either Variation Source or Annotation Source. Of each of them, the Coordinator needs only to know the type (variation or annotation) and the available filters and dimensions.

When the Representation layer requests a measure for a genomic population, the Coordinator selects only the sources able to apply the user-defined filters and asks them to return a partial result. A partial result is the output of a function from Variation Source or Annotation Source, and it is generally either a reference to a database table, a prepared query or, for the most straightforward methods, a Python collection object. At this point, the coordinator job is to compute the requested measure and to combine the

8.4. Software design and architecture

individual results it receives, eliminating the possible duplicates. Finally, it returns to the caller a suitable Python object representing the response.

Source layer

While we can take advantage of GCM [1] for uniformly accessing meta-data, there is not an equivalent model for region data. Regions of different sources can have different data models, so it is not possible to make a generic query that applies to all of them. To account for this, every source in Data-Summarization-API is added as a self-contained package. This package can have an unlimited number of Python modules, objects and functions but it must provide a concrete implementation of either Variation Source or Annotation Source. These classes hide all the source-dependent code and allow easy integration with the Orchestration layer.

Data layer

This layer is dedicated to accessing the raw data of the sources stored on a database. The Database module administrates this shared media for all the users and takes care of the low-level details. Its role is to make sure that a connection to the database is ready whenever needed.

Interoperability layer

The Interoperability layer supports the communications between the other layers and has a crucial role in ensuring the compatibility between genomic sources. It performs three essential functions:

- To uniquely name the entities used in different sources abstracting their internal differences. It's very common indeed that the same concept maps to different names across genomic sources.
- To provide a compact and valid representation of the variable request's parameters. Indeed, Data-Summarization-API makes available many possibilities to select and refine the set of donors under examination. So a simple Python object can group many filters and, at the same time, embed validation checks. For example, it checks that a variant included in the request body contains all the necessary information to describe it (chromosome, start, reference allele, alternative allele).
- To decouple the names adopted in the API-definition from the ones used internally in the Orchestration and Source layers. So the API definition can use the synonym that improves the most the usability of

the service, without affecting the names of the corresponding entities used in the other layers.

8.5 API definition

To compute the measures discussed in Section 8.1, the API exposes four endpoints. Other three endpoints provide additional functions to support the user while using the API. The complete list is reported in Table 8.1.

Table 8.1: Endpoints available in Data-Summarization-API.

HTTP method	endpoint name
POST	http://geco.deib.polimi.it/popstudy/api/donor_distribution
POST	http://geco.deib.polimi.it/popstudy/api/variant_distribution
POST	http://geco.deib.polimi.it/popstudy/api/most_common_variants
POST	http://geco.deib.polimi.it/popstudy/api/rarest_variants
GET	http://geco.deib.polimi.it/popstudy/api/values
POST	http://geco.deib.polimi.it/popstudy/api/annotate
POST	http://geco.deib.polimi.it/popstudy/api/variants_in_region

Full documentation of the endpoints with examples and instructions is available at <http://geco.deib.polimi.it/popstudy>

8.6 Combining the results of the sources - An example

The strategy adopted to combine the results coming from different sources largely depends on the measure requested by the API endpoint. As a reference, in this Section we are going to describe the method used for the endpoint `variant_distribution`.

Upon receiving a call to the function `variant_distribution`, the Coordinator selects the sources able to filter the population as indicated by the request parameters. Let's assume that the user asked to aggregate the frequency of the target variant by gender and that the sources A and B are selected for this purpose. The Coordinator calls the method `variant_occurrence` on each of the sources, passing the details of the request as argument. As reported from the documentation (Listing 8.1), every source is responsible for returning a query or a table providing a result similar to the one shown in the example below (See Table 8.2).

Listing 8.1: Definition of the method `variant_occurrence` in class `VariationSource`.

8.6. Combining the results of the sources - An example

Table 8.2: Graphical representation of the output of function `variant_occurrence` generated from an imaginary genomic variation data source. The `OCCURRENCE` column reports the number of times the variant is present in the corresponding donor (a variant can appear two times if it is homozygous).

DONOR_ID	GENDER	OCCURRENCE
77651	male	0
61384	male	1
99715	female	1
10432	male	0
22765	female	2

```
def variant_occurrence(self ,
    connection: Connection ,
    by_attributes: List[Vocabulary] ,
    meta_attrs: MetadataAttrs ,
    region_attrs: RegionAttrs ,
    variant: Mutation) -> FromClause:
    """
    Requests a source to return the individuals
        ↪ having the characteristics in meta attrs
        ↪ and region attrs. For each of them
        ↪ specify
    - the occurrence of the target variant (as a
        ↪ number 0/1/2) in a column named as
        ↪ Vocabulary.OCCURRENCE.name.
    - the attributes given in by_attributes (which
        ↪ always includes the donor identifier).
    Order is not important. Use the nomenclature
        ↪ available in Vocabulary for column names.
    """
    raise NotImplementedError('Any subclass of
        ↪ Source must implement the abstract method
        ↪ "variant_occurrence".')
```

When all the sources have returned from the call, the Coordinator makes a union of all the rows. The result of the union represents the set of all the donors corresponding to the filters defined by the user. Before proceeding, the Coordinator computes some additional values which are needed to compute the frequency measure. Then it aggregates the donors according to the dimensions defined by the user (in the previous example, the GENDER

attribute), and for each group it runs four functions:

1. 'count': it simply counts the donors in the group.
2. 'count_positive_donors': counts the donors having a value of OCCURRENCE > 0.
3. 'count_occurrence': makes the sum of the OCCURRENCE column for all the donors.
4. 'frequency': computes the frequency of the target variant as a function of

*'count_occurrence',
 number_of_non_females,
 number_of_females,
 reference_assembly,
 chromosome_of_target_variant,
 start_coordinate_of_target_variant*

In the simplest case of a target variant located in an autosomal region, the frequency matches roughly the value of

$$\frac{'count_occurrence'}{'count' * 2}$$

however, the function takes many more input parameters to take into consideration the different number of alleles in sex chromosomes between males and females as well as the pseudo-autosomal regions⁴.

8.7 Dealing with usability of API

Usability should always be a primary concern of any software design. In this extent, we provide this software as a public API built according to the REST design principle. A de-facto standard nowadays. Since RESTful interfaces are widely used in most web services, computer scientists and bioinformaticians know how they work and how to use them in most software scripts.

⁴ Pseudo-autosomal regions are short regions located at the extremities of the chromosomes X and Y, which share the same nucleic sequence. Basically, a variant occurring in these regions can possibly be homozygous, despite it being located in two different chromosomes. Also, note that the location of the pseudo-autosomal regions differs according to the reference assembly used.

8.7. Dealing with usability of API

This makes simple the integration of Data-Summarization-API into already existing workflows where multiple tools need to cooperate. For example, to call our API from a Python script, we need just three instructions.

Listing 8.2: Snippet of code calling the endpoint `most_common_variants`.

```
import requests
population = {
    "having_meta": {
        "assembly": "hg19",
        "gender": "female",
        "health_status": "true",
        "population": [
            "BEB"
        ]
    }
}
requests.post(
    'http://geco.deib.polimi.it/popstudy/api/
    ↪ most_common_variants',
    json=population)
```

The API is made of only POST and GET endpoints with self-descriptive names and reusable body parameters. Most endpoints share the same definitions in their request body schema definition. As an example, adding the attribute `distribute_by` to the request parameters of Listing 8.2, the request body becomes valid also for the endpoint `donor_distribution`. By defining a `target_variant`, the request parameter can be used also for calling `variant_distribution`.

Except for the `values` endpoint, the others always returns a table represented as a JSON object with 'columns' and 'rows' as children objects. Thus, transforming the output into a Pandas DataFrame⁵, is as simple as show in Listing 8.3.

Listing 8.3: Transformation of the output of Data-Summarization-API into a Pandas DataFrame.

```
import pandas
response = requests.post(...)
response_body = response.json()
```

⁵ A Pandas DataFrame is a Python object commonly used in demonstrative applications. They provide handy methods to display a table and perform operations on them.

```
pandas.DataFrame.from_records(
    response_body['rows'],
    columns=response_body['columns'])
```

Such details make it possible to gain confidence with the interface of Data-Summarization-API very quickly with little effort.

The software provides a convenient and complete documentation of the endpoints online at the address <http://geco.deib.polimi.it/popstudy/>. Under each endpoint, it defines the structure of the allowed input parameters and some examples. The user can also try the API functions directly from the documentation page with ready-made examples or with custom parameters. The Figure 8.3 displays a part of the online documentation.

Figure 8.3: Online documentation of Data-Summarization-API

Data summarization 1KG 1.3.0 OAS3

</popstudy/api/openapi.json>

This API returns summary statistics on user-defined populations and their variants.

Available data include gene annotations, germline variants from the 1000 Genomes Project and somatic mutations collected by The Cancer Genomes Atlas Program.

Additional Resources

- [List of the abbreviations used in the data sources](#)
- [Python Notebooks demo](#)
- [GitHub project repository](#)

How to

- [\(video\) Exploring the documentation](#)
- [\(video\) Try the API by yourself](#)

Servers

default ▾

- POST** **/annotate** Returns the list of genes overlapping - even only partially - with a genomic region or variant.
- POST** **/donor_distribution** Returns the distribution of the individuals inside a population having the requested characteristics.
- POST** **/most_common_variants** Returns a list of the most common variants inside the selected population.
- POST** **/rarest_variants** Returns a list of the rarest variants inside the selected population.
- GET** **/values/{attribute}** Returns the set of values available for the given attribute

8.8. Measures for guaranteeing scalability

To support its integration, we also provide four instructive examples (i.e. to familiarize with parameters and requests) and two demonstrative applications in the form of IPython Notebooks⁶. They are accessible directly from the API documentation webpage and at the https://github.com/tomalf2/data_summarization_1KGP/tree/master/demo.

8.8 Measures for guaranteeing scalability

In Section 8.4 we have introduced the architectural features that make Data-Summarization-API able to integrate multiple sources. In Section 8.6 we have shown how the Orchestration layer makes use of those features to query and combine the output of different data sources. However, new data sources will always present unique peculiarities. For example, a newly integrated source may bring information that is not available in the other sources, making such information not usable in the system. Because a similar condition would put a boundary to the possibility of the system to scale in future, Data-Summarization-API integrates a mechanism to choose the sources on a per-request level automatically. This mechanism relies on the cooperation of four units: Metadata Attributes, Region Attributes, Coordinator and Variation Source/Annotation Source. The logic is described below for a genomic variation data source (but it is similar also for annotation data sources):

1. A Variation Source object must have a static (i.e. predetermined) definition of the metadata filters and region filters that it can apply to its genomic data. Available filters can only be expressed with the nomenclature available in the enum class Vocabulary, which define various filter categories like GENDER, POPULATION, etc.
2. When a request is received, the input parameters are converted into objects as Metadata Attributes and Region Attributes. When created, these two containers automatically define a list of constraints each, representing the kind of filters that a source must apply to a genomic population to satisfy the user request.
3. Before ordering a source to reply, the Coordinator asks if it can apply the filters proposed by Metadata Attributes and Region Attributes through the class-method `can_express_constraints`. The method compares the required filters with the capabilities of the source described in the static definitions, so as to determine the compatibility

⁶ One of them is explained in Section 8.11.

of the source with the request. Variation Source provides a default implementation of this method to its subclasses, although the sources can customize its behaviour for each method.

In this way, the integration of a new source can enrich the system with novel capabilities, like filters, dimensions and new measures too, but only the compatible sources will be involved, leaving the others untouched.

Filters and dimensions are treated differently. The Coordinator may ask a source to provide a measure even if the requested dimension is not available. In this case, the Coordinator groups the results obtained from all the sources and puts the value 'unknown' in the rows where the dimension is not known.

After the initial design we decided to improve further the level of generality of the dimensions and filters available by adding:

- The 'ethnicity' dimension as a generalization of the concepts of 'population' (country of origin). Because the 1000 Genomes Project does not report the ethnicity of the donors, we tried to map this concept onto the population attribute. Such mapping is described in Table 8.3.
- A filter to select only specific genomic data sources. This attribute accepts a list of names corresponding to the available data sources and forces the Coordinator to consider only those.
- A filter to select the type of cell. Genomic variants can occur in somatic tissues or germline cells. Typically, these kinds of variants are not mixed in experiments, and they are used for different purposes, so it could be helpful to discriminate between the two also in our API.

8.8.1 Integration of TCGA

The Cancer Genome Atlas Program (TCGA) is a variation data source containing somatic mutations from 6852 cancer patients with regions aligned on the assembly hg19 and 10187 with regions aligned on GRCh38.

We achieved the integration of this source in a very straightforward way. Thanks to the previously described improvements of generality, as well as to the design properties of our API, we had to apply only very few changes. First, we added a dimension to discriminate between healthy and cancer patients. Even if our API already includes 'health_status', this attribute is too generic to deal with this kind of source. So, we added also 'disease' to our API as it is more specific about the clinical condition of the patient.

8.9. Measures for improving performance

Table 8.3: Mapping of the ethnicity over population values from 1000 Genomes Project.

Ethnicity	Population Code	Population	In diaspora	Super Population
white	CEU	Utah Residents (CEPH) with Northern and Western European Ancestry	yes	EUR
	TSI	Toscani in Italia	no	
	FIN	Finnish in Finland	no	
	GBR	British in England and Scotland	no	
	IBS	Iberian Population in Spain	no	
black or african american	YRI	Yoruba in Ibadan, Nigeria	no	AFR
	LWK	Luhya in Webuye, Kenya	no	
	GWD	Gambian in Western Divisions in the Gambia	no	
	MSL	Mende in Sierra Leone	no	
	ESN	Esan in Nigeria	no	
latin american	ASW	Americans of African Ancestry in SW USA	no	AMR
	ACB	African Caribbeans in Barbados	no	
	MXL	Mexican Ancestry from Los Angeles USA	no	
	PUR	Puerto Ricans from Puerto Rico	no	
	CLM	Colombians from Medellin, Colombia	no	
asian	PEL	Peruvians from Lima, Peru	no	SAS
	GIH	Gujarati Indian from Houston, Texas	yes	
	PJL	Punjabi from Lahore, Pakistan	no	
	BEB	Bengali from Bangladesh	no	
	STU	Sri Lankan Tamil from the UK	yes	
	ITU	Indian Telugu from the UK	yes	EAS
	CHB	Han Chinese in Beijing, China	no	
	JPT	Japanese in Tokyo, Japan	no	
	CHS	Southern Han Chinese	no	
	CDX	Chinese Dai in Xishuangbanna, China	no	
	KHV	Kinh in Ho Chi Minh City, Vietnam	no	

Then, we loaded TCGA data into the database from the GDM [20] repository already used to integrate the 1000 Genomes Project data. At this point, we implemented a concrete subclass of Variation Source for TCGA and declared the available static filters on regions and meta data as reported in Table 8.4.

8.9 Measures for improving performance

Currently, the Data layer is implemented through separate tables for the metadata and the region data and they are connected by a foreign key, both for 1000 Genomes Project data and TCGA data. The metadata table is built on top of the GCM entity-relationship model as a materialized view for the donors of all the sources. Instead for regions, every source has its own region data table. This data structure is optimal to select a genomic population because the source module can take advantage of the metadata filters to reduce the sample set at the very beginning of a request. So the region filters are applied only on a reduced set of variants, i.e. only the ones of the donors having the required metadata attributes. To achieve better performance, on the Data layer, we defined three indexes targeted on the needs of the sources for the endpoints `donor_distribution`, `variant_distribution`, `most_common_variants` / `rarest_variants` and `variants_in_region`. These indexes apply to the region data tables, which

Table 8.4: Filters available for TCGA and 1000 Genomes Project in Data-Summarization-API

Available in TCGA	Filter	Available in 1000 Genomes Project
x	GENDER	x
x	ETHNICITY	x
	POPULATION:	x
	SUPER_POPULATION:	x
x	HEALTH_STATUS	x
x	DISEASE	x ¹
x	ASSEMBLY	x
	DNA_SOURCE:	x
x	WITH_VARIANT (single/multiple)	x
x	WITHOUT_VARIANT (single/multiple)	x
x	WITH_VARIANT_IN_GENOMIC_INTERVAL	x
	WITH_VARIANT_SAME_C_COPY	x
	WITH_VARIANT_DIFF_C_COPY	x
	WITH_VARIANTS_IN_GERMLINE_CELLS	x
x	WITH_VARIANTS_IN_SOMATIC_CELLS	

¹ During the integration process, we extended its support also to 1000 Genomes project, so that the user can request a measure distributed on ‘disease’ and compare its value (for example, the frequency of a target variant) between healthy donors and patients with a specific pathology.

constitute the main performance bottleneck. We defined:

- An index on the foreign key. It serves the queries that request all the variants of a sample set without applying any region filter.
- A compound index on the variant id and the foreign key (in this order). It addresses those queries requiring precise variants referenced by IDs (currently dbSNP IDs are used).
- A compound index on the variant coordinates left, chromosome and on the foreign key (in this order). This index serves the same purpose of the previous one but covers the case in which the variant is referenced by coordinates. Typically when a variant is referenced by coordinates, we use the tuple (chromosome, left, reference allele, alternative allele). However, considering a generic locus, there are no more than two variants at the same place usually, so we judged the inclusion of the reference and alternative alleles not beneficial. The column’s order in the compound index is the result of a careful evaluation, which takes into consideration the selectivity of the attributes and the compatibility with the `variants_in_region` endpoint.

Secondly, the Database singleton object maintains a pool of connections and

8.10. Discussion on privacy and dynamic incompatibility

recycles them after they are used, eliminating the overhead of creating a new connection every time.

In the Orchestration Layer, the Coordinator calls the methods of the sources in parallel using different threads, and each source disposes of a separate connection to the database to perform queries concurrently. Thanks to this, the overall performance is independent of the number of sources integrated into Data-Summarization-API, instead the maximum wait time depends only on the slowest answer.

In general, the endpoints can answer very quickly when the query involves data from TCGA and a bit slower when also involving 1000 Genomes Project data. This is understandable because the latter has 21 billion rows and almost 3400 GB of region data. That said, normally the endpoints `donor_distribution` and `variant_distribution` are quite fast also for requests using 1000 Genomes Project. The most critical operations for this source are `most_common_variants` and `rarest_variants` as the database must group and order a number of variants that grows by around 4.4 million variants for each donor included in the population. Overall, considering the volume of data processed, we are satisfied of the performance we obtained. However, to mitigate this problem, the API will warn the user when a request to `most_common_variants` or `rarest_variants` involving 1000 Genomes Project data does not include any region filter that can decrease the size of the sample set. Finally, other solutions have been considered for future improvements:

- Precompute some summary statistics offline for the requests that do not make use of region filters.
- Add a Caching layer between the Representation and Orchestration that can save results and map them to the requests. Therefore, when a request comes, if the request is in the cache, the result is already available in memory and can be returned immediately.

8.10 Discussion on privacy and dynamic incompatibility

As we have shown in Table 8.4, TCGA and 1000 Genomes Project present each a different amount of attributes that we can use. Therefore, it is not always possible to find a correspondence between the attributes available, calculate the same measures or use the same filters. Since we do not control a priori the parameters received in a request, this situation can generate an

incompatibility between a request and a data source lacking the necessary information to handle it. This type of incompatibility concerns the availability of the information present in a source and is easy to predict and manage during the execution of a request, as we have seen in Section 8.8. However, there is a category of incompatibility which, despite the adoption of the precautions mentioned above, are difficult to predict and can still occur with some combinations of parameters and specific values in the data source used. For the sake of brevity, we will call the incompatibilities of the latter kind *dynamic*, and those of the former *static* (since they do not depend on the values of the attributes). To better clarify the concept of dynamic incompatibility, we show two situations in which they occur.

A possible case happens when the API is asked to compute the frequency of a target variant in a set of individuals whose gender is unknown; indeed, the number of total alleles depends on the number of males and females in the selected population, so we are not able to compute the frequency function when all these conditions are met:

1. The filters applied by the user select a population that includes at least a donor with gender unknown.
2. The chromosome of the target variant is a sex chromosome.
3. The start coordinate of the target variant is outside the pseudo-autosomal regions.

Another example of dynamic source incompatibility is when there are constraints on the quantity and kind of information returned about a population because of privacy concerns. Nowadays, whole-genome sequencing projects impose rules to protect the identity of the project participants⁷ and to prevent the risk of discrimination. Both of these aspects are very relevant to Data-Summarization-API. Even if the currently integrated data sources contain anonymized data and are released without any restriction, we expect that in the future this API will be used with additional sources that may need to implement further checks, before or after any query, to address some privacy policies. So when a privacy restriction applies to a source, it can occur a dynamic incompatibility between the source and the request, in which case the API would not be able to answer as the user expects.

Both of these situations require the Orchestration Layer to be flexible with the source modules by allowing them to not return any result, and

⁷ A study of 2013 demonstrates the possibility to identify with high confidence the participants of a project using only public data repositories. [8]

8.10. Discussion on privacy and dynamic incompatibility

need the Presentation Layer to justify what would be an incorrect result from the point of view of the user. Therefore, we developed a message subsystem which allows a rich communication between the Coordinator, the source modules and the user. With this solution, we are able to solve the first example discussed in this Section. Indeed, the source can detect the blocking condition, remove the individuals having gender unknown, proceed to compute the result with the new sample set, and notify directly the user of the problem and the action executed to solve it. Note that applying the same correction without warning the user is not acceptable, since it would drive the user to think that the result is wrong. Indeed the endpoint `donor_distribution` would still return all the individuals regardless of the gender, generating an inconsistent value of the population size from the user's point-of-view.

The Listing 8.4 shows the effect of applying the message subsystem to this specific case.

Listing 8.4: Excerpt of response from the endpoint `rarest_variants` when asking the rarest somatic mutations aligned on hg19 from the set of patients with testicular germ cell tumors. Note that the warning message motivates the variation in the population size between first and the second mutation. In this case, there were 17 donors with gender unknown.

```
{
  "columns": [
    "CHROM",
    "START",
    "REF",
    "ALT",
    "POPULATION_SIZE",
    "POSITIVE_DONORS",
    "OCCURRENCE_OF_VARIANT",
    "FREQUENCY_OF_VARIANT"
  ],
  "notice": [
    "Note for TCGA data: Individuals with an undefined gender
    ↪ have been excluded from the population while
    ↪ calculating the frequency of variants in chromosomes
    ↪ 23 and 24"
  ],
  "rows": [
    ...,
    [
      "12",
      132547086,
      "G",
```

```
"A",
156,
11,
11,
0.035256410256410256
],
[
  "23",
  7868820,
  "A",
  "G",
  139,
  5,
  5,
  0.03597122302158273
], ...
```

The subsystem would help even in the presence of privacy constraints because the source needs to notify the user when these restrictions occur. In general, the best way to do it is by coding the data-regulation policy (which is source-specific) within the source module and verifying the compliance of a query as soon as possible. The Listing 8.5 shows an example of implementation of such constraints for the case of a query asking the variants that fall into a genomic area, from individuals with precise characteristics imposed through filters on metadata and region data. In the example, we assumed a privacy regulation, requiring that the individuals in the selected population originate from at least a minimum number of different continents. This is in order to prevent the identification of the individuals themselves and ethnic discrimination based on DNA features. The example shows the query executed to check the violation of the privacy constraint and the use of a `Notice` exception (which is part of the message subsystem) to stop the request only for the interested source and inform the user (other sources will proceed anyway in computing the request without being affected by this action).

Listing 8.5: Snippet of code checking the violation of an imaginary rule imposing a minimum quantity of diversity in the selected population to prevent ethnic discrimination and protect the identity of the participants.

```
def variants_in_region( self,
                        connection: Connection,
                        genomic_interval: GenomicInterval,
                        output_region_attrs: List[Vocabulary],
                        meta_attrs: MetadataAttrs,
                        region_attrs: Optional[RegionAttrs]
                        ) -> Tuple[Selectable, Selectable]:
    # initialize state
```

8.10. Discussion on privacy and dynamic incompatibility

```
....

# merge filters on metadata and regions and
# build the population of interest, i.e. a list of donors
donor_list = ....

# verify compliance with privacy regulation
ethnic_diversity_query = \
    select([func.count(
        metadata.c.super_population.distinct()
    )]) \
    .join(metadata, donors_list,
        metadata.c.donor_id == donors_list.c.donor_id)

ethnic_diversity = \
    try_stmt(ethnic_diversity_query, None, None).scalar()

# test
if(ethnic_diversity < ETHNIC_DIVERSITY_THRESHOLD):
    # stop and warn the user
    raise Notice(WARNING_PRIVACY_RULE_VIOLATED)
else:
    # get the variants of the individuals in the population
    ....
```

However, if the privacy policy is simple, the source modules can delegate verification operations to external components. For example, we can assume a source constraint that allows Data-Sumamrization-API to return analyses on a population only if it has a sufficiently large number of individuals. To address such kind of limitation, a source module can implement the necessary checks on its own, or, in alternative, extend `SourceBlockingSmallPopulations`⁸ like shown in Listing 8.6, which verifies the compliance of the result out-of-the-box.

Listing 8.6: Snippet of code showing how a source module can simply set a lower bound on the size of the population of interest to protect the identity of the project participants.

```
class ExampleSource(SourceBlockingSmallPopulations):

    MIN_POPULATION_SIZE_THRESHOLD = 10

    # constructor of ExampleSource
    def __init__(self,
                  logger_instance,
```

⁸ `SourceBlockingSmallPopulations` is in turn a specialization of `VariationSource`, so the source module would continue to work seamlessly with the Orchestration Layer.

```

        notify_message=do_not_notify):
# initialize SourceBlockingSmallPopulations
super().__init__(MIN_POPULATION_SIZE_THRESHOLD,
                logger_instance,
                notify_message)

# rest of constructor of ExampleSource ...

# rest of class definition ....

```

When extending `SourceBlockingSmallPopulations`, the checks are done on the partial results returned by a source to the Orchestration Layer. Considering the simplicity of the solution (just one line of code to apply the privacy policy) and the fact that the overhead is very small - almost negligible - it is a solution worth of consideration if the regulation to apply, like in this case, is not overly complicated. But in general, assessing the validity of the privacy constraints inside the source module has a significant advantage in terms of efficiency for many reasons. First, a source can readily detect the violation of the privacy rules and stop the query before calculating the requested measure, allowing the Coordinator to provide faster responses. Second, the source can leverage certain data assumptions to create efficient queries and decide the most convenient time when to verify the requirements. For example, the query that finds the most common variants in a population needs to know the gender of all the individuals before proceeding. Thus, a constraint requiring a minimum population size could be verified in a simple way without adding further overhead because the necessary information would be already available. Finally, the origins themselves define the rules applying to the usage of data, and these can vary with the release of new dataset versions. Therefore managing rules inside the source module helps the division of responsibilities between the components making the software architecture and facilitates future maintenance.

Independently from the approach used to enforce a privacy policy, when a violation occurs the message subsystem can enrich the response with details explaining the cause of the issue, producing a response as the one shown in Listing 8.7.

Listing 8.7: Example of response when a privacy policy is violated. In this case, the policy requires at least ten donors in the selected population.

```

{
  "columns": [
    "CHROM",
    "START",

```

8.10. Discussion on privacy and dynamic incompatibility

```
"REF",
"ALT",
"POPULATION_SIZE",
"POSITIVE_DONORS",
"OCCURRENCE_OF_VARIANT",
"FREQUENCY_OF_VARIANT"
],
"notice": [
  "ExampleSource: The selected query does not comply with the
    ↪ privacy constraints imposed by the source and the
    ↪ result was removed from the output. Please relax some
    ↪ filters to reach a population of at least 10
    ↪ individuals."
],
"rows": [
  .... [output generated from other sources] ...
]
}
```

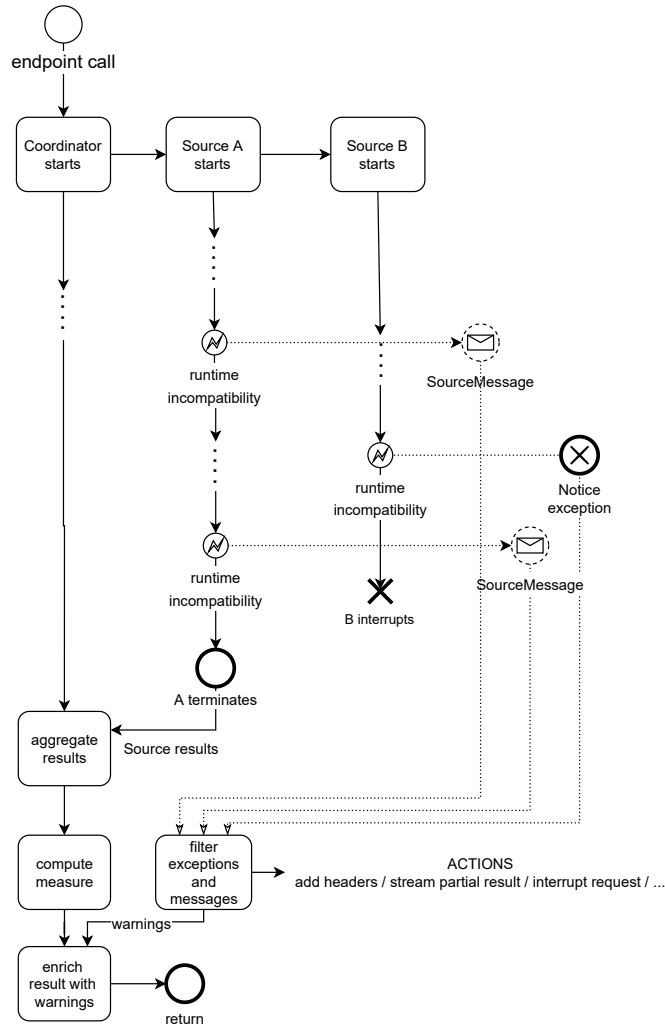
So the role of the proposed solution is triple: (i) to allow the user to receive always coherent and consistent results, (ii) to simplify the integration of the source data and (iii) to give the sources a powerful instrument to handle dynamic incompatibilities. As for the privacy and ethical issues, `SourceBlockingSmallPopulations` is an example of applied policy restriction ready to use, but our main focus has a larger scope. It aims at putting in place a general solution that will greatly ease the future integration of any source-specific policy restriction within the rest of the system.

Currently the subsystem provides two kinds of messages which can be used depending on the needs. A `Notice` can be used to stop the execution of a source module and warn of a dynamic incompatibility, like the violation of a privacy regulation. A `SourceMessage` is instead an extensible container for communicating warnings and actions without blocking the execution. A source can use it to warn the user of the action executed in order to address a non-blocking dynamic incompatibility (this is the case of the first example described in this subsection), or to ask the Coordinator to perform an action before the request terminates. For example, if the endpoint is a streaming endpoint which takes some time to complete, the sources can communicate the expected time left with a `SourceMessage` of type `ACTION`, and the Coordinator will send this information back to the Presentation Layer before the actual response is ready. This method would allow the user to receive an estimate of the time to wait before receiving the actual result.⁹ Figure 8.4 shows a high-level representation of the interactions between the sources

⁹ Unfortunately, the framework used to support the reception of calls to endpoints (Flask) does not allow the use of streaming-type responses, but the mechanisms for providing

and the Coordinator involving supplementary messages exchanged through the message subsystem.

Figure 8.4: Workflow of the message subsystem for Data-Summarization-API.



8.11 Use case - Genes functionally involved in skin melanoma by differential analysis

During this chapter, we have examined many aspects of software architecture and some considerations that justify the design choices. So in this section, we

time estimates have been set up in view of a future improvement of Data-Summarization-API.

8.11. Use case - Genes functionally involved in skin melanoma by differential analysis

want to take a more practical approach and show an example of application that uses Data-Summrization-API. Online (at the API documentation web-page <http://geco.deib.polimi.it/popstudy/>) it is possible to find and try some IPython Notebooks:

1. donor_distribution_part_1 and donor_distribution_part_2
2. variant_distribution
3. rarest_variants
4. correlation_in_genes_causing_tumor_development
5. exploring_the_link_between_stress-related_disorders_and_tumor_pathologies
6. differential_analysis_to_unveil_genes_involved_in_tumor_development

For this example, we are going to explain the main parts of the last IPython Notebook in the above list, for which we were inspired by the article "Differential analysis between somatic mutation and germline variation profiles reveals cancer-related genes." [21]. The article describes an alternative and quick technique to find genes likely involved in a specific disease based on the comparison of germline variants and somatic mutations. According to the source, this technique (DiffMut) [21] is even "more effective in discovering cancer genes than considerably more sophisticated approaches". In order to replicate this method using only aggregated data, we had to make some simplifications of the original method, however we will demonstrate that the result, at least in our case, is still valid. We focused on finding genes functionally involved in the development of cutaneous skin melanoma. The strategy adopted in our demonstration can be summarized through the following steps:

1. Select a set of candidate genes G .
2. Separately for a group of healthy donors (healthy cohort) and a group of patients affected by cutaneous skin melanoma (tumor cohort) do the following:
 - (a) Assign to each gene G_i a score S_i . Considering the set of variants V_i falling inside the region of G_i , we compute S_i as the sum of the number of donors of every variant. In mathematical form,

$$S_i = \sum_{v \in V_i} donors(v)$$

- (b) Rank normalize the gene scores by assigning the value 1 to the gene with the highest score, and then assigning to the other genes a proportionately lower value. This number represents the density of variants falling within each gene and it is a measure of how much a gene is subject to mutate in the cohort under consideration.
- 3. Compare the rank normalized scores to find the genes where the mutation and variation profiles differ the most between the tumor cohort and the healthy cohort.

In this section we will show examples of the requests used to get the data required for the experiment, without entering the details of the operations performed to compute the gene scores. However, all the operations are detailed in the IPython Notebook reachable online at <http://geco.deib.polimi.it/popstudy/>.

Find the variants falling inside a gene

The first step to compute gene scores is to get the list of variants falling inside each of them. To do so, we can query the endpoint http://geco.deib.polimi.it/popstudy/api/variants_in_region with the gene name, for each one present in the candidate list. An example of request is reported in Listing 8.8.

Listing 8.8: Example of request to know the variants falling inside the SERPINE1 gene. In this example we assume the tumor cohort to be the one formed by patients with skin melanoma and somatic mutations aligned on the grch38 assembly.

```
req_body = {
    'name': 'SERPINE1', # gene name
    'of': {
        'assembly': 'grch38',
        'disease': 'skin cutaneous melanoma',
        'having_variants': {
            'in_cell_type': ['somatic']
        }
    }
}

requests.post(
    'http://geco.deib.polimi.it/popstudy/api/
    ↪ variants_in_region',
```


8.11. Use case - Genes functionally involved in skin melanoma by differential analysis

```
json=req_body)
```

Donors of a variant

The second step to compute gene scores is to know the donors of a variant. It is possible to obtain this information by calling either the endpoint `donor_distribution` or `variant_distribution`. In the former case, we can write the variant into the population requirements `having_meta` -> `with`. In the latter, we would write the variant as "target variant". Below is reported an example of request that asks the donors of a mutation in the tumor cohort with the first method. Note that, since we are only interested in the total number of donors, it is not important the value of `distribute_by`, i.e. the user can use any arbitrary value since the response will always contain the total of donors.

Listing 8.9: Example of request to know the donors owning a variant. In this example we assume the tumor cohort to be the one formed by patients with skin melanoma and somatic mutations aligned on the grch38 assembly.

```
req_body = {
    'having_meta': {
        'assembly': 'grch38',
        'disease': 'skin cutaneous melanoma'
    },
    'having_variants': {
        'in_cell_type': ['somatic'],
        'with': [{ 'chrom': 2, 'start': 58995684, 'ref':
            ↪ 'G', 'alt': 'A' }]
    },
    'distribute_by': ['health_status']
    # value of "distribute_by" is not relevant
}
requests.post(
    'http://geco.deib.polimi.it/popstudy/api/
    ↪ donor_distribution',
    json=req_body)
```

Rank normalization and comparison

By composing the operations described above, we can compute the gene scores for a list of candidates and the two cohorts. Assuming the following

list of candidate genes, a tumor cohort including all the individuals with somatic mutations aligned on grch38, and a healthy cohort including all the individuals having germline variants aligned on grch38, through Data-Summarization-API we can compute the gene scores shown in Figure 8.5 and 8.6.

CTSZ, EFEMP2, ITGA5, KDELR2, MDK, MICALL2, MAP2K3, PLAUR, SERPINE1, SOCS3

	GENE	SCORE
0	ITGA5	45
1	MICALL2	23
2	SERPINE1	11
3	PLAUR	9
4	EFEMP2	8
5	MAP2K3	7
6	SOCS3	5
7	KDELR2	4
8	CTSZ	2
9	MDK	1

	GENE	SCORE
0	MICALL2	1530
1	KDELR2	1274
2	MAP2K3	830
3	PLAUR	740
4	ITGA5	596
5	SERPINE1	421
6	CTSZ	403
7	EFEMP2	197
8	SOCS3	83
9	MDK	61

Figure 8.5: Gene scores in tumor cohort. Figure 8.6: Gene scores in healthy cohort.

By computing the rank normalization step for the two lists of genes illustrated in Figures 8.5 and 8.6, we finally obtain the mutational profile (or density) of each gene in the healthy and tumor cohorts. At this point, the computation of the unidirectional distance of the two densities for each gene gives us a metric of evaluation to assess the functional relevance of each in the development of cutaneous skin melanoma. The density scores and their distance for the genes chosen in our example are illustrated in Figure 8.7. The blue, orange and green bars represent respectively the distance, the mutational profile in the tumor cohort and that in the healthy cohort. Following this method, we found that the gene ITGA5 is the one most likely involved in the development of cutaneous skin melanoma between the selected candidate genes, as also supported by these recent studies. [16, 10]

8.11. Use case - Genes functionally involved in skin melanoma by differential analysis

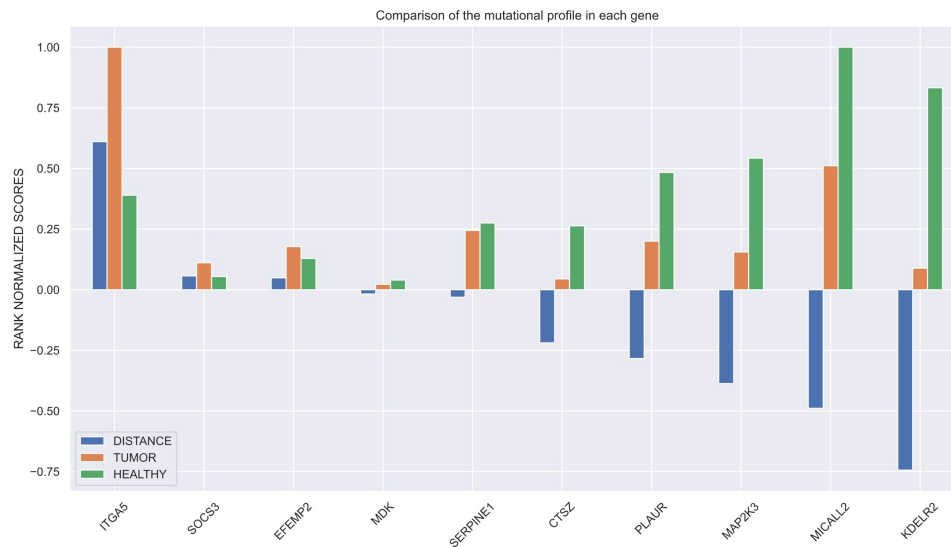


Figure 8.7: Comparison of the mutational profiles of the candidate genes for cutaneous skin melanoma.

Chapter 9

Conclusions and future prospects

In this thesis, we have documented the development of an effective tool for the analysis of genetic variations and populations. This result was possible through the study and subsequent transformation of the largest research of genetic variants of the past decade, the 1000 Genomes Project. The data thus obtained have been integrated into a Genomic Data Model repository, where they are available for tertiary data analysis, i.e. knowledge discovery, with other seven major genomic studies through GMQL and GenoSurf. The analysis of the source with solutions inspired by formal methods of data integration has allowed us to identify the most descriptive parameters of a genomic population and we have designed a software program capable of answering novel and very complex questions. During the process we took into account the aspects of usability and ease of integration, choosing to adopt an interface adhering to the industry standards and designed to simplify its use through the tools already in use by bioinformaticians and researchers. The result was the development of a public API that allows a very fine-grained selection of a population applying user-defined filters on both metadata and region data. Also, for a population of interest, the user can obtain detailed statistics about the composition of the individuals and the characteristics of their genome in aggregate form. We have designed this software to be flexible enough to accommodate similar sources that could be integrated into GDM in future, and we have demonstrated this property by making accessible TCGA's data through our API, that now can also be used to compare statistics on healthy and cancer patients with various forms of cancer. Finally, we have provided our software with suitable solutions to facilitate the integration of data sources having strict privacy regulations, thus allowing

the implementation of restrictions on the results produced according to the single data sources used and requests received from the user.

Similar software tools that we are aware of do not offer an API and do not allow to define a population of interest dynamically, nor do they allow the usage of user-defined filters on region data. Indeed they rely on pre-computed statistics only. Our solution instead answers all these questions and can do it on multiple sources, in compliance with all the privacy and ethical requirements specific to the sources.

Thanks to these results, we believe the integration of the 1000 Genomes Project into GDM and the realization of Data-Summarization API to be valuable instruments for the scientific community, with the capacity to help genomic researchers in numerous applications: from evolutionary studies to the selection of control populations in genomic experiments; from the prevention of rare disease to the identification of the driver genes for cancer.

This thesis has been fundamental to reach important achievements. However, many improvements are still possible and, as the research context continuously evolves, further expansions will be certainly necessary in order to address new expectations. Since the potential offered by Data-Summarization API, the Genomic Computing group of Politecnico di Milano has now a tool to ease genomic variant analysis and population studies while complying with the sources' restrictions on privacy. Therefore, it would be very interesting to leverage this possibility by integrating into our GDM repository a recent genomic study like the 100K Genomes Project, which imposes limitations of such kind.

As another relevant improvement, we suggest a deeper integration between GMQL and Data-Summarization API, where the latter could provide automatically - and whenever it is allowed by the genomic sources in use - the direct translation of the filters used to select a population of interest into equivalent instructions for GMQL, or a link to create them on-demand and access a corresponding dataset in GQML.

Finally, Data-Summarization API could be further improved through a more robust API framework capable of supporting streaming endpoints and request cancellation, the implementation of other endpoints, the integration of additional sources and metadata attributes and the optimization of the performance of the requests that are more computationally intensive.

Bibliography

- [1] A. Bernasconi, S. Ceri, A. Campi, and M. Masseroli. Conceptual modeling for genomics: building an integrated repository of open data. In *International Conference on Conceptual Modeling*, pages 325–339. Springer, 2017.
- [2] A. Bonifati et al. Designing data marts for data warehouses. *ACM Trans. Softw. Eng. Meth.*, 10(4):452–483, 2001.
- [3] A. Canakoglu, A. Bernasconi, J. I. Vera Pena, F. Gatti, R. Mologni, and A. Colombo. <https://github.com/deib-geco/metadata-manager>.
- [4] L. Clarke, S. Fairley, X. Zheng-Bradley, I. Streeter, E. Perry, E. Lowy, A.-M. Tassé, and P. Flicek. The international genome sample resource (igsr): A worldwide collection of genome variation incorporating the 1000 genomes project data. *Nucleic acids research*, 45(D1):D854–D859, 2017.
- [5] R. Dahm. Discovering dna: Friedrich miescher and the early years of nucleic acid research. *Human genetics*, 122(6):565–581, 2008.
- [6] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, and et al. The variant call format and vcftools, Aug 2011.
- [7] R. T. Fielding and R. N. Taylor. *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Irvine, 2000.
- [8] M. Gymrek, A. L. McGuire, D. Golan, E. Halperin, and Y. Erlich. Identifying personal genomes by surname inference. *Science*, 339(6117):321–324, 2013.
- [9] T. Hubbard, D. Barker, E. Birney, G. Cameron, Y. Chen, L. Clark, T. Cox, J. Cuff, V. Curwen, and T. Down. The ensembl genome database project. *Nucleic acids research*, 30(1):38–41, 2002.

BIBLIOGRAPHY

- [10] C. L. Kang, B. Qi, Q. Q. Cai, L. S. Fu, Y. Yang, C. Tang, P. Zhu, Q. W. Chen, J. Pan, M. H. Chen, et al. Lncrna ay promotes hepatocellular carcinoma metastasis by stimulating itgav transcription. *Theranostics*, 9(15):4421, 2019.
- [11] K. Karczewski and L. Francioli. The genome aggregation database (gnomad). *MacArthur Lab*, 2017.
- [12] W. J. Kent, C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, and D. Haussler. The human genome browser at ucsc. *Genome research*, 12(6):996–1006, 2002.
- [13] M. J. Landrum and B. L. Kattman. Clinvar at five years: delivering on the promise. *Human mutation*, 39(11):1623–1630, 2018.
- [14] I. Lappalainen, J. Almeida-King, V. Kumanduri, A. Senf, J. D. Spalding, G. Saunders, J. Kandasamy, M. Caccamo, R. Leinonen, B. Vaughan, et al. The european genome-phenome archive of human data consented for biomedical research. *Nature genetics*, 47(7):692–695, 2015.
- [15] I. Lappalainen, J. Lopez, L. Skipper, T. Hefferon, J. D. Spalding, J. Garner, C. Chen, M. Maguire, M. Corbett, G. Zhou, et al. Dbvar and dgva: public archives for genomic structural variation. *Nucleic acids research*, 41(D1):D936–D941, 2012.
- [16] Y. S. Lee, C. H. Lee, J. T. Bae, K. T. Nam, D. B. Moon, O. K. Hwang, J. S. Choi, T. H. Kim, H. O. Jun, Y. S. Jung, et al. Inhibition of skin carcinogenesis by suppression of $\text{nf-}\kappa\text{b}$ dependent itgav and timp-1 expression in il-32 γ overexpressed condition. *Journal of Experimental & Clinical Cancer Research*, 37(1):293, 2018.
- [17] H. Li. Tabix: fast retrieval of sequence features from generic tab-delimited files. *Bioinformatics*, 27(5):718–719, 2011.
- [18] E. Lowy-Gallego, S. Fairley, X. Zheng-Bradley, M. Ruffier, L. Clarke, P. Flicek, and 1000 Genomes Project Consortium. Variant calling on the grch38 assembly with the data from phase three of the 1000 genomes project [version 1; not peer reviewed]. *Wellcome Open Research*, 4, 2019.
- [19] M. Masseroli, A. Canakoglu, P. Pinoli, A. Kaitoua, A. Gulino, O. Horlova, L. Nanni, A. Bernasconi, S. Perna, E. Stamoulakatou, et al. Processing of big heterogeneous genomic datasets for tertiary analysis of next generation sequencing data. *Bioinformatics*, 35(5):729–736, 2019.

BIBLIOGRAPHY

- [20] M. Masseroli, A. Kaitoua, P. Pinoli, and S. Ceri. Modeling and interoperability of heterogeneous genomic big data for integrative processing and querying. *Methods*, 111:3–11, 2016.
- [21] P. F. Przytycki and M. Singh. Differential analysis between somatic mutation and germline variation profiles reveals cancer-related genes. *Genome medicine*, 9(1):79, 2017.
- [22] 1000 Genomes Project Consortium et al. A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061, 2010.
- [23] 1000 Genomes Project Consortium et al. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422):56, 2012.
- [24] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68–74, 2015.
- [25] S. C. Schuster. Next-generation sequencing transforms today’s biology. *Nature methods*, 5(1):16–18, 2008.
- [26] P. H. Sudmant, T. Rausch, E. J. Gardner, R. E. Handsaker, A. Abyzov, J. Huddleston, Y. Zhang, K. Ye, G. Jun, M. H.-Y. Fritz, et al. An integrated map of structural variation in 2,504 human genomes. *Nature*, 526(7571):75–81, 2015.
- [27] A. Tan, G. R. Abecasis, and H. M. Kang. Unified representation of genetic variants. *Bioinformatics*, 31(13):2202–2204, 2015.
- [28] J. G. Tate, S. Bamford, H. C. Jubb, Z. Sondka, D. M. Beare, N. Bindal, H. Boutselakis, C. G. Cole, C. Creatore, E. Dawson, et al. COSMIC: the catalogue of somatic mutations in cancer. *Nucleic acids research*, 47(D1):D941–D947, 2018.
- [29] K. A. Tryka, L. Hao, A. Sturcke, Y. Jin, Z. Y. Wang, L. Ziyabari, M. Lee, N. Popova, N. Sharopova, M. Kimura, et al. Ncbi’s database of genotypes and phenotypes: dbgap. *Nucleic acids research*, 42(D1):D975–D979, 2014.
- [30] J. Weissenbach. The rise of genomics. *Comptes rendus biologiques*, 339(7-8):231–239, 2016.
- [31] D. R. Zerbino, P. Achuthan, W. Akanni, M. R. Amode, D. Barrell, J. Bhai, K. Billis, C. Cummins, A. Gall, and C. G. Girón. Ensembl 2018. *Nucleic acids research*, 46(D1):D754–D761, 2017.

- [32] D. R. Zerbino, S. P. Wilder, N. Johnson, T. Juettemann, and P. R. Flicek. The ensembl regulatory build. *Genome biology*, 16(1):56, 2015.

Appendix A

Assessment of 1000 Genomes Project source access methods

A.1 Access methods

IGSR exposes a variety of ways to access, download and explore the data of 1000 Genomes Project. Here below some of these methods are presented.

A.1.1 Data browsers

During its development, 1000 Genomes Project Consortium used to build a genome browser at the end of each phase in order to make the data available as soon as they were discovered. These browsers were developed on custom versions of the Ensembl¹ browser [31], containing not just 1000 Genomes Project data, but also regulatory elements², genes, transcripts and the variants from the Ensembl Variation Database containing data from dbSNP, ClinVar [13], COSMIC [28], dbGaP [29], dbVar [15], EGA [14] and many other sources. A list of the versions built at each step follows in Table A.1.

In the Ensembl browser application, it is possible to browse 1000 Genomes Project data by selecting the filter “1000Genomes” inside the variant table for human genome. This table features all the variants identified during the project development, in addition to their subsequent realignments with respect to the assembly GRCh38 from IGSR.

A different kind of data browser, is the one built by IGSR which contains information from the 1000 Genomes Project and derived. It allows to browse

¹ Ensembl is a project based at EMBL-EBI started in 1999, some years before the completion of the Human Genome Project, in order to integrate and distribute reference datasets and analysis tools for researchers on genomics.

² The Ensembl Regulatory Build [32]

Table A.1: Genome browsers built during 1000 Genomes Project. URL of the browser is in the notes below the table.

Dataset contained	Ensembl genome browser version	Aligned reference genome
Pilot phase ^a	60	NCBI36 ^d
Phase 1 ^b	73	GRCh37
Phase 3 ^c	80	GRCh37

^a <http://pilotbrowser.1000genomes.org/index.html>.

Accessed 14 May 2020.

^b <http://phase1browser.1000genomes.org/index.html>.

Accessed 14 May 2020.

^c <http://phase3browser.1000genomes.org/index.html>.

Accessed 14 May 2020.

^d NCBI36, also known as hg18, is a reference assembly created in 2006 by the International Human Genome Sequencing Consortium.

a rich collection of files organized by sample, population, project (referred as to "data collection") and analysis group (e.g. exon and low-coverage genome wide sequencing)³. The files include sequences, alignment and variant files part of the pilot study, phase 1 and phase 3 of the main project, as well as the alignments on the assembly GRCh38 produced ex post from the IGSR team, and data originated from other projects which developed on top of 1000 Genomes Project analyses. The Data Portal is available at <https://www.internationalgenome.org/data-portal/sample>.

A.1.2 Application Programming Interfaces

The information displayed in Ensembl genome browser are provided by a database server running under the hood and queried by the genome browser web application, every time the user explores the sequences contained within. Ensembl relies on multiple database platforms, such as MySQL and MariaDB, and offers publicly accessible APIs to them.

In Table A.2, we show a summary of the available databases and some useful notes in order to create a connection to the server. In order to fetch large dataset and/or make complex analyses, Ensembl recommends using one of servers named `ensembl`, `useastdb` or `asiadb`, using its Perl

³ At the time of writing, Data Portal is released only in beta version, meaning it is still in development and possibly, it may contain errors or missing data.

Figure A.1: Excerpt of the 1000 Genomes Project phase 3 genome browser displaying some variants aligned on the chromosome 6 of the assembly GRCh37



API or the REST API. Differently, martdb is best suited to compose complex cross-database queries through the data-mining tool BioMart. Since the original 1000 Genomes browsers run on customized versions of Ensembl genome browser, they are supported by a MySQL server too. In this case, the database is shared among the resources and it is accessible through the same Perl API of Ensembl own browser, with the credentials and information summarized in Table A.3.

Figure A.2: With the IGSR Data portal it is possible to filter the samples and download all the related content: raw sequences, alignments, variants and statistics.

Samples

Filter by population ▾ Filter by analysis group ▾ Filter by data collection ▾ Toggle table view Download the list

Filters: GBR ✕ GRCh38 ✕ Low cov WGS ✕ Exome ✕

Showing 1 to 50 of 102 samples

« Previous Next »

Sample	Sex	Population	GRCh38	Phase 3	Phase 1	Platinum pedigree	Structural variation	Gambian variation	90 Han Chinese	Geuvadis
HG00099	Female	GBR	●	●	●					●
HG00102	Female	GBR	●	●	●					●
HG00107	Male	GBR	●	●	●					●
HG00234	Male	GBR	●	●	●					●
HG00239	Female	GBR	●	●	●					●
HG00114	Male	GBR	●	●	●					●

Table A.2: List of database servers at Ensembl, connection details and brief description of the available data at each endpoint.

Server	User	Password	Port(s)	Version	Notes
ensemblldb.ensembl.org	anonymous	-	3306, 5306	MySQL 5.6.33	From Ensembl 48 onwards only
useastdb.ensembl.org	anonymous	-	3306, 5306	MariaDB 10.0.30	Current and previous Ensembl version only
asiadb.ensembl.org	anonymous	-	3306, 5306	MariaDB 10.0.30	Current and previous Ensembl version only
martdb.ensembl.org	anonymous	-	5316	MariaDB 10.0.30	From Ensembl 48 onwards only
ensemblldb.ensembl.org	anonymous	-	3337	MySQL 5.6.33	Databases for archive grch37 release 79 onwards
ensemblldb.ensembl.org	anonymous	-	4306	MySQL 4.1.20	Up to Ensembl 47 only
martdb.ensembl.org	anonymous	-	3316	MySQL 4.1.20	Up to Ensembl 47 only

Table A.3: Database server and connection details for the MySQL database supporting the original genome browser built for exploring the results obtained from the pilot phase and phase 1 of the project.

Server	User	Password	Port(s)	Version	Notes
mysql-db.1000genomes.org	anonymous	-	4272	-	-

Perl API

Despite directly running SQL queries on the database server is possible, it requires the knowledge of the underlying database schemas⁴ used by Ensembl.

Instead, the Perl API is capable of providing the required data through object abstractions, which are independent of the database schemas (which may be subject to changes as new versions of Ensembl are released), thus creating a middle layer between the data source and any application using it. A detailed description of the API is available on Ensembl Documentation web page⁵. The following snippet inspired by the official documentation and shows how to connect to the database for the assembly GRCh37 and looking for a variant site by name (in this case, a reference SNP accession, assigned by the Single Nucleotide Polymorphism, or dbSNP, database).

Listing A.1: Snippet of code showing the usage of Ensembl's PERL API to fetch a variant from Ensembl variation database.

```
use Bio::EnsEMBL::Registry;
my $registry = 'Bio::EnsEMBL::Registry';

$registry->load_registry_from_db(
    -host => 'ensembl.ensembl.org',
    -user => 'anonymous',
    -port => 3337,
);
my $variation_adaptor = $registry->get_adaptor(
    'human', # species
    'variation', # database type
    'variation' # object type
);
my $variation = $variation_adaptor->
    fetch_by_name('rs1333049');
```

⁴ At the address http://apr2019.archive.ensembl.org/info/docs/api/variation/variation_schema.html it is described the schema of the Ensembl variation database version 96 (the current one, at the time of writing), to navigate across GRCh38 and GRCh37 aligned variant call sets of 1000 Genomes Project at the server "ensembl.ensembl.org".

⁵ Ensembl variations Perl API documentation: <http://apr2019.archive.ensembl.org/info/docs/api/variation/index.html>.

RESTful APIs

Ensembl offers a classical but still very powerful RESTful API . Access to this API is possible from the servers:

- <http://rest.ensembl.org/> - for the data referring to GRCh38.
- <http://grch37.rest.ensembl.org/> - for the data referring to GRCh37.

The data accessible at the second server is a snapshot from release 75 in Ensembl, i.e. it may miss some of call sets identified in the very last phase of the 1000 Genomes Project. Indeed, the 1000 Genomes genome browser for phase 3 data was released with a customized version of the Ensembl browser at version 80, as reported in the Data browsers section (A.1.1).

The specific API endpoints can be queried by sending HTTP formatted requests at URLs composed by the concatenation of the server address, the resource name and the arguments. See Table A.4 for a list of the available endpoints in the Variation API and their actions. As an example, we show in Listing A.2 a chunk of the response received from the GRCh38 server queried for the variant features and the genotypes of a variant identified in dbSNP with the accession number rs56116432. Nevertheless, the Variation API is just a component of the whole interface which comprises more than 20 endpoints, whose relation are schematized in Figures A.3 and A.4.

Table A.4: Endpoints of the Variation RESTful API from Ensembl. Mandatory parameters are the ones prefixed by ":". Possibly there may be other optional parameters specified with a "key=value" syntax located in the body of a POST request or instead listed after a "?" symbol and separated by "&" if it is a GET request. For example, `http://rest.ensembl.org/info/ping?content-type=application/json` performs a GET request to the `/info/ping` service without additional parameters.

Resource	Description
GET <code>variant_recoder/:species/:id</code>	Translate a variant identifier, HGVS notation or genomic SPDI notation to all possible variant IDs, HGVS and genomic SPDI
POST <code>variant_recoder/:species</code>	Translate a list of variant identifier, HGVS notation or genomic SPDI notation to all possible variant IDs, HGVS and genomic SPDI
GET <code>variation/:species/:id</code>	Use a variant identifier (e.g. rsID) to return the variation features including optional genotype, phenotype and population data
POST <code>variation/:species/</code>	Use a list of variant identifiers (e.g. rsID) to return the variation features including optional genotype, phenotype and population data
GET <code>variation/:species/pmcid/:pmcid</code>	Fetch variants by publication using PubMed Central reference number (PMCID)
GET <code>variation/:species/pmid/:pmid</code>	Fetch variants by publication using PubMed reference number (PMID)

Listing A.2: Chunk of response from the Ensembl RESTful API obtained by calling the endpoint `http://rest.ensembl.org/variation/human/rs56116432?content-type=application/json;genotypes=1`

Response:

```

    "ambiguity": "Y",
    "ancestral_allele": "C",
    "evidence": [
    "Frequency",
    "1000Genomes",
    "ESP",
    "ExAC",
    "TOPMed",
    "gnomAD"
    ],
    "synonyms": [
    "NP_065202.2:p.Gly230Asp",
    "NM_020469.2:c.689G>A"
    ],
    "source": "Variants (including SNPs and indels) imported from
dbSNP",
    "MAF": 0.00259585,
    "genotypes": [
    {
    "sample": "1000GENOMES:phase_3:HG00096",
    "gender": "Male",
    "genotype": "C|C"
    },
    {
    "sample": "1000GENOMES:phase_3:HG00097",
    "gender": "Female",
    "genotype": "C|C"
    },
    , ## data omitted for brevity ##
    ],
    "minor_allele": "T",
    "var_class": "SNP",
    "most_severe_consequence": "missense_variant",
    "mappings": [
    {

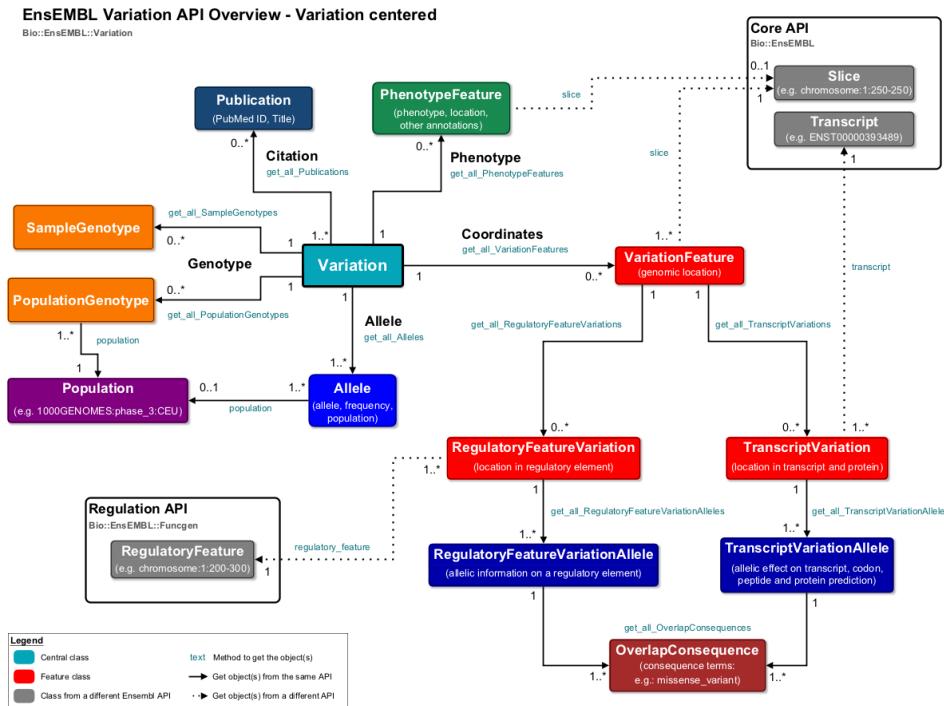
```

```

    "start": 133256042,
    "allele_string": "C/T",
    "seq_region_name": "9",
    "assembly_name": "GRCh38",
    "end": 133256042,
    "strand": 1,
    "coord_system": "chromosome",
    "location": "9:133256042-133256042"
  },
  {
    "seq_region_name": "CHR_HG2030_PATCH",
    "location": "CHR_HG2030_PATCH:133256189-
133256189",
    "coord_system": "chromosome",
    "strand": 1,
    "assembly_name": "GRCh38",
    "end": 133256189,
    "start": 133256189,
    "allele_string": "C/T"
  }
],
"name": "rs56116432"
}

```

Figure A.3: Overview of the Ensembl API



A.1.3 File Transfer Protocol

Compared to the access methods presented before, the FTP server is surely the most comprehensive resource available about 1000 Genomes Project in terms of volume, heterogeneity of data, and historical completeness. The server contains the data originated from the beginning of the project until its conclusion, the intermediate releases as well as the ones corresponding to the end of each phase⁶, and the variants realigned from IGSR onto the assembly GRCh38.

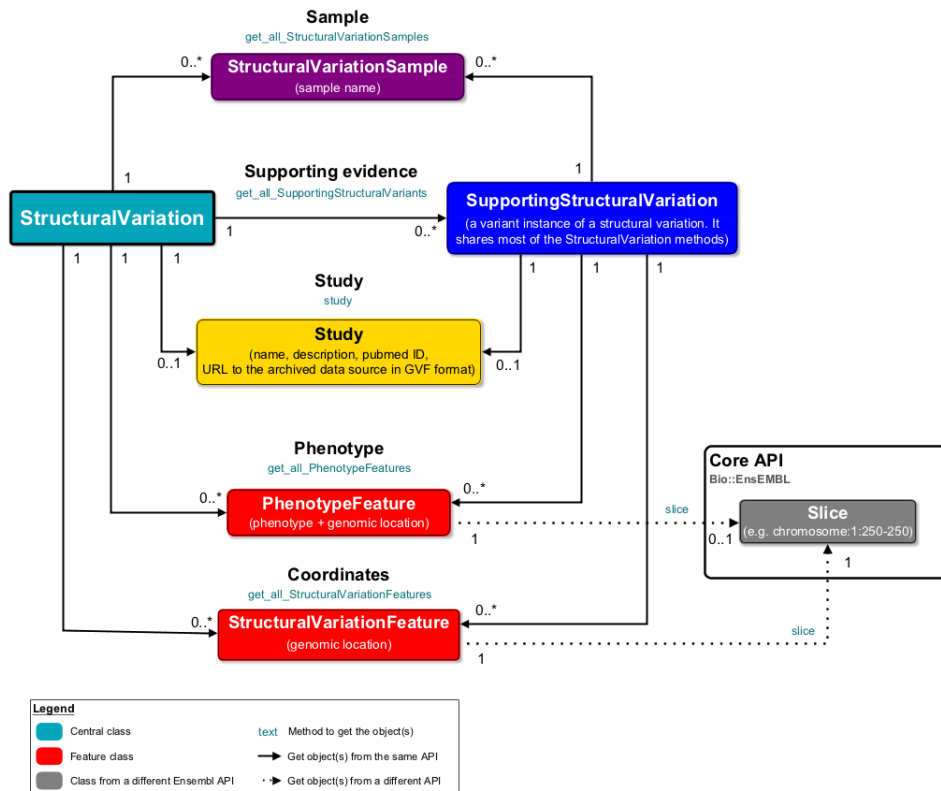
Of course, the higher quality of the results and the completeness of the data of the final phase 3 encourages its usage over the ones released while the project was still evolving, so the details related to the data structure of the intermediate releases, the first phase and the pilot phases, will be omitted from this document. The 1000 Genomes Project has two mirrored FTP sites:

- [ftp.1000genomes.ebi.ac.uk/vol1/ftp/](ftp://1000genomes.ebi.ac.uk/vol1/ftp/)
- [ftp-trace.ncbi.nih.gov/1000genomes/ftp/](ftp://trace.ncbi.nih.gov/1000genomes/ftp/)

⁶ Except for the phase 2 which was a technical phase and did not produce any variant.

Figure A.4: Overview of the Ensembl Structural Variation API

Ensembl Variation API Overview - StructuralVariation centered
Bio::Ensembl::Variation



The directory at `/phase3` holds index files for sequence reads and alignments used in the final data release. Note that the file `phase3/20130502.phase3.analysis.sequence.index` contains a subset of the sequences listed in `phase3/20130502.phase3.sequence.index`, and it corresponds to only the actual reads that were used for subsequent analyses. The most recent version of those reads and alignments are organized first by sample and then by subject (i.e. exome alignment, high coverage alignment or sequence read) inside the `/data` folder.

The analysis of the reads produced the variants discussed in final phase 3 paper of October 2015 [24]. In a second analysis [26], the Structural Variation Analysis Group of the 1000 Genomes Project, in collaboration with many different research groups, reported the identification of many more structural variants (DNA variants ≥ 50 base pairs) that were missing from the previous results. The SVs identified are available also as a standalone repository

at the path⁷ `ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/integrated_sv_map/`. Eventually, all the variant types were merged and collected in a single repository located at path `release/20130502/`, where variants have been revised (currently at version 5a) and organized as VCF compressed files, one per chromosome. For example, the file `ALL.chr1.phase3_shapeit2_mvncall_integrated_v5a.20130502.genotypes.vcf.gz` contains the variants discovered on the chromosome 1 using the samples of all populations on the last alignments, dated 2 May 2013, and revised at the version 5a. The variants located on chromosomes Y and mitochondrial (MT) derive instead from different call sets, so their files have a different signature, respectively: `ALL.chrY.phase3_integrated_v2a.20130502.genotypes.vcf.gz` and `ALL.chrMT.phase3_callmom-v0_4.20130502.genotypes.vcf.gz`.

A site file, listing variations without individual genotype data, is given for the autosomal chromosomes in the same directory, with filename `ALL.wgs.phase3_shapeit2_mvncall_integrated_v5b.20130502.sites.vcf.gz`.

The phased biallelic call set of SNPs and indels is the last dataset of variants aligned on GRCh38 available at the time of writing. The dataset is available in EMBL-EBI FTP server at the address `ftp.1000genomes.ebi.ac.uk/vol1/ftp/data_collections/1000_genomes_project/release/20190312_biallelic_SNV_and_INDEL/` and it is organized in 23 VCF compressed files, one for each chromosome, plus an additional file listing only the sites without genotype information.

A.1.4 Amazon S3 bucket

1000 Genomes Project data are also available through the cloud storage service Amazon S3 (or Amazon Simple Storage Service) at the address `http://s3.amazonaws.com/1000genomes`. The data available here reflects the state of the project until its final phase without the realignment on GRCh38. This "bucket" represents not just an alternative or backup resource, but provides also technologically different, and up to date access methods, enabling cloud-computing workflows for researchers.

The data can be accessed in a variety of ways, using one of the programming language specific SDKs (available, between the others, for Java, C++ and Python), via command line tools (such as `s3cmd`⁸ or `aws`⁹), via

⁷ For this dataset, the two servers show a few differences. The directory hosted on EBI servers provides a more updated version of the call set with respect to the one hosted on NCBI servers.

⁸ `http://s3tools.org/s3cmd`

⁹ `http://timkay.com/aws/`

browser and through cloud-computing infrastructures and services as Cloud-BioLinux¹⁰, Amazon Elastic Compute Cloud (Amazon EC2), Hadoop, Cloud-burst¹¹ and Crossbow¹². In Listing A.3 we show a code snippet exemplifying the usage of the Amazon S3 Java SDK.

¹⁰<http://cloudbiolinux.org/>

¹¹<https://www.ncbi.nlm.nih.gov/pubmed/19357099>

¹²<http://bowtie-bio.sourceforge.net/crossbow/index.shtml>

Listing A.3: Excerpt from the Amazon Web Services Documentation showing how to query an Amazon S3 bucket with the Java SDK.

```

private static final String BUCKET_NAME = "${my-s3-bucket}";
private static final String CSV_OBJECT_KEY = "${my-csv-object-key}";
private static final String S3_SELECT_RESULTS_PATH = "${my-s3-select-resultspath}";
private static final String QUERY = "select_s._1_from_S3Object_s";

final AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient();

SelectObjectContentRequest request = generateBaseCSVRequest(BUCKET_NAME, CSV_OBJECT_KEY,
    ↪ QUERY);
final AtomicBoolean isResultComplete = new AtomicBoolean(false);

try (
    OutputStream fileOutputStream = new FileOutputStream(new File(S3_SELECT_RESULTS_PATH));
    SelectObjectContentResult result = s3Client.selectObjectContent(request)){
    InputStream resultInputStream = result.getPayload().getRecordsInputStream(new
    ↪ SelectObjectContentEventVisitor() {
        @Override
        public void visit(SelectObjectContentEvent.StatsEvent event) {
            System.out.println(
                "Received_Stats,_Bytes_Scanned:_"
                + event.getDetails().getBytesScanned()

```

```

+ "_Bytes_Processed:_"
+ event.getDetails().getBytesProcessed();
}

/*
 * An End Event informs that the request has finished successfully.
 */
@Override
public void visit(SelectObjectContentEvent.EndEvent event){
    isResultComplete.set(true);
    System.out.println("Received_End_Event._Result_is_complete.");
}

});
copy(resultInputStream, fileOutputStream);
}

/*
 * The End Event indicates all matching records have been transmitted.
 * If the End Event is not received, the results may be incomplete.
 */
if (!isResultComplete.get()) {
    throw new Exception("S3_Select_request_was_incomplete_as_End_Event_was_not_received
    ↪.");
}

```

A.1.5 Other transfer services

IGSR supports other file transfer protocols, more than plain FTP. For example, the Globus Online system can set up a fast data transfer between a destination endpoint and the the 1000 Genomes Project FTP site (known with endpoint name "ebi#public"). Similarly, Aspera is another transfer protocol, which was developed at IBM to enable fast and reliable transfer for large files. The Aspera web interface is not supported any more by IGSR, but a command line interface can be used with the Aspera secure copy (ascp) tool.

A.2 Selection of the best access method

The 1000 Genomes Project made a major contribution in the genomic research field, whose results were collected during 7 years and are now freely and publicly available with the methods shown in Sections A.1.1 to A.1.5. We are going now to examine which of them is the most suitable for the purpose of importing this valuable resource into the GMQL framework. The integration process starts with the download of the data source, but one of the key features of the framework is the capability of maintaining the repository automatically updated, integrating the changes occurring in the original source and reflecting them in the GMQL framework without further intervention and given that the data model has not changed. For this reason, we are not interested in just downloading the data, but we are seeking a way to programmatically do it, even after the integration process has completed. The Data portal at IGSR and the genome browsers described in the Section A.1.1 are handful interfaces for accessing 1000 Genomes Project, but they are better designed for human exploration than for programmatic download. Even if building an automated wrapper for importing the information from the browsers is theoretically possible, probably it is not the most convenient approach given the available alternatives, i.e. APIs and data servers as illustrated in the Sections A.1.2 to A.1.5, not to mention the inherent difficulty in developing and maintaining the wrapper compatible with the browser, which, being a web application, is prone to change in future. Furthermore, the genome browsers and the Data portal at IGSR are basically views over the underlying data sources, respectively the MySQL databases and the FTP site at EMBL-EBI so the effort of building a wrapper would not be balanced by any practical advantage.

The FTP sites at EMBL-EBI and NCBI have been the primary repositories

for the data concerning the 1000 Genomes Project during its development. Even now, it hosts the latest version of the sequences, alignments and variants for both the assemblies GRCh37 and GRCh38. This represents an optimal solution to the problem of importing 1000 Genomes Project data, but it is not the only one.

Since the same data has been imported by Ensembl, its database offer almost the same level of completeness of the FTP servers, being the only difference in the integration with other projects and data sources that may add evidence to the 1000 Genomes Project variants. Ensembl database can be accessed through one of these methods:

1. Creating a MySQL client and sending queries directly to the database server.

Since the Ensembl variation database schema alone consists of 54 tables, learning to use it properly to reach our goal, may be not a straightforward operation. Most of all, the Ensembl database was created to support the genome browser and it is not guaranteed to maintain a stable structure over the time¹³.

2. Using the Perl API.

Ensembl provides a powerful API for Perl programming language, bundled with a rich documentation. An overview of the kind of data that can be retrieved with this API is described in Figures A.3 and A.4. An API is designed to provide a representation of the data, which is independent from the underlying data structure, so this access method overcomes the limitations encountered when accessing an external database server directly. However, this access method requires using Perl as programming language, or to include a Perl script into a different language, if this is an option.

3. Using the RESTful API.

The RESTful APIs (there are two: one for data aligned on GRCh37 and one for those aligned on GRCh38) of Ensembl are programming language independent and they abstract the underlying data source,

¹³Since the release of the first Ensembl genome browser [9], in 2002, 95 updates have been published. This means that the update frequency is quite high and, even if the underlying schema may be less prone to change as frequently as the application, the requirement for a constant maintenance of a compatible MySQL client should be considered if this strategy is chosen.

but they are not as complete as the previous two alternatives. In particular, the Ensembl RESTful APIs can be queried for variants only if the variant identifier from an external database like rsIDs from dbSNP, is already known¹⁴, or by scanning¹⁵ the reference genome with predefined region intervals and then filtering the output to keep only the 1000 Genomes Project variants.

That said, accessing the data through via FTP seems the most complete and simple solution. Furthermore, the benefit discussed before about using the Ensembl database, i.e. the additional evidence brought to 1000 Genomes Project data from other projects, can be easily reproduced by simply calling the Ensembl RESTful APIs to look for the variant IDs listed in the VCF files, once they have been downloaded from the FTP site. The same considerations are summarized in Table A.5, together with the evaluation of the access methods Amazon S3, Globus and Aspera.

¹⁴The endpoint `GET variation/:species/:id` requires the specie and the variant id as mandatory parameters

¹⁵The endpoint `POST ga4gh/variants/search` returns the variants in Global Alliance for Genomics and Health (GA4GH) format for a region on a reference sequence.

Table A.5: Comparison of the access methods available to programmatically download 1000 Genomes Project data. A "x" indicates that the feature described inside the column is present.

	Region data		Metadata	Additional requirements
	aligned on GRCh37	aligned on GRCh38		
MySQL server <code>ensembl.db.ensembl.org</code> (on ports 3306 or 5306 for GRCh38 and 3337 for GRCh37 aligned data)	x	x	x	The source may be subject to changes in future, requiring additional software maintenance.
Perl API connected to <code>ensembl.db.ensembl.org</code> (at ports 3306 or 5306 for GRCh38 and 3337 for GRCh37 aligned data)	x	x	x	It requires building an interface to use code written in Perl.
RESTful API on servers <code>http://rest.ensembl.org</code> for GRCh38 and <code>http://grch37.rest.ensembl.org/</code> for GRCh37 aligned data	x	x	x	It requires fetching variants ID in external sources or scanning the entire genome, in both cases with a high cost of time and bandwidth
FTP site at EMBL-EBI or NCBI	x	x	x	It may require updating a configuration file for accommodating future changes in the data aligned on GRCh38.

	Region data		Metadata	Additional requirements
	aligned on GRCh37	aligned on GRCh38		
Globus / Aspera	x	x	x	<p>It requires using the RESTful APIs of Globus or Aspera.</p> <p>It is also subjected to the same requirements of the standard FTP client access method.</p>
Amazon S3	x		x	<p>It requires building an interface for interfacing with Amazon Web Services since no Scala SDK is available.</p>

In conclusion, the simplicity of the protocol and the completeness of the data accessible through FTP, are enough to convince us to favour it over other methods for the purpose of integrating the 1000 Genomes Project data into the GMQL framework.

Appendix B

Examples of transformed data

B.1 Region data examples

The two tables in this Appendix show some examples of conversion from the original VCF format to GDM. Of each mutation we report only the attributes that are more relevant for the conversion of coordinates and the computation of the length.

Table B.1: Examples of mutations from 1000 Genomes Project on chromosome X. For space reasons, only a part of the original data is shown.

Example	#CHROM	POS	REF	ALT	INFO	FORMAT	sample's GT
a	X	170752	T	G	VT=SNP	GT	1 1
b	X	673980	TA	T	VT=INDEL	GT	1 0
c	X	297824	T	TC	VT=INDEL	GT	0 1
d	X	288023	T	<CN0>	CIEND=0,135;CIPOS=-150,0; END=290864;SVTYPE=DEL;VT=SV	GT	0 1
e	X	187604	A	<CN0>, <CN2>	END=218561;SVTYPE=CNV;VT=SV	GT	0 2
f.1	X	68498	A	AC,C	VT=SNP,INDEL	GT	0 1
f.2	X	68498	A	AC,C	VT=SNP,INDEL	GT	0 2

Table B.2: Variants of Table B.1 converted in GDM format as described in Section 6.1.8. For space reasons, only a part of the output is shown.

Example	chr	left	right	strand	AL1	AL2	ref	alt	mut_type	length
a	chrX	170751	170752	1	1	1	T	G	SNP	1
b	chrX	673980	673981	1	1	0	A		DEL	1
c	chrX	297824	297824	1	0	1		C	INS	1
d	chrX	288023	290864	1	0	1		<CN0>	DEL	2841
e	chrX	187604	218561	1	0	1		<CN2>	CNV	30957
f.1	chrX	68498	68498	1	0	1		C	INS	1
f.2	chrX	68497	68498	1	0	1	A	C	SNP	1

B.2 Metadata examples

Table B.3: Full list of metadata attributes included in a sample and example values. The third column indicates if the metadata can appear multiple times.

Output metadata	Example value	Cardinality > 1
analysis_group	low coverage	x
center_name	BGI, ILLUMINA	
dna_from_blood	FALSE	
dna_source_from_coriell	lcl	
experiment_id	ERX224791, ERX224792, ERX232563	
family_id	HG00288	
gender	female	
insert_size	180, 185, 400	
instrument_model	Illumina HiSeq 2000	
instrument_platform	ILLUMINA	
library_layout	PAIRED	
manually_curated__assembly	GRCh38	
manually_curated__chromosome	chr8	x
manually_curated__data_type	variant calling	
manually_curated__data_url	ftp.1000genomes.ebi.ac.uk/...	x
manually_curated__download_date	2020-01-23T02:19:43.786+01:00	x
manually_curated__feature	variants	
manually_curated__file_format	bed	
manually_curated__file_name	HG00288.gdm	
manually_curated__is_healthy	TRUE	
manually_curated__local_file_size	408695129	
manually_curated__local_md5	f2f2ee146cf20d90c6859bf091c1a49c	
manually_curated__origin_file_size	1121169725	x
manually_curated__origin_last_modified_date	Tue Mar 12 00:00:00 CET 2019	
manually_curated__origin_md5	[NULL]	x
manually_curated__pipeline	BCFTools, Freebayes, GATK...	
manually_curated__project_source	1000 Genomes	
manually_curated__source_page	ftp.1000genomes.ebi.ac.uk/...	
manually_curated__species	Homo Sapiens	
population	FIN	
sample_id	SRS368498	
sample_name	HG00288	
study_id	SRP004058, SRP000808	
study_name	1000 Genomes Finnish ...	
super_population	EUR	

Appendix C

Rules of mapper module for transformed metadata of 1000 Genomes Project

Table C.1: Rules adopted for the generation of the Mapper XML configuration file of the Mapper stage of Metadata-Manager.

GCM TABLE	entity modeled	source attribute / derivation process
PROJECTS	<i>projectName</i>	<i>programName</i>
CASES	<i>programName</i>	<i>manually_curated__project_source</i>
	<i>sourceId</i>	<i>experiment_id</i>
	<i>sourceSite</i>	<i>center_name</i>
ITEMS	<i>sourceId</i>	remove ".gdm" extension from <i>manually_curated__file_name</i>
		+
	<i>size</i>	<i>manually_curated__assembly</i>
	<i>date</i>	<i>manually_curated__local_file_size</i>
	<i>checksum</i>	<i>manually_curated__origin_last_modified_date</i>
	<i>platform</i>	<i>manually_curated__local_md5</i>
	<i>pipeline</i>	<i>instrument_model</i>
		<i>manually_curated__pipeline</i>
		"thttp://www.gmdl.eu/gmdl-rest/datasets/public."
	<i>sourceUrl</i>	+
		<i>dataset_name</i>
		+
		<i>file_identifier</i>
		+
		"/region"
	<i>fileName</i>	<i>manually_curated__file_name</i>
	<i>sourcePage</i>	<i>manually_curated__source_page</i>
REPLICATES	<i>sourceId</i>	<i>sample_id</i>

GCM TABLE	entity modeled	source attribute / derivation process
BIOSAMPLES	<i>bioReplicateNum</i>	1
	<i>techReplicateNum</i>	1
	<i>sourceId</i>	<i>sample_id</i>
	<i>types</i>	"cell line" when <i>dna_source_from_corie</i> == "lcl", "tissue" when <i>dna_source_from_corie</i> == "blood"
	<i>tissue</i>	"blood" when <i>dna_source_from_corie</i> == "blood", NULL otherwise
	<i>cell</i>	"lymphoblastoid cell line" when <i>dna_source_from_corie</i> == "lcl"
	<i>isHealthy</i>	<i>manually_curated__is_healthy</i>
	<i>sourceId</i>	<i>sample_id</i>
	<i>altDonorsSourceId</i>	<i>family_id</i>
	<i>species</i>	<i>manually_curated__species</i>
DATASETS	<i>gender</i>	<i>gender</i>
	<i>ethnicity</i>	<i>population</i>
	<i>name</i>	<i>dataset_name</i>
	<i>dataType</i>	<i>manually_curated__data_type</i>
	<i>format</i>	<i>manually_curated__file_format</i>
EXPERIMENTSTYPE	<i>assembly</i>	<i>manually_curated__assembly</i>
	<i>isAnn</i>	"false"
	<i>technique</i>	<i>analysis_group</i>
	<i>feature</i>	<i>manually_curated__feature</i>