POLITECNICO DI TORINO

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

TOOLS FOR INTEGRATIVE CANCER DATA ANNOTATION: A VISUAL MINING-BASED APPROACH

Relatore
Prof. Elena M. BARALIS

Co-relatore
Dott. Alessandro FIORI

Candidata
Marilisa MONTEMURRO

Anno Accademico 2017/2018

"Basterebbe abbandonare l'idea di qualsiasi facile soluzione, ma abbandonare anche il nostro appassionato pessimismo e trovare finalmente l'audacia, di frequentare il futuro, con gioia.
Perché la spinta utopistica, non è mai accorata o piangente. La spinta utopistica non ha memoria, e non si cura di dolorose attese. La spinta utopistica è, subito. Qui e ora."

Giorgio Gaber

A mia madre e mio padre.

Abstract

In the Big Data era there is an increasing interest in integrating different software systems based on different technologies and different data models. Cancer research laboratories are one of the places where data integration has become a crucial issue, due to the intrinsic complexity of treated data.
The aim of this thesis is to present a tool capable of integrating a platform for cancer genomic data management, the LAS system, with the analytical and visualization tools of an open-source platform, called cBioPortal.
The tool is based on a browser extension and co-operates with LAS APIs. It performs cluster analysis on data starting from the cBioPortal visualization instruments and allows to store the resulting high-level labels into the LAS system, in form of new annotations on the analyzed samples.
The final goal is to provide a support instrument for LAS users to let them further investigate genomic patterns and clinical evidences occurring in patients affected by cancer, in order to possibly find new cures and therapies.

Ringraziamenti

Il primo, doveroso, ringraziamento va ai docenti Elena Baralis e Alessandro Fiori, per avermi dato l'opportunità di svolgere questo lavoro di tesi. Ringrazio, poi, Alberto Grand e Andrea Bertotti per avermi seguita, aiutata e guidata nello sviluppo del progetto.

Ringrazio Sara che, in questi lunghi e faticosi anni, mi ha supportata e sopportata con pazienza, vivendo da vicino ogni mia difficoltà e condividendo con sincera gioia ogni soddisfazione.

Ringrazio, poi, Luigia e Cristiana, le migliori amiche che io abbia mai avuto; mi hanno accompagnata durante questo percorso, donandomi forza e spensieratezza, quando ne avevo bisogno, affetto, quando mi sentivo sola, e determinazione, quando mi sentivo persa.

Ringrazio, ancora, Diana, Antonella, Elisabetta, Donatella, Laura e Federico, per esser stati la mia rete di salvataggio, ogni volta che mi sono sentita crollare.

Un ringraziamento di tutto cuore va, poi, alla mia piccola famiglia, ed in particolar modo alle mie due zie e a nonna Maria, per avermi incoraggiata, sorretta e aiutata a credere in me stessa, anche quando mi sentivo il più piccolo granello di polvere su questo pianeta.

Ringrazio Giovanni, mio fratello e complice, perché averlo, qui, fisicamente vicino a me, mi ha fatto sentire meno sola quando l'inevitabile nostalgia mi travolgeva.

Ed, infine, il ringraziamento più importante, il più sentito, il più sincero: grazie, mamma e papà, per aver creduto in me, per avermi dato questa incredibile opportunità, affrontando mille difficoltà, ma senza mai perdere la fiducia in me, anche quando non credevo in me stessa. Grazie per averci creduto, voi. Grazie per avermi resa quella che sono, una donna capace e determinata. È da voi che l'ho imparato. Siete e sarete sempre il mio modello di riferimento ed è a voi che devo questo impagabile traguardo. Se un giorno avrò realizzato i miei sogni sarà per merito vostro. Per ora, un pezzetto di sogno è già divenuto realtà, per cui, grazie, dal profondo del cuore. È dedicato a voi.

Contents

Introduction **1**

I Theoretical and Technological Background **3**

1 Third-Party Software and Data Integration **5**
1.1 Data Integration 5
1.1.1 Methodologies 5
1.1.2 Data centralization 6
1.1.3 Data warehousing 6
1.1.4 Dataset integration 8
1.1.5 Hyperlinks 8
1.1.6 Federated datasets 8
1.1.7 Linked data 9
1.1.8 Technological solution: graph databases 10
1.1.9 Ontologies 19

2 Introduction to cancer genomics **23**
2.1 From DNA to the proteins, how does it work? 23
2.2 DNA duplication, key concepts 24
2.2.1 DNA duplication errors 24
2.3 Genetic mutations 24
2.3.1 Sequence alteration 24
2.3.2 Copy Number Alteration 26
2.4 Oncogenesis 26

3 LAS **27**
3.1 Basic concepts 27
3.2 Architecture 28
3.3 Modules 29

4 cBioPortal **33**
4.1 Data structure 33
4.2 Querying Individual Cancer Studies 34
4.2.1 Download Data 35
4.2.2 Query 36

5 Browser Extensions **37**
5.1 Google Chrome Extensions 37
5.1.1 The basics 37
5.1.2 Architecture 37

5.1.3 Using the chrome.*APIs 41
5.1.4 Communication between pages 42
5.1.5 Content Security Policy 42
5.1.6 Message Passing 43
5.2 Firefox Extensions 44
5.2.1 JavaScript APIs 44
5.2.2 Miscellaneous incompatibilities 45

II Design and Implementation 47

6 Architecture 49
6.1 Architecture and data flow 49
6.1.1 cBioPortal data loading 49
6.1.2 cBioPortal Downloader 53
6.1.3 LAS sample annotation 54
6.2 cBioPortal functionalities 54
6.3 LAS annotations 56

7 cBioPortal Downloader 59
7.1 Data capture 59
7.2 Clustering 61
7.3 Data model and storage 63
7.4 Data browsing 65
7.4.1 Sample data panel 65
7.4.2 Cluster panel 65

8 LAS annotation 71
8.1 Data model 71
8.2 Export log 73

9 Use case 75

III Conclusions 89

10 Deployment and future development 91

Bibliography 93

List of Figures

1.1 Software and data integration. 6
1.2 Data integration methodologies. 7
1.3 Basic graph model. 12
1.4 Hypergraph model. 12
1.5 Hypernode graph model. 12
1.6 RDF graph model. 13
1.7 Property graph model. 13
1.8 Example graph. 15
1.9 AllegroGraph logical structure. 15
1.10 MongoDB documents in a graph structure. 17
1.11 Taxonomy hierarchy example. 21

3.1 GenealogyID structure 28
3.2 Graph representation of LAS entity relationships 29
3.3 LAS query blocks. 31

4.1 Cancer study summary view 34
4.2 Single cancer study query 35
4.3 Example of downloaded text file content. 36

5.1 Extension icon. 39
5.2 Extension popup. 39

6.1 Framework architecture. 50
6.2 Browser extension icon. 54
6.3 OncoPrint view. 55
6.4 OncoPrint "View" dropdown menu. 55
6.5 OncoPrint patient details. 56
6.6 cBioPortal patient view. 56
6.7 OncoPrint "Clinical Tracks" dropdown menu. 57
6.8 OncoPrint clinical track visualization. 57
6.9 LAS cluster data representation. 58

7.1 cBioPortal data capture schema. 60
7.2 Cluster analysis result example. 62
7.3 Genomic data file structure. 64
7.4 Genomic sample data. 64
7.5 Clinical data file structure. 65
7.6 Clinical sample data. 65
7.7 cBioPortal Downloader page. 66
7.8 cBioPortal Downloader sample data panel. 67

7.9 cBioPortal downloader: CNA details for sample TCGA-02-0003-01 on gene EGFR. 67
7.10 cBioPortal Downloader cluster panel. 68
7.11 cBioPortal Downloader: cluster sample list. 68
7.12 cBioPortal Downloader: hybrid cluster. 69
7.13 cBioPortal Downloader: interesting/un-interesting clusters. 69
7.14 cBioPortal Downloader: send analysis to LAS. 69

8.1 Cluster data structure. 72
8.2 Cluster data structure: sample detail. 72
8.3 Layer 2 annotation model. 73
8.4 Example export log file structure. 74

9.1 Glioblastoma Multiforme import folder. 75
9.2 cBioPortal query (use-case). 76
9.3 cBioPortal download file with CNA data (use-case). 77
9.4 cBioPortal transposed download file with CNA data (use-case). 77
9.5 cBioPortal oncoprint patient visualization (use-case). 77
9.6 cBioPortal oncoprint sample visualization (use-case). 78
9.7 cBioPortal CNA event details (use-case). 78
9.8 cBioPortal patient view (use-case). 79
9.9 cBioPortal OncoPrint with clinical tracks (use-case). 80
9.10 cBioPortal Downloader view (use-case). 81
9.11 cBioPortal Downloader data panel (use-case). 82
9.12 cBioPortal Downloader data panel: alteration details (use-case). 82
9.13 cBioPortal Downloader cluster panel (use-case). 83
9.14 cBioPortal Downloader cluster panel: cluster sample list (use-case). 83
9.15 cBioPortal Downloader cluster panel: cluster sample alteration (use-case). 84
9.16 cBioPortal Downloader cluster panel: dimension removal (use-case). 85
9.17 cBioPortal Downloader cluster panel: cluster merging and uninteresting cluster flagging (use-case). 86
9.18 Layer 2 annotations for cluster 2 (use-case). 87

List of Tables

1.1 RDFS main properties. 10
1.2 Graph-based data model. 11
1.3 Cypher clauses. 14
1.4 SPARQL query types. 16

2.1 HGVS recommendations. 26

5.1 Manifest file features. 38

6.1 cBioPortal meta study file fields. 51
6.2 cBioPortal meta cancer type file fields. 51
6.3 cBioPortal cancer data file columns. 52
6.4 cBioPortal clinical meta data file fields. 52

7.1 Dataset example. 62
7.2 RECIST tumor change classification. 63

Introduction

Modern high-throughput technologies produce a huge amount of data which can be further combined with data coming from open databases or public datasets and, then, used by a variety of automatic analysis tools: that is why software and data integration have become a core issue for organizations interested in knowledge extraction, cancer research laboratories being one of them.
The LAS system was born to support Candiolo Cancer Institute (IRCCS) researchers in data management activities. It helps them to collect experimental data in the laboratories and to integrate them with other domain knowledge by means of a semantic knowledge base, modeled as a graph database. Within the LAS genomic data model gene alterations and clinical information are represented by semantic statements, named "annotations", establishing a conceptual relationship between the samples and their features. To ensure semantic coherence and adopt a standardized nomenclature, all relevant concepts from the genomic and biological domains used for labeling samples have been drawn from a number of public, freely accessible databases and ontologies. Concepts are interlinked with the other ones, according to both general-purpose semantic relationships such as containment ("part of") and generalization ("is a"), and domain specific relationships (e.g. indicating an underlying biochemical process, as in "is transcribed from"). New concepts and relationships, as well as new domains of interest, may be added or layered as needed, to account for novel findings and broaden the spectrum of investigation. An annotation is represented within the graph database as a node of type "annotation" with a pair of incoming and outgoing edges - one linking the biological sample to the annotation node, and the other linking the annotation node to the reference node in the knowledge base. The annotation node is often linked to other nodes, such as the process that produced the annotation or the raw experimental data. The LAS knowledge base can be queried by means of an unified query system.
The cBioPortal for Cancer Genomics is an open-source platform for statistical analysis and visual mining of biomedical data developed at Memorial Sloan-Kettering Cancer Center (MSKCC). It allows to perform an interactive exploration of multi-dimensional cancer genomics data sets, where data are organized by samples and genes, and can be associated to other clinical attributes (e.g. drug response). The user can request information about some genomic profiles (mutations, copy number alterations, protein expression, etc.) investigated by a cancer study and associated to the reference dataset for that study. The query answer is presented by means of views, where data are organized in diagrams, associated to the statistics the portal computed. One specific view, named OncoPrint associates each gene, the user inserted in its query, to a row made of little cells, each one corresponding to a genomic event on a single sample. Different genomic events are associated to different colors and, in this way, the user can visually spot trends and patterns among the given events. Additionally, the user can enrich this graphical analysis with clinical features, called "clinical tracks", which are additional attributes available for that study. Each additional track is represented as a new row in the OncoPrint, where colors and glyphs allow to distinguish different events and behaviors. This allows to understand how a certain clinical evidence is correlated to the genomic events of interest.

The aim of this thesis has been to integrate the LAS system with the functionalities provided by the cBioPortal.

Chapter 1 presents the main concepts and issues of system and data integration. It also introduces the graph databases as a possible solution and presents some of the most commonly used database management systems and query languages. It also introduces the main concepts of ontology development. Chapter 2 provides a brief introduction to Genetics and cancer genomics. It presents an overview of the main molecular processes that regulate genetic activities within living beings and presents the most relevant phenomena underlying oncogenesis. Chapter 3 describes the main concepts, modules and working principles of the LAS platform, focusing on the molecular annotation model, underlying its knowledge base. Chapter 4 provides an overview of the cBioPortal functionalities and query system. Chapter 5 presents the main concepts, components and working principles of browser extensions, focusing on Google Chrome and Mozilla Firefox ones. Chapter 6 provides an high-level description of the architecture and the dataflow of the framework which has been developed. Chapter 7 provides the details of the cBioPortal Downloader working principles and visualization tools, focusing on the clustering algorithm which has been designed. Chapter 8 provides a description of the data model which has been developed to produce the new LAS annotations. Chapter 9 illustrates how the framework works by means of a use-case scenario. Chapter 10 collects some final remarks about the tool and its possible future development.

Part I

Theoretical and Technological Background

Chapter 1

Third-Party Software and Data Integration

One way to reduce the burden of high costs of many software development and maintenance projects is by integrating third-party software with your own application [1]: it allows to benefit from pre-existing applications without the need of re-implementing their functionalities. Third-party software integration may be performed in two ways [2]:

- integration: the third-party software is hardwired to your application. The products work as one, sharing the same code and database: they are like pieces of a puzzle where each one of them interlocks with the other;
- interface: the third-party software communicates with your application through an interface. A possible technological solution is to let the components communicate through their application programming interfaces (APIs).

One downside of using an interface is that it does not allow you to synchronize data between the systems in real-time. Moreover, since the interfaced systems do not share the same database, they also require a data integration process, in order to maintain "mappings" between the systems. On the other side, the intrinsic value of this approach relies in the fact that it does not need to hardwire the policy and the behavior of the integration inside the applications. When the system evolves, there is not need to go inside each application and re-code it to meet the new integration requirements [1].

For all of these reasons, interfaces are preferred to full integration, in many cases; in fact, they allow to integrate different applications, based on different technologies and communication protocols, with little effort and in a completely transparent way.

1.1 Data Integration

As mentioned in the previous section, integrating heterogeneous systems, by means of interfaces, requires also to perform a data integration process (Figure 1.1). Specifically, data integration addresses the issues related to fetching data from different sources and, then, manipulating and combining them, in order to achieve interoperability and data consistency between the systems.

1.1.1 Methodologies

In computational sciences the theoretical frameworks for data integration have been classified into two major categories, “eager” and “lazy”. In the eager approach data are copied over a

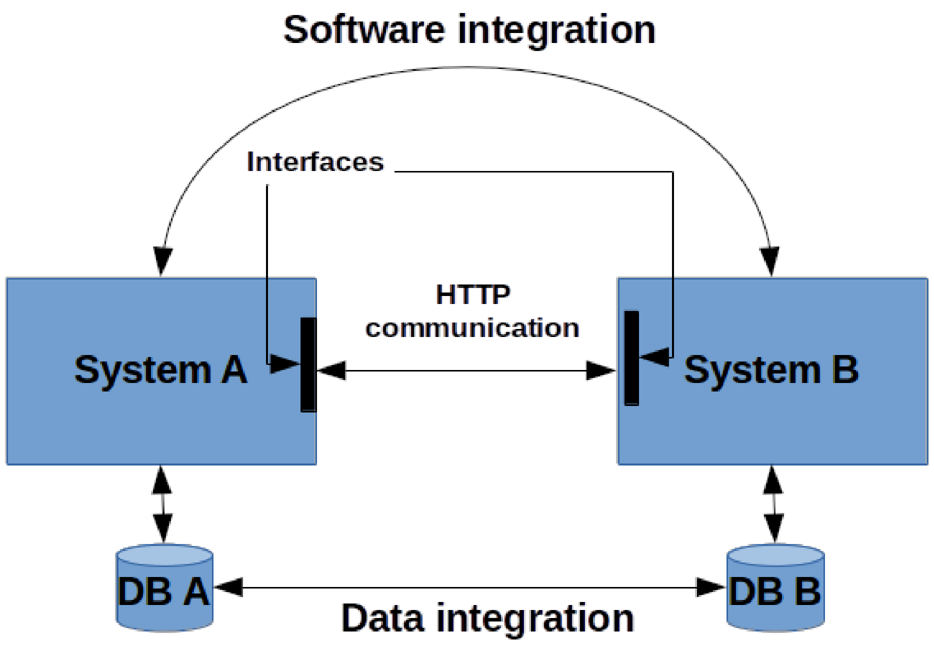


Figure 1.1: Software and data integration.

global and normalized schema and stored into a central repository; whereas in the lazy approach data remain in their original databases and no modification to their schema is made: they are integrated on demand based on a global logical schema used to map data between sources [3].

Each of the two main approaches presents its own complexities. In fact in the eager approach, developers face the trouble of keeping the data updated and consistent, protecting them from being corrupted. While in the lazy approach the main difficulty consists in defining a unified mechanism to retrieve data from different sources, and then combine and present them to the final user in a meaningful format.

In biology both of these approaches are used in different ways. Figure 1.2 shows some common solutions, with possible real-world application, which are discussed below.

1.1.2 Data centralization

A **centralized database** is a database where data are physically stored and maintained in the same place, holding the same schema. This is the simplest and most robust solution, since it should be compliant with the ACID properties, provide bigger data security and portability, and user's queries don't require any special treatment; on the other side it is not useful for the purposes of this discussion, since it doesn't respond to the need of integrating heterogeneous data coming from different sources.

1.1.3 Data warehousing

A **data warehouse** is the first actual solution to achieve data integration in a distributed scenario. DWs are central repositories where data coming from different and disparate sources are integrated. They typically store historical data from the same company and are used to create

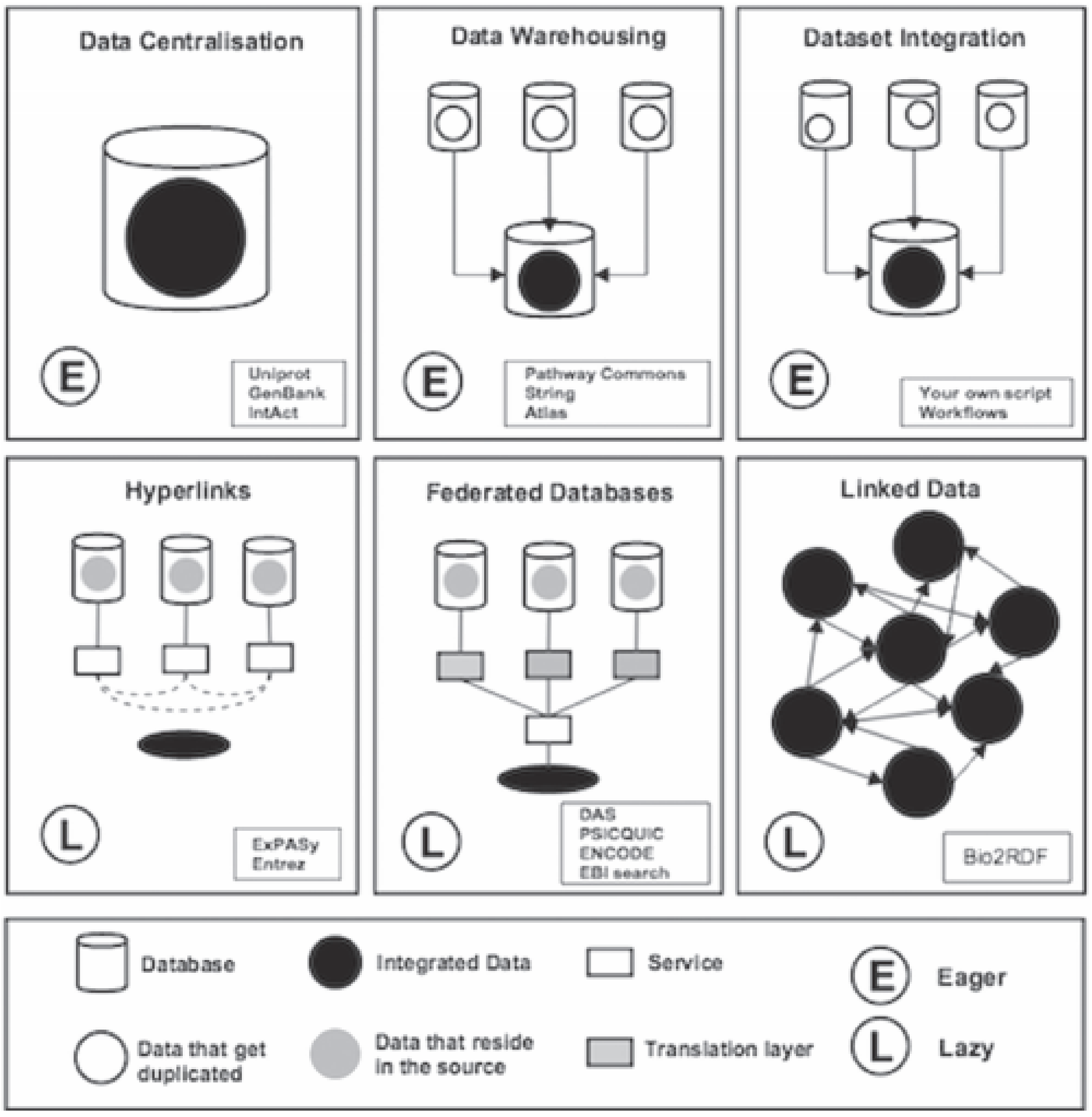


Figure 1.2: Data integration methodologies.

analytical reports.

Before being stored into the DW, data coming from operational databases, need to be cleaned and normalized. This is often referred to as an ETL (*Extraction, Transformation and Load*) process [4]. ETL operations typically require a *staging database* where data coming from operational databases are transformed and integrated. Once data are ready to be stored, they are loaded into the DW, where they will be mainly queried and hardly ever updated, since updates are heavy operations.

The main advantage of this solution is its robustness, since it guarantees a good quality of data which are actually integrated, so users' queries can be performed directly on the physical data, since the DW presents a single data model. Moreover, it allows to maintain historical data, even if the source transaction systems do not. Finally, it doesn't impact operational systems. On the other side, it doesn't allow any flexibility, the ETL operations may represent a difficult task and it requires to move a big amount of data over the network which could be a bottleneck in the overall framework.

1.1.4 Dataset integration

Datasets integration can also be made by in-house scripts accessing distributed databases and downloading data to a local repository rather than letting an automatic system fetch them. This solution has basically the same pros and cons of the data warehousing one, allowing more flexibility but charging the user with the additional difficulty of managing the ETL and the integration processes.

1.1.5 Hyperlinks

Link integration directly cross-references a data entry in a data source with another entry in another data source. Users follow the references. As these entries are usually presented as Web pages, the users surf across datasets by following **hyperlinks**. The approach leans heavily on ontology and identity authorities to enable the cross-referencing [5]. This approach relies on **data services**, which are Web services specialized in offering data access and data manipulation services through simple Web APIs. Data services are typically implemented by the data sources themselves.

This is a lazy approach, so the main advantage is that data need no manipulation, normalization or any kind of transfer, since they're just accessed as Web resources as any other Web page. The main disadvantage is that, if the user needs to explore, mine and analyze the retrieved data, he still has to normalize them, otherwise no analysis makes sense and no automatic tool can be used.

1.1.6 Federated datasets

A **federated database system** is a meta-database management system (DBMS) which maps autonomous and, possibly, heterogeneous database systems into a single federated database. The data sources remain separate while the system presents a unique interface to the user who can retrieve data from multiple sources with a single query. At this point, the federated database system decomposes the query, wraps it into sub-queries performed into the query language which each single database expects, and then combines the retrieved data and presents them to the user. Obviously, this process requires an intermediate stage where queries are decomposed and data are combined [6]. In a way, a federated database system is a **virtual database** which allows the users to query multiple data sources transparently.

This approach is very powerful and, at the same time, light and portable. Whereas it has the disadvantage of having a single point of failure: if the federated DBMS, which is loaded with a lot of work, breaks down, the whole system becomes unusable.

1.1.7 Linked data

The **linked data** approach is based on the idea of building an integrated warehouse on the **semantic Web**, where resources have been assigned to *Universal Resource Identifiers* (URIs) and data are connected on the web [7]. In this way, users can use a graphical interface (GUI) which provides them hyperlinks to navigate through a semantic network, connecting data from multiple data providers.

The biggest novelty of this solution is the semantic network itself with all of its pros and cons. On one side, a semantic network allows a more precise exploration of data, since the relationships among them are based on conceptual links. On the other side, building a semantic network is a tricky task and final users may face some troubles in trying to learn this new. Anyhow, this is the most traveled path nowadays, since the potentialities of semantic networks seem to obfuscate the critical points.

RDF. One fundamental characteristic of this solution is that data, which are represented as concepts, need to be described in a standard way. On this purpose W3C[1], proposed the *Resource Description Framework* (RDF) as a standard for coding, exchanging and re-using structured metadata which allows interoperability among Web applications which rely on the semantic Web [8].
RDF data model breaks the knowledge in *statements* about *resources* (in particular web resources) in expressions of the form *subject–predicate–object*, known as *triples*. The subject denotes the resource, and the predicate denotes traits or aspects of the resource, and expresses a relationship between the subject and the object. A collection of RDF statements intrinsically represents a labeled, directed multi-graph.
Here is an example triple stating that Dante Alighieri (subject) is the author (predicate) of the "Divina Commedia":

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:au="http://description.org/schema/">
  <rdf:Description about="http://www.book.it/Divina_Commedia/">
    <au:author>Dante Alighieri</au:author>
  </rdf:Description>
</rdf:RDF>
```

RDFS. RDF, per se, does not allow to define resources and properties: this is managed by RDF Schema (RDFS) [9]. RDF Schema describes RDF resources and properties in terms of *classes* and *properties*. Classes in RDFS are similar to classes in object oriented programming languages: resources are defined as instances of classes, and subclasses of classes. Differently from an XML schema, RDFS does not limit the document structure, but it provides useful information about the document itself. Basically, it provides a mechanism to assign data types to RDF resources. The schema is defined with RDF syntax too. Table 1.1 shows the main RDFS features:

[1]The *World Wide Web Consortium* is a non governmental organization occupied in expanding all Web potentialities. Its main activity is defining technical standards for the Web.

Tag	Usage
`rdfs:Resource`	All RDF objects are resources and each resources are instances of this class.
`rdfs:Literal`	`rdfs:Resource` subclass which represents a string.
`rdf:Property`	`rdfs:Resource` subclass which represents an RDF property.
`rdfs:Class`	It corresponds to the object-programming concept of class. When a new class is defined, it must come with the `rdf:type` set to `rdfs:Class`.
`rdfs:subClassOf`	It specifies that a class is also an instance of a super-class. It defines hereditary.

Table 1.1: RDFS main properties.

The following example shows how RDFS is used. It is a simple RDF document where resources are treated as class intances. Specifically, it defines two classes, one named "Animal" and another one named "Cat"; additionally, the "Cat" class is also a subclass of the "Animal class:

```
 <rdf:Description rdf:ID="Animal">
    <rdf:type
     rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>

  <rdf:Description rdf:ID="Cat">
    <rdf:type
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Animal"/>
  </rdf:Description>
```

1.1.8 Technological solution: graph databases

According to the domain of application, the most appropriate technological solution has to be identified and **graph databases**, a technology widely used among science and IT communities, have been proved to be particularly flexible and highly performing in representing complex biological relationships [10].

A graph database is a NoSQL database which models data in terms of **nodes** (entities) and **edges** (relationships). It is more expressive than a traditional relational database and very useful for situations with heavily interconnected data. It allows fast relationship-based searches and a flexible representation of data, which are typically stored in form of documents (usually XML or JSON documents) that permit to change or add new attributes and objects. On the other side, graph databases, which are optimized to be fast and flexible, don't care about data redundancy and this, in particular cases, may overload the system.

Graph Database Models

Graph databases can model data in different ways. Table 1.2 shows the existing graph-based data models [11].

Graph Data Model	Description

Basic Graph Data Model	A directed graph with nodes and edges labeled by some vocabulary. Each node is labeled with a symbol called *type*, which has associated a domain of possible values. In the same way, each edge has assigned a label representing a relation between types (Figure 1.3).
Hypergraph Data Model	A generalization of graphs in which an edge, or *hyperedge*, can join any number of nodes. While graph edges are pairs of nodes, hyperedges are arbitrary sets of nodes, and can therefore contain an arbitrary number of nodes (Figure 1.4).
Hypernode Data Model	A directed graph whose nodes can themselves be graphs, allowing nesting of graphs. A key feature is its inherent ability to *encapsulate information* (Figure 1.5).
Resource Description Framework Model	A recommendation of the W3C, originally designed to represent metadata. An atomic RDF expression is a triple consisting of a subject (the resource being described), a predicate (the property) and an object (the property value). Each triple represents a statement of a relationship between the subject and the object. A general RDF expression is a set of such triples, which can be intuitively considered as a labeled graph, although formally is not a graph (Figure 1.6).
Property Graph Data Model	A directed, labeled, attributed multigraph. Both nodes and edges are labeled and can have any number of properties, edges are directed and there can be multiple edges between any two vertices. Properties are expressed in form of key-value pairs representing metadata for nodes and edges (Figure 1.7).

Table 1.2: Graph-based data model.

Graph Database Examples

Neo4j. Neo4j is the most famous and deployed graph database management system [12]. It is an open source software, completely implemented in Java, built from the ground to be a graph database.

Neo4j relies on the property graph data model. It allows to assign, to both nodes and edges, an arbitrary number of properties, represented as key-value pairs. The key is, typically, a string representing the property name, unique for the element it belongs to, while the the value may be a number, a string or an array (of numbers or strings).

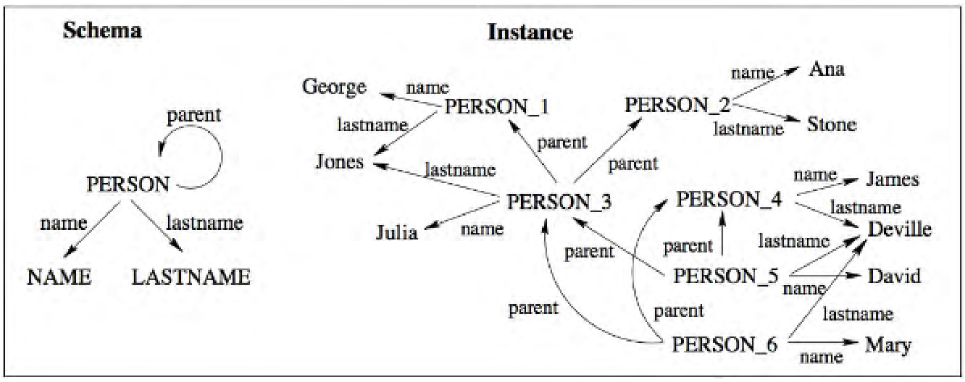


Figure 1.3: Basic graph model.

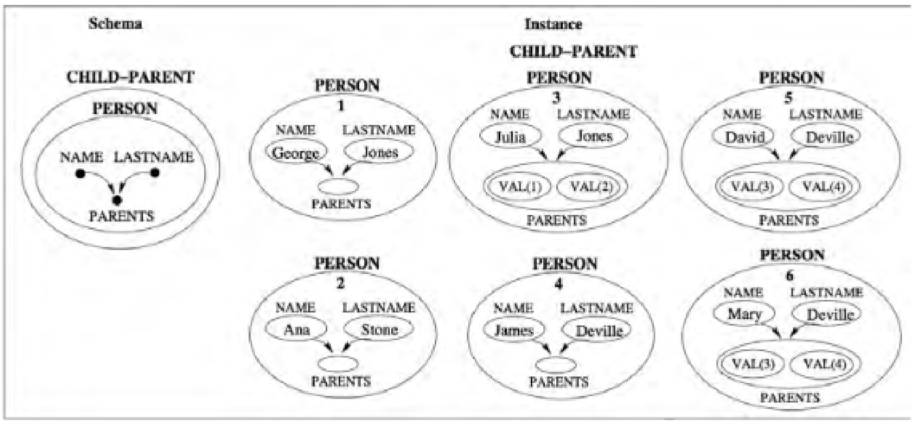


Figure 1.4: Hypergraph model.

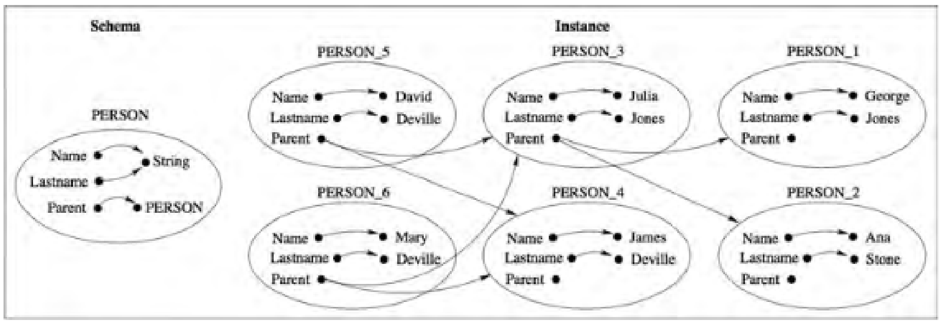


Figure 1.5: Hypernode graph model.

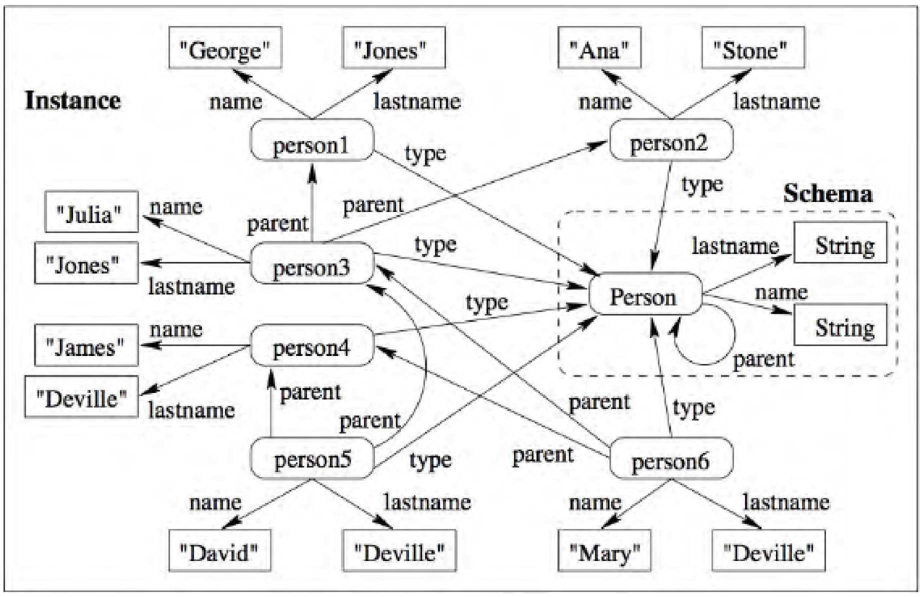


Figure 1.6: RDF graph model.

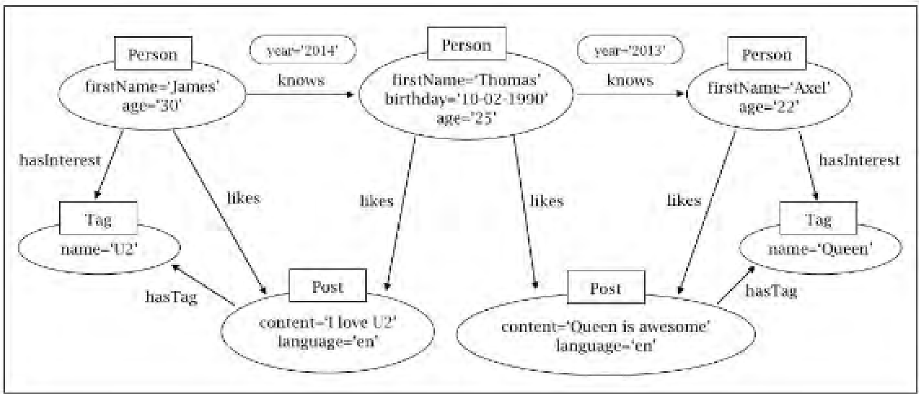


Figure 1.7: Property graph model.

Nodes must have a unique ID, automatically assigned by Neo4j at creation time, and can have one or more "labels". Labels are typically used to characterize nodes and allow to group them according to their type. Relationships, in their turn, must have a type, which must be specified by the user at creation time. Types can be assigned arbitrary, they don't require to be declared. However, nodes and relationships are simply low-level building blocks. The real strength of property graphs is the possibility to build *patterns* of nodes and relationships, allowing to conceive quite complex concepts. For example, a simple pattern which connects a couple of nodes, (e.g `Person LIVES_IN City`), allows to express a conceptualization which is more complex than the elementary ideas expressed by the single nodes. The more elaborate the pattern becomes, the more complex the expressed concepts are.
Neo4j data model is particularly useful when semantic searches should be performed.

Neo4j databases can be managed and accessed through a declarative query language, named **Cypher**, developed by Neo4j team. Similarly to SQL, Cypher builds its queries using clauses, chained together [13]. Table 1.3 shows some of the clauses, typically, used to access the graph

Clause	Description
`MATCH`	The graph pattern to match. This is the most common way to get data from the graph.
`WHERE`	Not a clause in its own right, but rather a constraint to a pattern, which filters the intermediate results.
`RETURN`	What to return.
`CREATE` (and `DELETE`)	Create (and delete) nodes and relationships.
`SET` (and `REMOVE`)	Set (and remove) values to properties and add labels on nodes.
`WITH`	Pass an intermediate result.

Table 1.3: Cypher clauses.

Cypher is strongly based on patterns: they allow to match desired graph structures and use them for further processing. It uses a form of ASCII art[2] to represent nodes and relationships and build up patterns, which, combined with the appropriate clauses, create the Neo4j queries. Specifically, nodes are represented by means of round parentheses, (e.g. `(node)`), while undirected relationships are represented by a pair of dashes, (`--`), which become directed if a arrowhead is added in the appropriate direction (`<--`, `-->`). Bracketed expressions (`[...]`) allow to add details such as variables, properties or type information. Finally, node representation allows to specify a variable to save the returned element and to filter by a certain property value, specifying it in a JSON format (e.g. `(n: node propKey : "propValue" )`).

Figure 1.8 shows an example graph describing the users of a social network, characterized by a *name* property, and their relationships. Here is a query which finds a user called 'John' and 'John's' friends (though not his direct friends) and returns both 'John' and any friends-of-friends that are found:

```
MATCH (john {name: 'John'})-[:friend]->()-[:friend]->(fof)
RETURN john.name, fof.name
```

The query result would be:

[2]a graphic design technique which consists in drawing pictures by means of the printable ASCII symbols

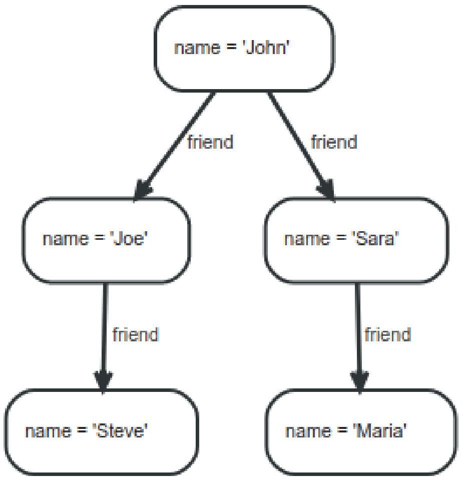


Figure 1.8: Example graph.

```
+-----------------------------+
| john.name  | fof.name  |
+-----------------------------+
| "John"     | "Maria"   |
| "John"     | "Steve"   |
+-----------------------------+
2 rows
```

AllegroGraph. AllegroGraph is a *triplestore*: a database and application framework for building Semantic Web applications capable of storing data and metadata as RDF triples (see Section 1.1.7)[14]. Figure 1.9 shows AllegroGraph building blocks: its bulk is made of *assertions* made

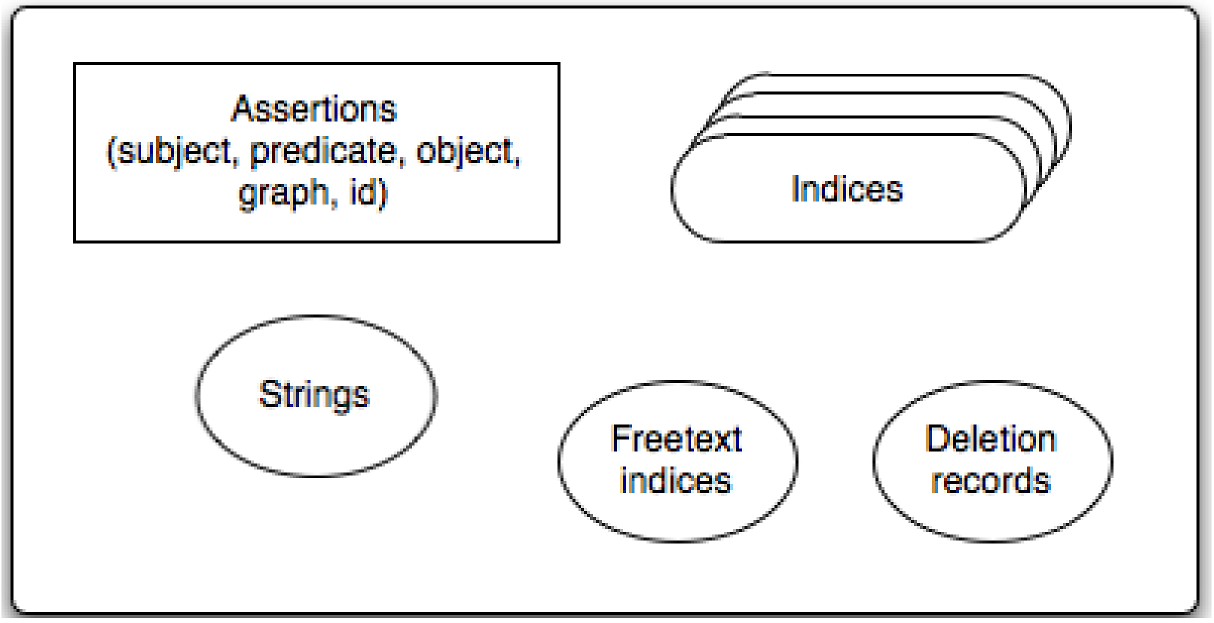


Figure 1.9: AllegroGraph logical structure.

of *subject*, *object*, *predicate*, *graph name* and *triple-id*. All of this five fields are strings of arbitrary size, which is automatically concatenated to a number to avoid duplications. To speed queries, AllegroGraph creates *indices* which contain the assertions plus additional information. Finally, AllegroGraph is capable of keeping track of the deleted triples.

AllegroGraph triples can be queried through various query APIs, SPARQL being one of them. SPARQL is a semantic query language for RDF graph databases, recommended by the W3C [15]. It is recognized as the standard technology for the Linked Data framework, for the semantic web (see Section 1.1.7).A SPARQL query consists of a set of triple patterns in which each element can be a variable (wildcard). The variables are then replaced by the values found by matching the patterns in the query to the triples in the dataset. Table 1.4 shows SPARQL types of queries; each of them takes a `WHERE` block to select the data to return, with the exception of the `DESCRIBE` query, where it is optional.

Query type	**description**
`SELECT` query	Used to extract raw values from a SPARQL endpoint, in a table format.
`CONSTRUCT` query	Used to extract information from the SPARQL endpoint, represented in form of valid RDF.
`ASK` query	Used to provide a simple True/False result for a query on a SPARQL endpoint.
`DESCRIBE` query	Used to extract an RDF graph from the SPARQL endpoint, the content of which is left to the endpoint to decide based on what the maintainer deems as useful information.

Table 1.4: SPARQL query types.

Here is a SPARQL query which models the question: "What are all the country capitals in Africa?":

```
PREFIX  ex: <http://example.com/exampleOntology#>
SELECT  ?capital
    ?country
WHERE
{
  ?x  ex:cityname       ?capital   ;
      ex:isCapitalOf    ?y         .
  ?y  ex:countryname    ?country   ;
      ex:isInContinent  ex:Africa  .
}
```

Variables are indicated by a "?" or "$" prefix. Bindings for `?capital` and the `?country` will be returned.

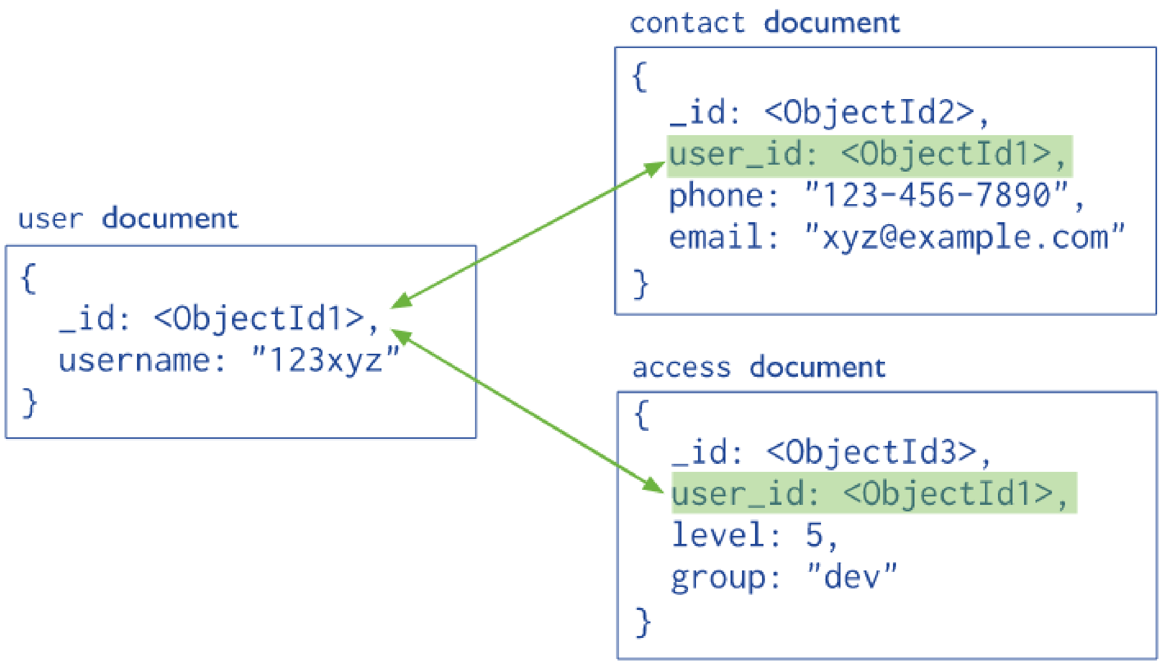
SPARQL allows to query graph databases but also any database that can be viewed as RDF via middleware. For example, a relational database or a document database, such as MongoDB, can be queried with SPARQL by using a mapping software, capable of mapping the content of a non-RDF database to RDF [16].

MongoDB. MongoDB is a, free and open-source, document-based database management system which stores data in flexible, JSON documents. This means that the fields can vary from document to document and data structure can be changed over the time [17].

MongoDB stores data in a binary representation called BSON (Binary JSON). The BSON encoding extends the popular JSON (JavaScript Object Notation) representation to include additional types such as int, long, date, floating point, and decimal128. BSON documents contain one or more fields, and each field contains a value of a specific data type, including arrays, binary data and sub-documents. MongoDB BSON documents are closely aligned to the structure of objects in the programming language. This makes it simpler and faster for developers to model how data in the application will map to data stored in the database.

MongoDB documents tend to have all data for a given record in a single document, whereas in a relational database information for a given record is usually spread across many tables.As a result, data are localized and joins are hardly-ever needed, even if supported to provide additional flexibility.

Moreover, it allows to define relationships among documents, by means of external keys referencing one document from another: this means, that data objects can be organized in a graph structure whose nodes are represented by the documents, while the edges are modeled by the external reference keys (Figure 1.10).



```
user document
{
  _id: <ObjectId1>,
  username: "123xyz"
}

contact document
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}

access document
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Figure 1.10: MongoDB documents in a graph structure.

MongoDB provides also the possibility to use and validate a schema. This functionality can be exploited to use the document database as an RDF triplestore, allowing to accesses MongoDB documents with AllegroGraph and query them with SPARQL [18]. In order to learn the procedures required to synchronize the two platforms, please visit `https://franz.com/agraph/support/documentation/current/mongo-interface.html`. Here, it is provided just an example, to illustrate how it is possible to use a MongoDB document database to model an RDF-based graph.

Here is a MongoDB JSON document:

```
{   _id: "1",
    Name: "Finn",
    Occupation: "Great Hero",
    Alignment: "Good" }
{   _id: "2",
    Name: "Jake",
    Occupation: "Best Friend",
    Alignment: "Good" }
{   _id: "3",
    Name: "Ice King",
    Occupation: "King",
    Alignment: "Bad" }
{   _id: "4",
  Name: "Gunter",
  Occupation: "Servant",
  Alignment: "Bad"
}
```

In this case, the documents present a schema made of three attributes, named "Name", "Occupation" and "Alignment" and an automatically generated ID.
Here is the corresponding linking triples in AllegroGraph:

```
@prefix f: <http://www.franz.com/> .
@prefix id: <http://www.example.com/id#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

id:subject1 f:hasMongoId "1" ;
      f:likes id:subject3 ;
            f:likes id:subject2 .

id:subject2 f:hasMongoId "2" .
id:subject3 f:hasMongoId "3" .
id:subject4 f:hasMongoId "4" .
```

It maps each MongoDB document to one RDF triple, by linking the IDs used in the two databases to identify the same entity, through a `:hasMongoId` relationship; in addition, in introduces a `:likes` relationship to specify that a user likes another one. Here is a SPARQL query which finds people with Good Alignment who like people with Bad Alignment:

```
prefix mongo: <http://franz.com/ns/allegrograph/4.7/mongo/>
prefix f: <http://www.franz.com/>
select ?good ?bad {
    ?good mongo:find '{Alignment:"Good"}' .
    ?bad mongo:find '{Alignment:"Bad"}' .
    ?good f:likes ?bad .
}
```

The result is:

```
?good     ?bad
==================
subject1  subject3
```

1.1.9 Ontologies

The underlying assumption to integrate data is that they should be compliant with some sort of standards. **Standards** define rules and norms to describe entities in a uniform way. For instance, A, C, G and T are a standard way to name DNA nucleotides. Without them it is almost impossible to integrate data.

Ontologies are an instrument used for standardization in the biological field. The word "Ontology" represents a Philosophy sub-domain which investigates the global and intrinsic features of what exists. AI borrowed this concept to refer to a way to represent knowledge. Basically, an ontology is a domain-specific vocabulary where terms represent concepts and links represent correlations among them (e.g. "is type of", "is part of"). Such a vocabulary is language-independent. Let's consider an ontology designed to model an electronic-devices domain which might include a bunch of entities - transistors, operational amplifiers, and voltages - and the relations among them - operational amplifiers are *type-of* electronic device, and transistors are *part-of* operational amplifiers. This ontology may be translated to Italian or to French, it does not matter: the conceptualization it represents does not change [19]. That's what an on ontology is: a formal representation of a knowledge base in terms of entities and relationships, where the entities represent concepts and the relationships describe how entities are conceptually linked. Applied ontology offers a strategy for the organization of scientific information in computer-tractable form, drawing on concepts not only from computer and information science but also from linguistics, logic, and philosophy [20].

One way to define ontologies is my means of a semantic web language, called **Web Ontology Language** (**OWL**), developed by the W3C team [21], to let the machines "reason" over the RDF schemas.

OWL provides three increasingly expressive sub-languages designed for use of specific communities. *OWL Lite* supports those users interested in a classification hierarchy or simple constraints. *OWL DL* supports those users who want the maximum expressiveness saving computational completeness and decidability (operations are guaranteed to be computable and completed in a reasonable time). *OWL Full* is for users who want the maximum expressiveness with no computational guarantees. Each of these sub-languages is an extension of its simpler predecessor, while the opposite assumption is not true. The expressiveness of the language depends on the logic it is based on (first-order logic or its modifications).

Just to show what OWL looks like, here is a quick example [22]:

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:dc="http://purl.org/dc/elements/1.1/">

<!-- OWL Header Example -->
<owl:Ontology rdf:about="http://www.linkeddatatools.com/plants">
  <dc:title>The LinkedDataTools.com Example Plant Ontology</dc:title>
  <dc:description>An example ontology written for the LinkedDataTools.com RDFS
      & OWL introduction tutorial</dc:description>
</owl:Ontology>

<!-- OWL Class Definition Example -->
```

```
<owl:Class rdf:about="http://www.linkeddatatools.com/plants#planttype">
  <rdfs:label>The plant type</rdfs:label>
  <rdfs:comment>The class of plant types.</rdfs:comment>
</owl:Class>

<!-- OWL Subclass Definition - Flower -->
<owl:Class rdf:about="http://www.linkeddatatools.com/plants#flowers">

  <!-- Flowers is a subclassification of planttype -->
  <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/plants#
      planttype"/>

  <rdfs:label>Flowering plants</rdfs:label>
  <rdfs:comment>Flowering plants, also known as angiosperms.</rdfs:comment>

</owl:Class>

<!-- OWL Subclass Definition - Shrub -->
<owl:Class rdf:about="http://www.linkeddatatools.com/plants#shrubs">

  <!-- Shrubs is a subclassification of planttype -->
  <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/plants#
      planttype"/>

  <rdfs:label>Shrubbery</rdfs:label>
  <rdfs:comment>Shrubs, a type of plant which branches from the base.</rdfs:
      comment>

</owl:Class>

<!-- Define the family property -->
<owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/plants#family"
    />

<!-- Define the similarlyPopularTo property -->
<owl:ObjectProperty rdf:about="http://www.linkeddatatools.com/plants#
    similarlyPopularTo"/>

<!-- Define the Orchid class instance -->
<rdf:Description rdf:about="http://www.linkeddatatools.com/plants#orchid">

  <!-- Orchid is an individual (instance) of the flowers class -->
  <rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>

  <!-- The orchid is part of the 'Orchidaceae' family -->
  <plants:family>Orchidaceae</plants:family>

  <!-- The orchid is similarly popular to the magnolia -->
  <plants:similarlyPopularTo rdf:resource="http://www.linkeddatatools.com/
      plants#magnolia"/>

</rdf:Description>

<!-- Define the Magnolia class instance -->
<rdf:Description rdf:about="http://www.linkeddatatools.com/plants#magnolia">

  <!-- Magnolia is an individual (instance) of the flowers class -->
  <rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>

  <!-- The magnolia is part of the 'Magnoliaceae' family -->
  <plants:family>Magnoliaceae</plants:family>
  <!-- The magnolia is similarly popular to the orchid -->
  <plants:similarlyPopularTo rdf:resource="http://www.linkeddatatools.com/
      plants#orchid"/>

</rdf:Description>

</rdf:RDF>
```

First of all, the ontology must be defined in RDF. Moreover, even if it is not required, it is a good practice to define a header where all information need to understand the ontology should be placed. In the example, a title and a description have been included, but it is a good idea to include a version number too.
Class and *subclasses* are used to classify terms in terms of semantics: a class is a group of individuals sharing common characteristics. The *individuals*, conversely, are instances of a given class. Here, a class which represents all plants have been defined. Then plants are split into two subclasses: flowering plants and the shrubs.
In other words, semantic terms have been organized in a hierarchy, which in the semantic web world, is known as a *taxonomy* (Figure 1.11). Individuals in OWL are related by *properties*.

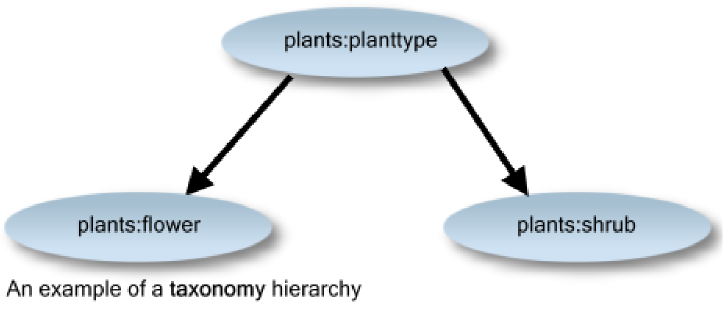


Figure 1.11: Taxonomy hierarchy example.

There are two types of properties: **Object properties**, which relate individuals of two classes, and **DataType properties**, which relate individuals to literal values. In the example, two properties have been defined: a data type property, to specify the name of the species family the Magnolia is part of, and an object property defining the concept of "similarity" between plants. Finally, two individuals have been defined: a magnolia, which is an instance of the "Flower" class, of type "Magnoliaceae" and similar to an orchid, and an orchid, an individual of the "Flower" class, of type "Orchidaceae" and similar to a magnolia.

Biomedical Ontology projects

Here is a list of some of the most known projects for ontology definition in the biomedical field. They are all part of the The Open Biological and Biomedical Ontology (OBO) Foundry project.

The OBO foundry. The Open Biological and Biomedical Ontology (OBO) Foundry is a collective of ontology developers who aim at defining a a family of interoperable ontologies in the biological and biomedical fields [23]. To achieve this, OBO Foundry participants voluntarily adhere to and contribute to the development of an evolving set of principles including open use, collaborative development, non-overlapping and strictly-scoped content, and common syntax and relations, based on the ontology model defined by the Basic Formal Ontology (BFO).

Basic Formal Ontology. The Basic Formal Ontology is the upper level ontology upon which OBO Foundry ontologies are built. BFO does not contain physical, chemical, biological or other terms which would properly fall within the special sciences domains: it is focused on the task of providing an upper ontology to be used in support of domain ontologies developed for scientific research [20]. The structure of BFO is based on a division of entities into two disjoint categories of "continuant" and "occurrent", the former comprehending for example objects and spatial regions, the latter comprehending processes conceived as extended through (or as spanning) time. So it provides both a three-dimensional and a four-dimensional perspective of reality.

Gene Ontology. The Gene Ontology is a project to provide a uniform way to describe the functions of the gene products from organisms across al kingdoms of life. It's knowledge base is composed of two main parts: the Gene Ontology which provides the conceptual correlation between the biological functions ('terms') and their relationships to each other; the corpus of GO annotations, specifying evidence-based statements relating a specific gene product (e.g. "actin") to a specif ontology term (e.g. " heart contraction") [24].

Sequence Ontology. The Sequence Ontology (SO) is a collaborative ontology project for the definition of sequence features used in biological sequence annotation [25]. It includes different kind of features which can be found on the sequence (e.g. "exon", "binding_site", etc.). It defines also biomaterial features which are intended for use in experiments (e.g. PCR_product") and experimental features, which are the result of an experiment.

Ontology for Biomedical Investigations. The Ontology for Biomedical Investigations (OBI) is an integrated ontology that provides terms with precisely defined meanings to describe all aspects of how investigations in the biological and medical domains are conducted. It defines more than 2500 terms for assays, devices, objectives and more (e.g. "assay", "genotyping by high throughput sequencing assay", etc.) to cover all phases of the investigation process, such as planning, execution and reporting. It represents information and material entities that participate in these processes, as well as roles and functions [26].

Foundational Model of Anatomy. The Foundational Model of Anatomy (FMA) is a reference ontology for the domain of anatomy. It is a symbolic representation of the canonical, phenotypic structure of an organism; a spatial-structural ontology of anatomical entities and relations which form the physical organization of an organism at all salient levels of granularity [27].

Human Disease Ontology. The Human Disease Ontology (DOID) is a community-driven, community-accepted ontology of diseases for clinical research and medicine inclusive of genetic, environmental and infectious diseases [28].

Chapter 2

Introduction to cancer genomics

The revelation of genetics and heredity reshaped all the fields of biomedical study, cancer research being only one of them. In fact today our understanding of how cancers arise and develop is mainly based on discoveries in the field of molecular biology and genetics [29].

Genetics is the study of what a computer scientist could call the source code of every living being, and a life scientist properly calls DNA. DNA determines all somatic features of individuals - e.g. eye color, height, hair color, etc., and regulates all the activities which occur in each living organism.

DNA (DeoxyriboNucleic Acid) is a double helix molecule which stands in the nucleus of the cells of every living organism and is made of a sequence of *nucleotides* organized into small units called *genes*. Nucleotides, in their turn, are molecules made of a sugar (*deoxyribose* or *ribose*) plus one of these four nitrogenous bases: *adenine* (A), *cytosine* (C), *guanine* (G) and *thymine* (T). Nucleotides also form **RNA** (RiboNucleic Acid), another nucleic acid, made of a single strand of nucleotides where *uracil* (U) replaces thymine.

DNA is the molecular basis of *heredity* and encodes the instructions needed to build proteins; RNA is involved in the process through which DNA instructions are read and used to build proteins [30].

2.1 From DNA to the proteins, how does it work?

One fundamental assumption about life is that proteins are made starting from DNA instructions, but how? The answer stands in the sequence of nucleotide bases on each filament, which differs from one kind of organism to another. In order to execute DNA instructions, two operations are needed:

- *transcription*, DNA controls RNA synthesis starting from a bunch of free nucleotides using itself as a mold
- *translation*, RNA moves from the cell nucleus to the cytoplasm, where it controls amino-acid synthesis into polypeptide chains which, when ready, fold into themselves to form proteins. In this phase, *ribosomes*, read RNA nucleotide bases which are organized in ternaries called *codons*. The order of appearance of the different codons determines how the corresponding amino-acids will be assembled to build the proteins.

$$\textbf{DNA} \xrightarrow{\textbf{transcription}} \textbf{RNA} \xrightarrow{\textbf{translation}} \textbf{PROTEIN}$$

This means that the order of the nucleotides in the DNA which generated the current filament of RNA **matters** and must be propagated correctly to the offspring.

2.2 DNA duplication, key concepts

DNA duplication, that is the process through which the DNA molecule replicates itself before a cell divides, is a fascinating and fundamental cellular activity. This is the moment when genetic propagation happens and if something goes wrong, the offspring cells will not be able to properly synthesize proteins. Let's briefly see how duplication works.

The nucleotides of the two DNA helixes are always paired in the same way: adenine with thymine (A-T) and guanine with cytosine (G-C). When duplication happens the two filaments are separated and some free nucleotides, which fluctuate in the cell, combine with them to build two complementary filaments which, then, combine to form an exact replica of the parent's DNA molecule.

2.2.1 DNA duplication errors

Mistakes are rare events and typically some biological mechanisms intervene for a fast fault recovery. But sometimes also the recovery mechanisms fail and this is when a **mutation** happens.

A mutation, as just mentioned, is some variation from the original genome which is originated during DNA duplication and not corrected. A mutation may or may not have consequences. If it affects a *non-encoding* portion of the DNA (an *intron*) it has no consequences; if it interests an *coding* portion of the DNA (an *exon*), it may have consequences which may be more or less severe. In fact there exists a tolerance mechanism: the same amino-acid can be synthesized starting from different codons; but if it is not the case, the resulting protein structure will be different and this may cause anomalies and dysfunctions in the affected organism.

2.3 Genetic mutations

The mutations causing tumors can be classified into multiple categories, two of which are particularly well-studied: **sequence alterations** and **copy number alterations**.

2.3.1 Sequence alteration

A sequence alteration (SA) happens when the nucleotide bases sequence is altered during duplication with the result that the corresponding amino-acid sequence is altered too. Specifically, the possible sequence alterations are:

- **point mutation**, a nucleotide is changed with another one
- **deletion**, one or more contiguous nucleotides are excised
- **insertion**, one or more nucleotides are added between two adjacent nucleotides
- **in-del**, similar to point mutation, but here a group of contiguous nucleotides is substituted by another sequence of nucleotides
- **duplication**, one or more nucleotides are duplicated

- **inversion**, the nucleotides replacing the original sequence are the reverse complement of the original sequence

These alterations may cause a *frameshift*, that is a misalignment in the codons, so that the DNA sequence is no more divisible by three; if, instead, the codons remain aligned, the mutation is said to be an *in-frame* mutation.

Alterations may have a different effect on the protein coded by the altered sequence: a **silent mutation** happens if the altered codons code for the same amino-acids and there are no consequences on the resulting protein; a **missense mutation** occurs if the altered codons code for different amino-acids with the effect that the resulting protein may be malformed with unexpected amino-acids in it; finally, a **nonsense mutation** happens when the altered codon is a premature stop sequence or a point-nonsense codon, so that the resulting protein is truncated, incomplete and non functional.

Sequence Variant Nomenclature, HGVS nomenclature recommendations

The *HGVS*, *Human Genome Variation Society*, is a society which aims to characterize human genome variations and suggests a standard nomenclature to describe them synthetically [31].

First of all, all variations should be described at the most basic level, the **DNA level**. RNA and protein level information may be provided in addition. A **letter prefix** identifies the type of sequence a variation affects:

- "**g.**", genomic sequence
- "**c.**", coding DNA sequence
- "**n.**", non-coding DNA sequence
- "**r.**", RNA sequence
- "**p.**", protein sequence

DNA recommendations	
Mutation	**Format**
Point mutation	g.123A>G, in a genomic sequence, at position 123, A is changed with G
Deletion	g.123_127del, in a genomic sequence, the nucleotides at positions from 123 to 127, have are missing
Insertion	g.123_124insAGC, in a genomic sequence, a nucleotides' sequence made by A, C and C, have been inserted between the nucleotides at positions 123 and 124
In-Del	g.123_127delinsAG, in a genomic sequence, nucleotides a positions from 123 to 127 have been deleted and the sequence AG has been inserted at that position
Duplication	g.123_345dup, in a genomic sequence, nucleotides at positions from 123 to 345 have been duplicated
Inversion	g.123_345inv, in a genomic sequence, the nucleotides at positions from 123 to 345, have been inverted

Protein recommendations

Mutation	Format
Silent mutation	p.Arg54=, altered codon at position 54, codes for the same amino-acid
Missense mutation	p.Arg54Ser, altered codon at position 54, codes for another amino-acid
Nonsense mutation	p.Arg54Ter or p.Arg54*, altered codon at position 54, results in a stop codon
Frameshift	p.Arg54LysfsTer16 or p.Arg54Lysfs*16, altered codon at position 54, results in a new codon and the new termination site is at position 16

Table 2.1: HGVS recommendations.

2.3.2 Copy Number Alteration

The wording "copy number alteration" (CNA) refers to the alteration of the correct number of copies of each gene, which, in a **diploid** organism, like humans, is normally two. If in some point of the DNA some extra copies of a gene are transcribed and translated some extra proteins are produced too; conversely, if the number of copies of a gene is smaller than the normal, proteins may be synthesized in smaller amounts. This may affect the normal functioning of the organism - an abnormally higher amount of protein is known as **protein abundance**. Copy number alterations are divided into the following classes:

- deep loss, both copies loss
- loss, single copy loss
- gain, single copy gain
- amplification, multi-copy gain

2.4 Oncogenesis

In conclusion, when mutations affect proteins which control the cellular life-cycle, a cancer may arise.

A **cancer** is a malignant tumor, made of an anomalous mass of cells which don't respond to control mechanisms and start dividing and growing in an uncontrolled way, as long as the environment is still favorable in the growth.

Benign tumor cells remain attached to the original tissue. *Malignant* tumors instead bypass every control mechanism so they are able to grow rapidly and even leave the original tissue, affecting also other tissues, causing what is called a **metastasis**.

Chapter 3

LAS

A lot of different biological and medical aspects of tumors are analyzed every day in research laboratories. As a result a huge amount of data is produced, and they need to be collected, managed and integrated. This is the main motivation which led to the development of the **Laboratory Assistant Suite** (**LAS**), a platform that supports the biomedical researchers in data management and analysis operations.

The LAS platform has been developed at the *Candiolo Cancer Institute* (**IRCCS**) thanks to the collaboration between IT and biomedical researchers [32] .

3.1 Basic concepts

Within the LAS framework data are modeled in terms of entities and relationships among them. An entity represents an object which can be an abstract concept or physical object. The system defines the following entity types [33]:

- **Patient**. A patient is a person who has given his consent to collect tumor samples by signing an Informed Consent.
- **IC**. It represents the Informed Consent signed by a patient.
- **Study**. A study can be a clinical trial or a research study approved by one or more institutions. Each study defines also a set of rules for samples' collection and research experiments' execution.
- **Collection**.It is the collection of specimens with analogous features or a common source. For example, all of the tissues extracted from the same patient after a surgical intervention are part of the same collection. Another intervention on the same patient starts a new collection.
- **Aliquot**. The collected samples are divided into different aliquots according to their characteristics. Different aliquots can follow a different path. They can be (i) stored into containers , (ii) used to derive other aliquots (e.g. DNA, RNA), and (iii) implanted in immunocompromised animals (i.e., xenografts) or (iv) used to derive cell lines.
- **Biomouse**. When an aliquot is implanted into an animal, it generates a biomouse. The same animal can host different biomice: this happens when different aliquots are implanted in different sites of the same animal.
- **Cell Line**. Viable aliquots may be used to produce a cell line, to execute in-vitro experiments on tumor samples.

All collections and biological entities (i.e., Aliquot, Biomouse, Cell Line) have an identifier, the *GenealogyID*. It is a mnemonic key which is automatically generated by the platform and encodes all relevant information about the entity. In Figure 3.1 the structure of the GenealogyID is shown through an example. The reader can notice that the first half of the string (up to the instance ID field) encodes the information about the entity derivation; whereas the second half describes specific features of the current bioentity (e.g., aliquot type, implant site, etc.).

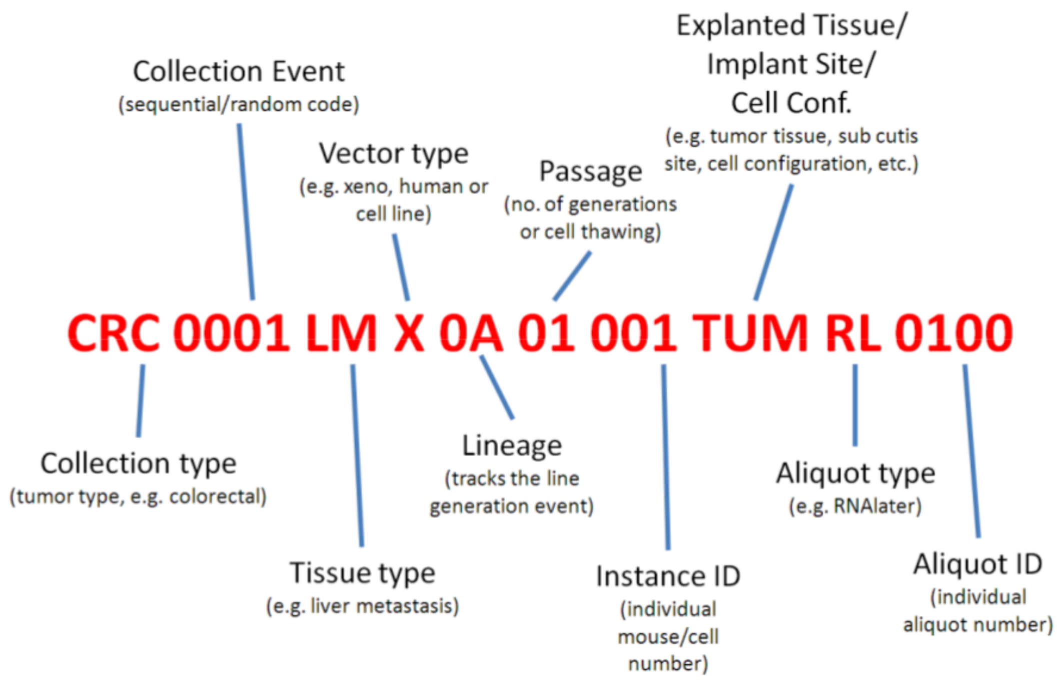

Collection Event
(sequential/random code)
Vector type
(e.g. xeno, human or cell line)
Passage
(no. of generations or cell thawing)
Explanted Tissue/ Implant Site/ Cell Conf.
(e.g. tumor tissue, sub cutis site, cell configuration, etc.)
CRC 0001 LM X 0A 01 001 TUM RL 0100
Collection type
(tumor type, e.g. colorectal)
Lineage
(tracks the line generation event)
Aliquot type
(e.g. RNAlater)
Tissue type
(e.g. liver metastasis)
Instance ID
(individual mouse/cell number)
Aliquot ID
(individual aliquot number)

Figure 3.1: GenealogyID structure

Figure 3.2 reports a graph representation of the relationships among LAS entities. A patient may generate one or more collection, by means of the same informed consent, or through different ICs related to different studies. Each collection can be composed of different aliquot types and several instances of the same aliquot type. If a viable specimen is implanted into animals, it generates one or more biomice. First-generation (or "first-passage") biomice are also labeled with a different lineage identifier (e.g. A, B). Each biomouse can generate aliquots that can in turn be implanted into other animals, so as to generate second-passage biomice. The implantation/explant process can be repeated several times according to the purpose of the research study. Viable aliquots can also be used to generate cell lines, which can be expanded and/or archived to produce other generations. All such relationships are stored in the graph database, while detailed information about each entity and the description of the procedures applied are collected in the relational database.

3.2 Architecture

The LAS platform is based on the Model-View-Controller pattern, which allows to decouple the user interface management from the data model and the control mechanisms.
Regarding data modeling, LAS uses different database solutions and technologies to fit different needs. In particular, it uses a relational database to collect entities and their properties and to store information about laboratory procedures; whereas, a non-relational document

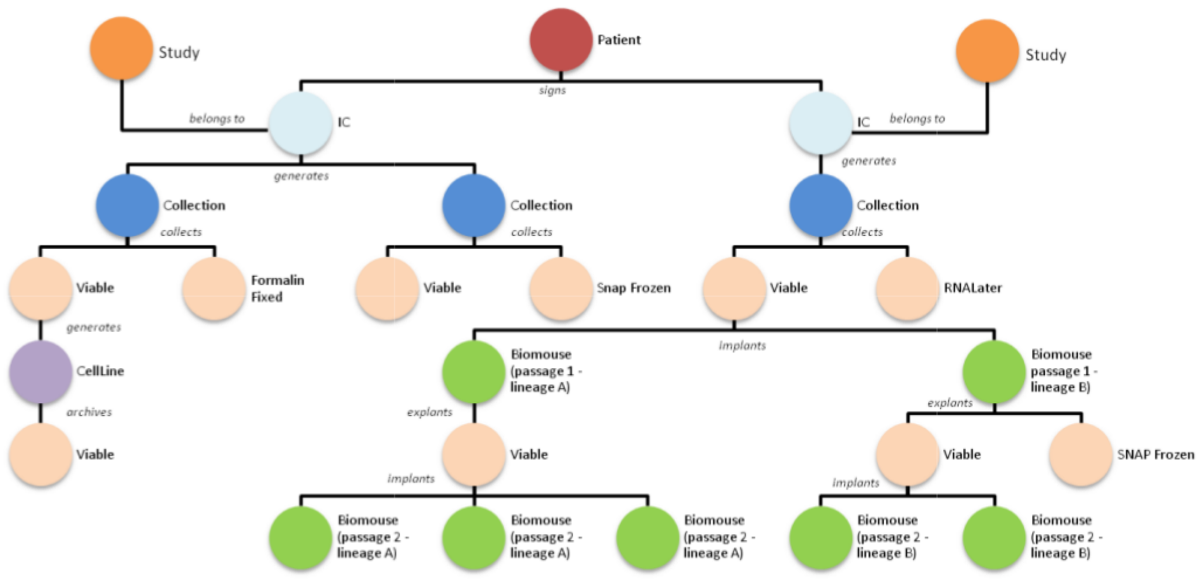


Figure 3.2: Graph representation of LAS entity relationships

database stores complex experimental data. Besides, a graph database is also used to (i) represent biological entities and their relationships, (ii) manage the LAS knowledge base and the ontologies it is base on and to (iii) enable data access control, by modeling data ownership as a social network.

Data are processed through different procedures in an increasing level of abstraction. These procedures may be classified into three main categories: (i) operative, (ii) integrative and (iii) analytical. The LAS architecture emulates this abstraction crescendo: it is organized into three tiers, each one dealing with one of three procedural layers.

The operative tier deals with raw data. Specifically, it is in charge of collecting, tracking and storing all data coming from laboratory experiments entered directly by the researchers. For this reason the GUI is thought to make data entry operations as easy ad possible and to support researchers in their operations.
The integration tier is responsible of integrating heterogeneous experimental data by means of complex queries. Biological entities are interlinked in a unique networks by means of special identifiers. Finally, the platform offers a graph visualization of integrated data.
The analysis tier (currently a prototype) offers the possibility to define workflows for the analysis of integrated data. The main idea is to provide the possibility to design complex analyses by means of a simple graphical representation.

3.3 Modules

The LAS platform has a modular architecture. Each module handles specific activities or data types and it's associated with experimental procedures. A brief description of the main functionalities of each module is described below.

Clinical The Clinical Manager Module is devoted to the management of patient clinical information, collected during trials and follow-ups. This module collects both context information (i.e. personal data, medical center of the trial, etc.) and relevant clinical events through the LAS Case-Report-Form (CRF). All data are linked to the corresponding informed consent.

Biobank A biobank is a collection of biological materials used by researchers to study pathologies and find possible therapeutic applications. Such collections are commonly divided in tissue and genetic biobanks according to the kind of biological materials they store. The BioBanking Module addresses both issues.

The scope of this module spans a wide range of activities, including management of biological samples and associated pathological information, as well as support to a number of laboratory-related procedures.

In vivo experiments The LAS platform allows to monitor the *xenopatient* life cycle, from their acquisition by the research institute to their death. In particular, during the acquisition of the animals, several features (e.g. status, strain, age, and source) are tracked. To speed up the identification of the animal and the retrieval of related information, the system promotes the usage of barcode readers when mice are equipped with RFid tags. Furthermore, the platform manages the implants/explants of tumor tissue into/from the xenopatients. Moreover, since in vivo experiments are usually aimed at testing treatments, some interfaces allow to define the characteristics of treatments and track tumor growth. To perform these operations, the Biobank and Storage modules are involved in retrieving the tumor aliquots stored in the containers. Finally, scientists are supported along the decision process by means of ad-hoc GUIs allowing them to monitor all experimental features (e.g., tumor growth) and to plan their activities.

In vitro experiments The LAS platform allows to monitor cell line life cycle. It requires to speficy the experimental conditions under which they were generated. The experimental conditions are defined by the protocols that describe the type of process and the set of culturing conditions applied. The platform also allows the management of the generation/thawing procedures of cell lines. Similarly to the xenopatients' management, the generation/thawing process can be handle with the support of the Biobank and Storage modules for retrieving aliquots of interest. During the cell line life cycle, scientists can perform a set of operations which are, again, supported by means og graphical user interfaces.

Molecular experiments Molecular experiments on biological samples allow to investigate the genetic events which caused the cancer onset. Different kinds of technologies may be adopted to execute those experiments, thus different kind of results are then produced. In an effort to closely track the translational research pipeline from the collection of samples to their analysis, Laboratory Assistant Suite provides support to tracking the most frequently used techniques in the Candiolo Cancer Institute, by dedicated modules.

Query This module is in charge of integrating all the information coming from experimental procedures and related to biological entities. The Query module, named Multi-Dimensional Data Manger (MDDM), can extract all information of interest from the databases in a uniform way by exploiting a graphical tool, named the query generator. Queries are generated by defining a workflow (block B in Figure 3.3) composed of one or more blocks, named query blocks, which are shown on the left hand side of the editor (block A) and categorized according to the module from which the data are drawn (e.g., the flask icon for biobank data, the mouse icon for xenopatient data). Each query block defines the object that will be retrieved (e.g., aliquot, xenopatient, container), its related information of interest and the filtering conditions. Set operators (union, intersection, difference) and special operators (group-count, extend, template blocks), listed in block C, can also be used in the workflow. Before retrieving the data from the corresponding modules, the workflow is analyzed to detect improperly defined operations (e.g., intersections among disjoint sets of objects) and define an optimal execution plan on the distributed databases. Once the workflow has been defined, a title and a description may be assigned to the query (block D) to reuse it in the future for different purposes. A query may also be designed, saved as a template and provided to unexperienced users, for use by means of wizards. Finally, the system allows enriching the result set with additional information, by means of predefined templates.

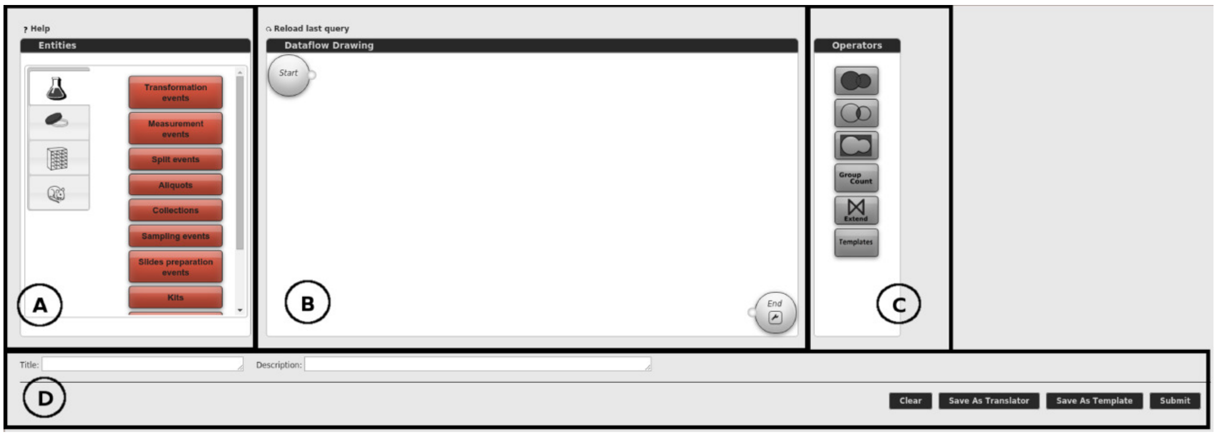


Figure 3.3: LAS query blocks.

Genomic Annotation Manager Genome annotation is the process of finding an designating locations of individual genes and other features on raw DNA sequences, called assemblies. Annotation gives meaning to a given sequence and makes it much easier for researchers to view and analyze its contents [34]. The Genomic Annotation Manager is meant to provide the functionality to annotate experimental and biological data collected by the Molecular Experiment Manager, in order to enrich them with high-level, qualitative information. Specifically, annotations are treated as labels attached to biological samples, highlighting their relevant features.
In order to achieve semantical coherence and adopt a standardized nomenclature, the relevant genomic and biological concepts have been taken from freely-accessible and public databases and ontologies (e.g. COSMIC [1], The Sequence Ontology, etc.). This information has been structured into a knowledge base, modeled as a graph and stored in a Neo4j graph database. Concepts are interlinked by both general purpose relationships (e.g. "part of", "is a", etc.) and domain specific relationships (e.g. "is transcribed from", etc.). New concepts and relationships may be added as needed.
So, an annotation is an abstract representation of a semantical link between a biological sample and a certain feature (e.g. genetic mutation). It is represented, within the graph, as an "annotation" node linking both the sample of interest and another node, named "reference" node. This kind of nodes actually specify the type of annotated feature for that sample and refer to one or more nodes representing the actual attribute value. Moreover, the annotation node is pointed to by an "analysis" node, corresponding to the laboratory analysis which originated the annotation process.

[1]The *Catalogue Of Somatic Mutations In Cancer* (COSMIC), is the the world's largest and most comprehensive resource for exploring the impact of somatic mutations in human cancer [35].

LAS Molecular Annotation Model The Genome Annotation Manager is based on descriptive semantic model for molecular data, relying on semantic concepts and relationships (ontologies): data generated by different sources are mapped to the semantic model and therefore normalized [36]. The main advantages of this approach are the following ones:

- associated unambiguous semantics allow coherent integration of datasets
- semantics provide proactive support for data enrichment and knowledge discovery (e.g. with semantic technologies or data mining techniques)

The LAS Molecular Annotation Model provides a comprehensive framework for recording molecularly-oriented annotations of biological samples in the context of the LAS platform. Moreover it integrates genomic knowledge from public authoritative sources and exploits a semantics-driven model with key concepts and relationships drawn from community-contributed ontologies. It uses a **graph** based representation and is structured into multiple, interconnected layers.

Genomic knowledge base It collects sequence data and genome annotation data imported from the *Gencode* project - a National Human Genome Research Institute (NHGRI) project aimed at identifying all functional elements in human genome sequence; it also stores all relevant concepts and relationships describing genomic domain taken from *The Sequence Ontology* - a collaborative ontology project for the definition of sequence features used in biological sequence annotation.

Genomic alteration knowledge base It defines a model which describes different classes of genomic variability or aberration. It includes the most representative and meaningful occurrences for each class, drawn from some public databases (i.e. *dbSNP* for SNP's and other short variations, *COSMIC* for sequence alterations and *Cancer Gene Census* for copy number variations). New variations may be added.

Experimental/technological knowledge base It maps genomic experiments performed with different technologies to their corresponding scrutinized genomic region(s). Moreover it identifies the universe of all genomic locations analyzed (and of all possible genomic alterations detectable) by an experiment, given its settings. Finally it allows the integration of experimental datasets obtained with different settings.

Experimental data A common repository stores raw experimental data, coming from laboratory instruments, and all intermediate processed data derived from them. Those data are managed through a non-relational document database (MongoDB); then some pointers link them to the graph nodes, which are then linked to each other in processing order. The last node in the chain points to the biological sample nodes. Moreover whenever an annotation is produced, as a result of an analysis, it is stored into the graph keeping a reference to its generating data.

Annotations Each annotation stored in the graph relates to one or more samples and to one specific genomic feature. They can be built and exported from third-party visualization tools. Finally, disambiguation of missing information (e.g. missing information vs wild type) is guaranteed.

Chapter 4

cBioPortal

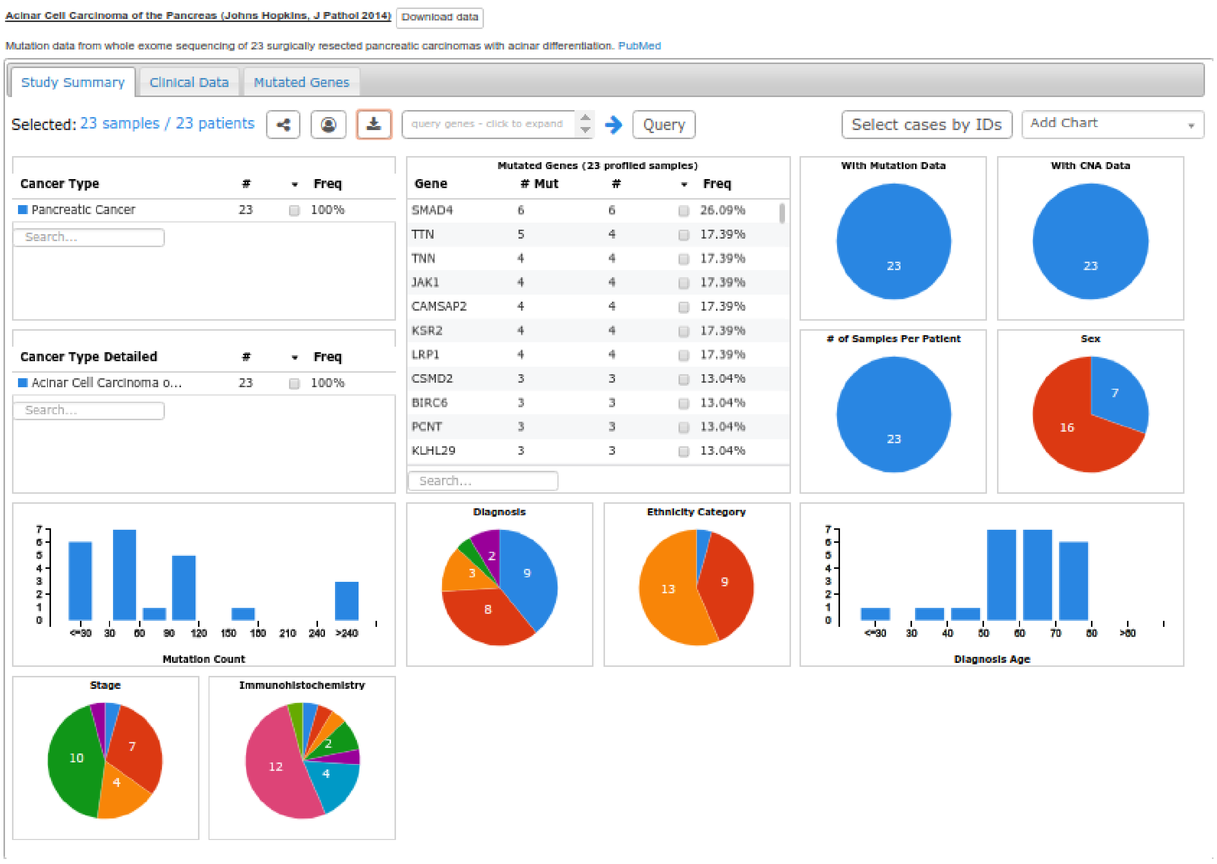
With the rapidly declining cost of sequencing technologies, and major international efforts, the field of cancer genomics continues to advance rapidly. Unfortunately data produced by research projects are not easily or directly available to the cancer research community. The **cBioPortal for Cancer Genomics**, developed at *Memorial Sloan-Kettering Cancer Center* (MSKCC), was specifically designed to address the data integration issues and to make raw data more easily and directly available to the research community.[37] There exists a public instance of the portal, which can be accessed through the Web; but there's also the possibility to install and deploy a local instance of the application, by downloading it from a GitHub repository or by means of a Docker container.
The cBioPortal is an open-access, open-source resource for interactive exploration of multidimensional cancer genomics data sets. The cBioPortal facilitates the access to complex genomic data and provides the possibility to transform large and complex data sets into clinical applications and biological insights [38].

4.1 Data structure

The public instance of cBioPortal contains *The Cancer Genome Atlas* (TCGA) datasets and other datasets from literature which can be browsed through a specific tab, named *Data Sets*.

The portal currently stores DNA copy-number data, mRNA and microRNA expression data, non-synonymous mutations (i.e. non-silent mutations), protein-level and phosphoprotein level data, DNA methylation data, and limited de-identified clinical data such as overall survival and disease-free survival intervals. Each data type is stored at gene level. The data are then organized as a function of patient and gene, and the portal's fundamental abstraction is the concept of altered genes; specifically, a gene is classified as altered in a specific patient if it is mutated, homozygously deleted, amplified, or its relative mRNA expression is less than or greater than a user-defined threshold.

The datasets provide the data needed to answer users' queries and can also be analyzed individually. The cBioPortal, in fact, provides access to summary information about each cancer study. The data available include various clinical details about the patients (survival and age at diagnosis), details about the tumor (histology, stage, grade), and summaries of the genomic data (number of nonsynonymous mutations and fraction of genome altered), details about the recurrently mutated genes, and details about recurrent CNAs. The clinical data are presented both graphically and in table format (Figure 4.1). The mutated gene and CNA data are presented in tables. All tables have a search option. [39]


Acinar Cell Carcinoma of the Pancreas (Johns Hopkins, J Pathol 2014) Download data

Mutation data from whole exome sequencing of 23 surgically resected pancreatic carcinomas with acinar differentiation. PubMed

Study Summary | Clinical Data | Mutated Genes

Selected: 23 samples / 23 patients

Cancer Type	#	Freq
🟦 Pancreatic Cancer	23	100%

Cancer Type Detailed	#	Freq
🟦 Acinar Cell Carcinoma o...	23	100%

Mutated Genes (23 profiled samples)

Gene	# Mut	#	Freq
SMAD4	6	6	26.09%
TTN	5	4	17.39%
TNN	4	4	17.39%
JAK1	4	4	17.39%
CAMSAP2	4	4	17.39%
KSR2	4	4	17.39%
LRP1	4	4	17.39%
CSMD2	3	3	13.04%
BIRC6	3	3	13.04%
PCNT	3	3	13.04%
KLHL29	3	3	13.04%

Figure 4.1: Cancer study summary view

4.2 Querying Individual Cancer Studies

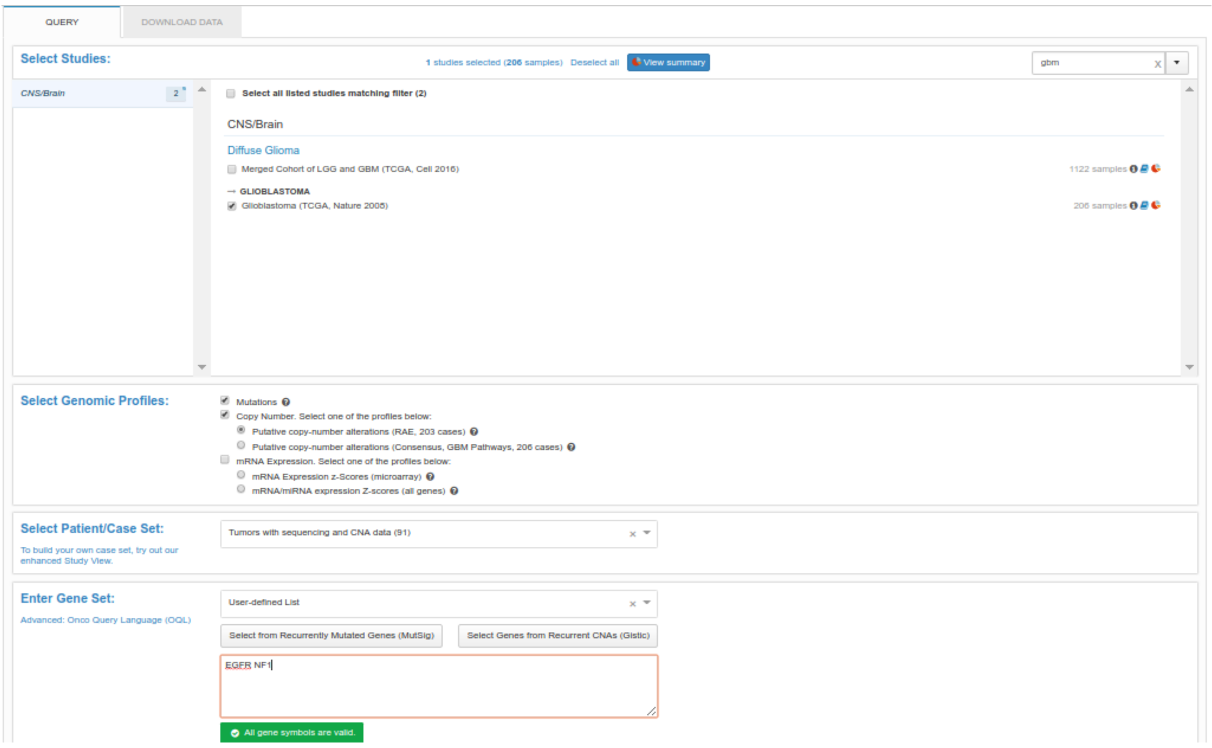
In a single-cancer query, users can explore and visualize genomic alterations in a selected set of genes, including the relationship between alterations in these genes across all selected samples and the relationship between different data types for the same gene. There are four steps to performing a query of a single-cancer study (Figure 4.2).

First, users need to select one of the cancer studies stored in the portal database. Next, they have to select the genomic profiles of interest; the portal specifies mutations and CNAs are by default and, when available, it also allows to select relative mRNA or miRNA expressionor of relative protein abundance[1]. After that the user has to define a case sets for analysis; the default option is set to match the selected genomic profiles: for example, cases with sequencing data will be selected if querying for mutations only. However, the user can change this selection by choosing from the drop-down list of case sets defined by the available data. Users may also input specific cases of interest by selecting “User-Defined Case List” or build a customized case set based on clinical attributes in the “Build Case Set” dialog. Finally, a gene set, made at list of one item, must bu specified; the user can manually enter HUGO gene symbols [2], Entrez Gene

[1]**Gene expression** is the process by which information from a gene is used in the synthesis of a functional gene product. These products are often proteins, but in non-protein coding genes such as transfer RNA (tRNA) or small nuclear RNA (snRNA) genes, the product is a functional RNA. It is measured in terms quantity of RNA (mRNA or miRNA expression) or of protein (protein abundance) produced, according to the type of analyzed genes (coding/non-coding).

[2]The *Human Genome Organization* (HUGO) is an organization involved in the Human Genome Project - a project about mapping the human genome - that sets the standards for human gene nomenclature.

identifiers [3], and gene aliases or select from predefined gene sets or pathways of interest.



QUERY | DOWNLOAD DATA

Select Studies: 1 studies selected (206 samples) Deselect all | View summary | gbm

CNS/Brain 2

Select all listed studies matching filter (2)

CNS/Brain

Diffuse Glioma

Merged Cohort of LGG and GBM (TCGA, Cell 2016) 1122 samples

→ GLIOBLASTOMA

[✓] Glioblastoma (TCGA, Nature 2008) 206 samples

Select Genomic Profiles:

[✓] Mutations

[✓] Copy Number. Select one of the profiles below:

[●] Putative copy-number alterations (RAE, 203 cases)

Putative copy-number alterations (Consensus, GBM Pathways, 206 cases)

mRNA Expression. Select one of the profiles below:

mRNA Expression z-Scores (microarray)

mRNA/miRNA expression Z-scores (all genes)

Select Patient/Case Set: Tumors with sequencing and CNA data (91)

To build your own case set, try out our enhanced Study View.

Enter Gene Set: User-defined List

Advanced: Onco Query Language (OQL)

Select from Recurrently Mutated Genes (MutSig) | Select Genes from Recurrent CNAs (Gistic)

EGFR NF1

All gene symbols are valid.

Figure 4.2: Single cancer study query

Onco Query Language The Onco Query Language (OQL) can be used to refine the query. Specifically, it allows to to select and define genetic alterations for all output on the cBioPortal Users can define filter data based on the copy number alterations, the mutations, the fusions, the mRNA Expression and on the protein/phosphoprotein level.
For example, assuming the he has selected mutations, copy number data and mRNA expression data in the step 2 of the query building, it is possible to select only amplified cases for a given gene, by specifying, in the gene input box:

```
CCNE1: AMP
```

To obtain amplified and gained cases, he has to write:

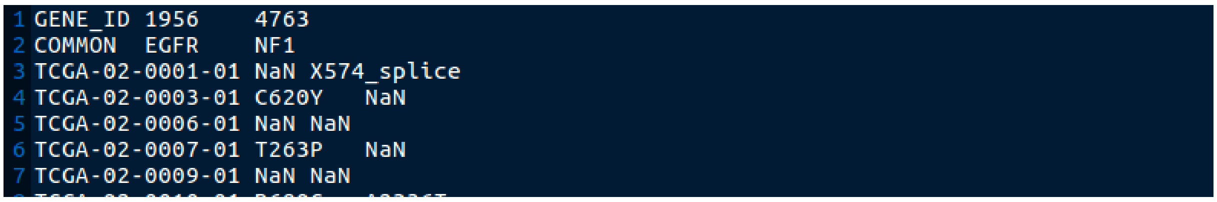
```
CCNE1:  CNA >= GAIN
```

For a complete list of the functionalities provided by the Onco Query Language, please refer to: `http://www.cbioportal.org/onco_query_lang_desc.jsp`.

4.2.1 Download Data

The *Download Data* tab allows to download all data in a per-sample alteration format. The portal produces a tab-delimited text file with all the retrieved data and, optionally, it can transpose them in a matrix (Figure 4.3).

[3] *Entrez Global Query Cross-Database Search System* is an integrated search database storing genomic data from different biobanks. It is coordinated by the *National Center for Biotechnology Information* (NCBI) in the US.



```
GENE_ID 1956    4763
COMMON  EGFR    NF1
TCGA-02-0001-01 NaN X574_splice
TCGA-02-0003-01 C620Y   NaN
TCGA-02-0006-01 NaN NaN
TCGA-02-0007-01 T263P   NaN
TCGA-02-0009-01 NaN NaN
```

Figure 4.3: Example of downloaded text file content.

4.2.2 Query

The *Query* tab allows the user to visualize the data returned by the portal in form of histograms and graphs which show trends, patterns and statistics. Each analysis and visualization is shown in a different tab.
The "OncoPrint" tab presents a concise and compact graphical summary of genomic alterations in multiple genes across a set of tumor samples. Rows represent genes and columns represent samples. Glyphs and color coding are used to summarize distinct genomic alterations, CNAs and change in the gene expression or protein abundance. Moreover, each study comes with a bunch of clinical attributes (e.g. fraction genome altered, diagnosis age, total mutations) which can be used to enrich the OncoPrint.Finally the users can customize the view (e.g. change color code, zoom-in/zoom-out) and download a printable version of the produced OncoPrint (see Section 6.2 for a detailed description of the OncoPrint functionalities). The "Mutual Exclusivity" tab provides a set of simple statistics computed to dentify patterns of mutual exclusivity or co-occurrence. The concept of mutual exclusivity can be exploited to identify previously unknown mechanisms that contribute to oncogenesis and cancer progression (12). In mutual exclusivity, events in genes associated with a specific cancer tend to be mutually exclusive across a set of tumors —that is, each tumor is likely to have only one of the genetic events. The opposite situation (co-occurrence) is when genetic alterations occur in multiple genes in the same cancer sample. Moreover, the "Correlation Plots" tab offers various plotting and visualization tools of discrete genetic events (CNAs or mutations) and continuous events. Besides detailed information about mutations, protein changes and survival statistics are accessible through homonym tabs. Additionally, the "Network" tab provides interactive analysis and visualization of networks that are altered in cancer. Finally the "Download", "IGV" and "Bookmark" tabs allow to download all genomic data and per-sample alteration events, to visualize copy number details by means of a Web start version of the IGV [4] and to to save or bookmark a specific query (the entire query can be stored in a URL) or share their results with collaborators by generating a short URL (using bit.ly).

[4]The *Integrative Genomics Viewer* (IGV) is a high-performance visualization tool for interactive exploration of large, integrated genomic datasets.

Chapter 5

Browser Extensions

A **browser extension** is a plug-in which extends browser functionalities without dealing with browser internals. So it is possible to write a small application which is capable of interacting with the browser environment and Web pages without modifying one single line of code of existing Web applications. They often have a little user interface and are activated by simply clicking on their icon on the browser toolbar. They are written using web technologies such as HTML, JavaScript and CSS and exploiting browser specific APIs.

Most of the effort spent for this thesis has been put on developing a Google Chrome extension which has been adapted and ported to Firefox too. For this reason, the working principles of Google Chrome extensions will be described first; after that a brief explanation of how Firefox extensions differ from the other ones will be provided too.

5.1 Google Chrome Extensions

5.1.1 The basics

Google Chrome is a famous freeware browser developed by Google and it provides its own API to develop extensions [40].

5.1.2 Architecture

Each extension has the following files:

- Manifest file
- (*Optional.*): HTML files
- (*Optional.*): JavaScript files
- (*Optional.*): Other files (e.g. images)

Reference to a certain file from another file can be done by means of a relative URL (e.g. `images/myimage.png`). Every file in an extension is also accessible by an absolute URL like this:

```
chrome-extension://<extensionID>/<pathToFile>
```

In that URL, the `<extensionID>` is a unique identifier that the extension system generates for each extension. The IDs for all loaded extensions can be found at the URL `chrome://extensions`. The `<pathToFile>` is the location of the file under the extension's top folder; and it's the same as the relative URL.

The manifest file The manifest file, called ***manifest.json***, is a JSON document providing information about the extension, such as the most important files and resources that the extension might use. Table 5.1 shows some of its common features.

Feature	Usage
`manifest_version`	*Required.* An integer specifying the version of the manifest file format.
`name`	*Required.* The extension name.
`version`	*Required.* The extension version.
`default_locale`	*Recommended.* The default local language.
`description`	*Recommended.* A plain text description.
`icons`	*Recommended.* A list made of one or more icons that represent the extension.
`browser_action` or `page_action`	*One of the two.* Allows to specify, through a nested variable, the icon that goes in the main Google Chrome toolbar, to the right of the address bar. Additionally, it `page_action` feature is used, a nested variable must be set to specify the page on which Web page the extension action can be applied.
`permissions`	*Optional.* Allows to declare the permissions the extension needs in order to be installed in a browser.
`background`	*Optional.* Allows to to register content scripts which must always be injected.
`key`	*Optional.* Allows to control the unique ID of an extension, app, or theme when it is loaded during development.
`web_accessible_resources`,	*Optional.* an array of strings specifying the paths of packaged resources that are expected to be usable in the context of a web page (images, scripts, stylesheets, etc.). These paths are relative to the package root.
`content_security_policy`	*Optional.* Allows to declare no-default content security policies.

Table 5.1: Manifest file features.

Here is an example manifest of an extension capable of changing the background color of the Web page where it is activated. It declares a browser action, the activeTab permission to see the URL of the current tab and the storage permission to remember the user's choice of the background color of the page.

```
{
  "manifest_version": 2,

  "name": "Getting started example",
  "description": "This extension allows the user to change the
     background color of the current page.",
  "version": "1.0",

  "browser_action": {
    "default_icon": "icon.png",
    "default_popup": "popup.html"
  },
    "permissions": [
      "activeTab",
      "storage"
    ],
    "background": {
      "scripts": ["popup.js"]
    }
}
```

Resources The manifest file points at two resource files when defining the browser action: icon.png and popup.png. Both resources must exist inside the extension package: icon.png will be displayed next to the Omnibox, waiting for user interaction (Figure 5.1), while popup.html will be rendered inside the popup window that's created in response to a user's click on the browser action. It is a normal HTML page, showing a select menu to let the user pick his favorite background color (Figure 5.2).

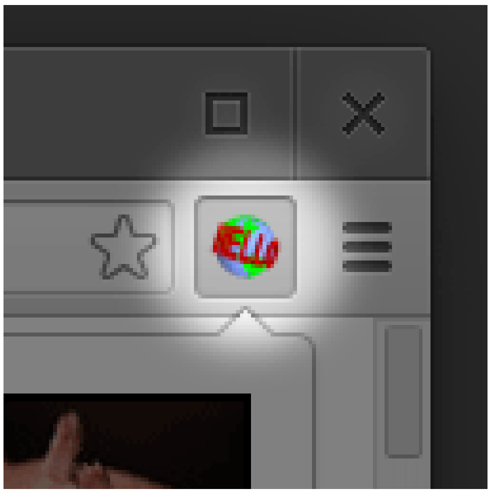


Figure 5.1: Extension icon.

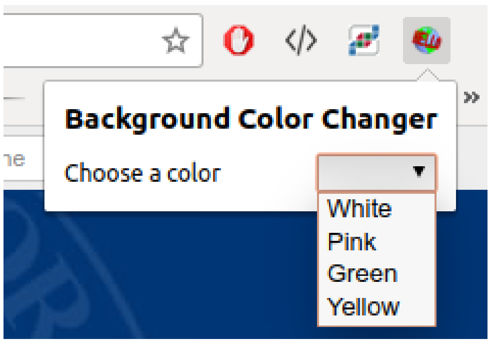


Figure 5.2: Extension popup.

The background script The background script is a JavaScript file, invisible to the user, which holds the main logic of the extension. It can be a *persistent background page* or an *event page*.

Event pages are background scripts which are loaded only when needed: when the event page is not doing actively nothing, it is unloaded, freeing memory and other system resources. For this reason, they should always be preferred to persistent background pages. The event page must be registered in the extension manifest as follows:

```
{
    "name": "My extension",
    ...
    "background": {
      "scripts": ["eventPage.js"],
      "persistent": false
    },
    ...
}
```

Once it has been loaded, the event page will stay running as long as it is active (for example, calling an extension API or issuing a network request). Additionally, the event page will not undload until all visible views are closed and all message ports are closed. Note that opening a view does not cause the event page to load, but only prevents it from closing once loaded.

Persistent background pages run in the extension process as long as the extension exists and is active. The persistent background page must be registered in the extension manifest as follows:

```
{
    "name": "My extension",
    ...
    "background": {
      "scripts": ["background.js"]
    },
    ...
}
```

In the example extension, the background script is named `popup.js`: it listens to the click events on `popup.html` and manages the background color change and storage by means of normal JavaScript routines.

Content Scripts A content script is a JavaScript page that runs in the context of the Web page where it has been injected into by the background script. Here is an example piece of code which a background page should execute to inject a content script when a button, named `downloadBtn`, is clicked:

```
document.addEventListener('DOMContentLoaded', function() {
  document.getElementById('downloadBtn').addEventListener('click'
    , function() {
    chrome.tabs.executeScript( { file: 'inject.js' } );
  });
});
```

Content scripts have complete access to the Web page DOM: they can microformat data, change the page background color, find unlinked URLs in web pages and convert them into hyperlinks or add new tags into the page DOM. On the other side they cannot use some of the extension APIs (e.g. `extension`, `runtime`) and do not have access to the scope of the page hosting them or of other content scripts.

If the content script code should always be injected, it should be registered in the extension manifest as follows:

```
{
  "name": "My extension",
  ...
  "content_scripts": [
    {
      "matches": ["http://www.google.com/*"],
        "css": ["mystyles.css"],
        "js": ["jquery.js", "myscript.js"]
    }
  ],
  ...
}
```

Each declared content script can have the multiples properties. The most important ones are: `matches` (required), which specifies which pages this content script will be injected into, `css` (optional), which lists of CSS files to be injected into matching pages and `js` (optional), which declares the JavaScript files to be injected into matching pages.

If the code should be injected only sometimes, the *permission* field must be used intead:

```
{
  "name": "My extension",
  ...
  "permissions": [
    "tabs", "http://www.google.com/*"
  ],
  ...
}
```

5.1.3 Using the chrome.*APIs

In addition to having access to all the APIs that web pages and apps can use, extensions can also use Chrome-only APIs (often called `chrome.* APIs`) that allow tight integration with the browser. For example, any extension or web app can use the standard `window.open()` method to open a URL. But if you want to specify which window that URL should be displayed in, your extension can use the Chrome-only `tabs.create` method instead.

Asynchronous vs. synchronous methods Most methods in the chrome.* APIs are asynchronous: they return immediately, without waiting for the operation to finish. If you need to know the outcome of that operation, then you pass a callback function into the method. That callback is executed later (potentially much later), sometime after the method returns. Here's an example of the signature for an asynchronous method:

```
chrome.tabs.create(object createProperties, function callback)
```

Other chrome.*APIs methods are synchronous. Synchronous methods never have a callback because they do not return until they have completed all their work. Often, synchronous methods have a return type. For instance, the `runtime.getURL` method is synchronous:

```
string chrome.runtime.getURL()
```

5.1.4 Communication between pages

The HTML pages within an extension often need to communicate. Because all of an extension pages execute in same process on the same thread, the pages can make direct function calls to each other. To find pages in the extension, use `chrome.extension` methods such as `getViews()` and `getBackgroundPage()`. Once a page has a reference to other pages within the extension, the first page can invoke functions on the other pages, and it can manipulate their DOMs.

5.1.5 Content Security Policy

In order to mitigate a large class of potential cross-site scripting issues, Chrome's extension system has incorporated the general concept of **Content Security Policy** (CSP). This introduces some fairly strict policies that will make extensions more secure by default, and provides the ability to create and enforce rules governing the types of content that can be loaded and executed the extensions.

In general, CSP works as a black/white mechanism for resources loaded or executed by the extension: they allow to define where resources can be loaded from, preventing browsers from loading data from any other locations These policies provide security over and above the host permissions the extension requests; they are an additional layer of protection, not a replacement.

An extension policy is defined by means the `manifest.json` file as follows:

```
{
    ...,
    "content_security_policy": "[POLICY STRING GOES HERE]"
    ...
}
```

Default policy restrictions

Packages that do not define a `manifest_version` have no default content security policy. Those that select `manifest_version 2`, have a default content security policy of:

```
script-src 'self'; object-src 'self'
```

This policy defines the extension package as the only valid source of JavaScript and of plug-ins. This means that it is not possible to evaluate inline JavaScript code. This restriction bans both inline `<script>` blocks and inline-event handlers (e.g. `<button onclick="...">`) within the extension views. Also the `eval()` JavaScript, within the extension scripts is disabled. Moreover, if an attacker injects JavaScript code within your extension, it will be not executed.

Relaxing the default policy

In order to relax the default policy limitations, it is possible to attach a certain number of values to the `script_src` feature. For instance, `'unsafe-inline'` allows use of inline source elements such as style attribute, onclick, or script tag bodies; `'unsafe-eval'` allows unsafe dynamic code evaluation such as JavaScript `eval()` and `domain.example.com` allows loading resources from the specified domain name.

The security policy applies to the background pages and event pages of the extension. Content scripts are generally not subject to the CSP of the extension. Since content scripts are not associated to HTML pages, the main impact of this is that they may use `eval()` even if the extension's CSP does not specify `unsafe-eval`. Moreover the CSP does not prevent content scripts from injecting `<script>` tags within the Web page DOM. And this is an important feature, because it allows to overcome the environment separation and to let the extension access to the Web page scope.

However, the behavior becomes more complicated both inside that DOM injected script and for any script that does not immediately execute upon injection (e.g. a click event handlers, registered inside the HTML tag): code not interpreted until the click event occurs is not considered part of the content script, so the CSP of the Web page (not of the extension) restricts its behavior. The correct way to implement the desired behavior in this case would be to add the event handlers as functions of the content script.

5.1.6 Message Passing

Since content scripts run in the context of a web page and not the extension, they often need some way of communicating with the rest of the extension. Communication between extensions and their content scripts works by using message passing. Either side can listen for messages sent from the other end, and respond on the same channel. A message can contain any valid JSON object (null, boolean, number, string, array, or object). There is a simple API which manages message passing.
In order to send a message to another part of your extension (and optionally get a response back), you should use `runtime.sendMessage` or `tabs.sendMessage`. This lets you send a one-time JSON-serializable message from a content script to extension , or vice versa, respectively . An optional callback parameter allows you handle the response from the other side, if there is one.

Sending a request from a content script looks like this:

```
chrome.runtime.sendMessage({greeting: "hello"},function(response)
     {
     console.log(response.farewell);
});
```

Sending a request from the extension to a content script looks very similar, except that you need to specify which tab to send it to. This example demonstrates sending a message to the content script in the selected tab.

```
chrome.tabs.query({active: true, currentWindow: true}, function(
    tabs) {
     chrome.tabs.sendMessage(tabs[0].id, {greeting: "hello"},
         function(response) {
        console.log(response.farewell);
   });
});
```

On the receiving end, you need to set up an `runtime.onMessage` event listener to handle the message. This looks the same from a content script or extension page.

```
chrome.runtime.onMessage.addListener(function(request, sender,
    sendResponse) {
     console.log(sender.tab ?
        "from␣a␣content␣script:" + sender.tab.url : "from␣the␣
           extension");
      if (request.greeting == "hello")
        sendResponse({farewell: "goodbye"});
  });
```

5.2 Firefox Extensions

Firefox, is the common name for **Mozilla Firefox**, a freeware and multiplatform browser developed by Mozilla Foundation.

Extensions for Firefox are built using WebExtensions APIs, a cross-browser system to develop extensions[41]. To large extent, the API is compatible with Google Chrome and Opera extension API. In this way extensions written for these browsers will run on Firefox with a few and small changes.

5.2.1 JavaScript APIs

Callbacks and the chrome.*namespace In Chrome, extensions access privileged JavaScript APIs using the `chrome` namespace:

```
  chrome.browserAction.setIcon({path: "path/to/icon.png"});
```

WebExtensions access the equivalent APIs using the `browser` namespace:

```
  browser.browserAction.setIcon({path: "path/to/icon.png"});
```

Many of the APIs are asynchronous, and in Chrome they use *callbacks* to return values and `runtime.lastError` to communicate errors:

```
  function logCookie(c) {
      if (chrome.runtime.lastError) {
          console.error(chrome.runtime.lastError);
      } else {
          console.log(c);
      }
   }
  chrome.cookies.set(
    {url: "https://developer.mozilla.org/"},
        logCookie
  );
```

The equivalent WebExtensions APIs use `promises` instead:

```
  function logCookie(c) {
      console.log(c);
    }
  function logError(e) {
    console.error(e);
  }
  var setCookie = browser.cookies.set(
    {url: "https://developer.mozilla.org/"}
  );
  setCookie.then(logCookie, logError);
```

Anyhow, Firefox, in order to help developers, supports both `browser` namespace with `promises` and `chrome` namespace with `callbacks`. This means that what works in Chrome, can be ported to Firefox without any changes This is an exception to WebExtensions standards.

Partially supported APIs In reality, not all of the JavaScript and chrome.* APIs are supported by Firefox. One notable example is the `tabs` API: it allows to execute a content script or to inject a CSS stylesheet by passing its URL, respectively, to `tabs.executeScript()` or to `tabs.insertCSS()`. In Chrome, this URL is resolved relative to the extension's base URL, while in Firefox it is considered relative to the current page URL. To work cross-browser, it is sufficient to specify the path as an absolute path, starting from the extension's root and it works fine in both cases:

```
/path/to/script.js
```

The page "Browser support for JavaScript APIs"[1] includes tables for all the APIs that have any support in Firefox.

5.2.2 Miscellaneous incompatibilities

In addition to what has been explained, Firefox resolves URLs in injected CSS files relative to the CSS file itself, rather than to the page it's injected into. Moreover, while in Chrome the extension ID is fixed for a given extension, in Firefox it is a random UUID that changes for every instance of the browser. This randomness can prevent you from doing a few things, such as add your specific extension's URL to another domain's content security policy. Additionally, when working with an unpacked extension, Chrome allows for a "key" property to be added to the manifest to pin the extension ID across different machines. In Firefox, because of random UUIDs, this property is unsupported. Finally, Firefox content script requests happen in the context of extension, not of the content page has it happens for Chrome extensions.

[1] `https://developer.mozilla.org/en-US/Add-ons/WebExtensions/Browser_support_for_JavaScript_APIs`

Part II

Design and Implementation

Chapter 6

Architecture

The aim of this thesis project was to design and implement a framework to allow LAS users to take advantage of the cBioPortal tools and, besides, to enrich their knowledge base by means of a cluster analysis, resulting in a higher-level annotation process of samples. In other words, the framework should allow to:

1. format and inject a dataset from the LAS to the cBioPortal;
2. take advantage of the data processing performed by the cBioPortal to obtain new information (e.g. statistical computation, aggregation according to different criteria, etc.);
3. identify data of interest and export them from the cBioPortal, organized by samples;
4. aggregate samples into clusters, according to their genomic and clinical features;
5. import the high-level labels produced by the clustering process into the LAS system, in form of new annotations on samples.

The first stages of the pipeline (from the LAS to the cBioPortal) already existed: this project goal was to design and implement the software needed to manage the process of extracting interesting information from the cBioPortal and transforming them in high-level LAS annotions.

6.1 Architecture and data flow

The framework is made of three components (Figure 6.1): a cBioPortal local instance, the LAS platform and a browser extension, named cBioPortal Downloader. Additionally, since both the cBioPortal and the LAS system are Web-applications, a Web-browser is required. The browser can be chosen between Mozilla Firefox and Google Chrome, since there exists an implementation of the extension for both of them.
In order to start the dataflow, the user has to load the cBioPortal Downloader into the browser, to deploy a cBioPortal instance by means of a Docker container (visit `http://cbioportal-inodb.readthedocs.io/en/latest/Docker-Prerequisites.html` for a step by step guide) and to populate it with some LAS datasets.

6.1.1 cBioPortal data loading

cBioPortal database population requires to produce a set of files containing the data and the metadata, formatted to be compliant with the cBioPortal requirements [42]. The file set generation is done automatically, by a script capable of collecting all the data of interest from the LAS database and generating the needed files. A valid cBioPortal study can basically consist of a

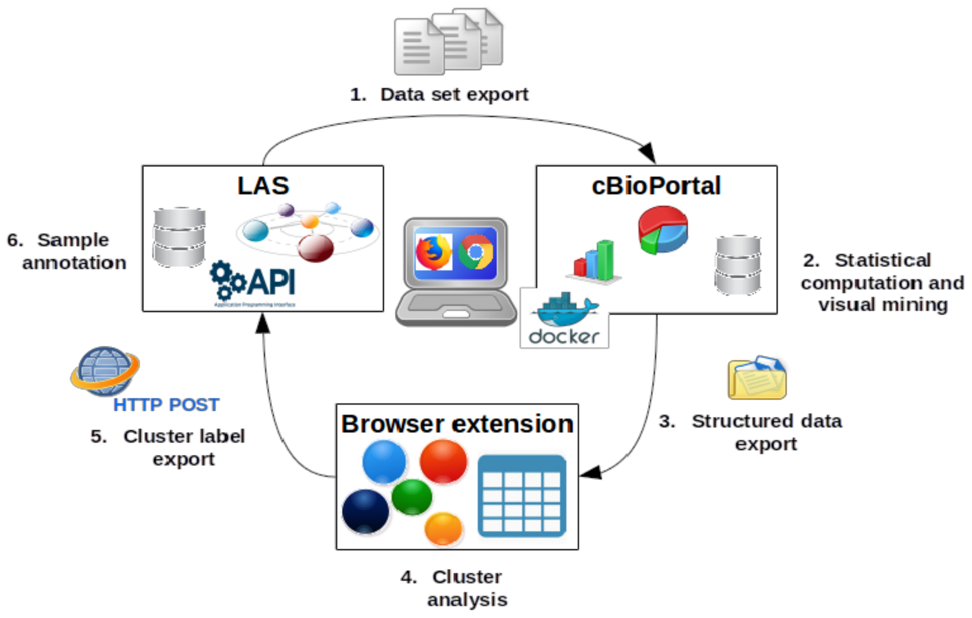


Figure 6.1: Framework architecture.

directory where all the data files are located. Each *data* file needs a *meta* file that refers to it and both files need to comply the format required for the specific data type. Here is an example of the files in such a directory:

```
dir
|-meta_study.txt
|-meta_cancer_type.txt -> cancer_type.txt
|-meta_clinical.txt -> data_clinical.txt
```

There are just a few rules to follow:

- `meta_study`, `meta_clinical` and respective data file are the only mandatory files;
- cancer type files can be mandatory if the study is referring to a cancer type that does not yet exist in the database;
- meta files can me named anything, as long as they start or end with the keyword 'meta'. E.g. `meta_test`, `meta.test`, `test.meta` are all fine; `metal_test` and `metastudy` are wrong;
- data files can be named anything and are referenced by the property `data_filename` in the meta file.

Meta study The meta study file contains metadata about the cancer study. It is made by the following fields:

Field	Description
`type_of_cancer`	The cancer type abbreviation, e.g. "brca". This should be the same cancer type as specified in the `meta_cancer_type.txt` file, if available.
`cancer_study_identifier`	A string used to uniquely identify this cancer study within the database, e.g. "brca_joneslab_2013".
`name`	The name of the cancer study, e.g. "Breast Cancer (Jones Lab 2013)".

`description`	A description of the cancer study, e.g. "Comprehensive profiling of 103 breast cancer samples. Generated by the Jones Lab 2013". This description may contain one or more URLs to relevant information.
`type_of_cancer`	The cancer type abbreviation, e.g. "brca". This should be the same cancer type as specified in the `meta_cancer_type.txt` file, if available.
`citation`	*Optional.*A relevant citation, e.g. "TCGA, Nature 2012".
`pmid`	*Optional.* A relevant pubmed id.
`short_name`	A short name used for display used on various web pages within the cBioPortal, e.g. "BRCA (Jones)".
`groups`	*Optional.* When using an authenticating cBioPortal, lists the user-groups that are allowed access to this study.
`add_global_case_list`	*Optional.* If set to 'true' allows to generate "All samples" case list automatically.

Table 6.1: cBioPortal meta study file fields.

An example `meta_study.txt` file would be:

```
type_of_cancer: brca
cancer_study_identifier: brca_joneslab_2013
name: Breast Cancer (Jones Lab 2013)
short_name: BRCA (Jones)
description: Comprehensive profiling of 103 breast cancer samples. Generated
    by the Jones Lab 2013.
add_global_case_list: true
```

Cancer type If the type of cancer, specified in the meta study file, does not yet exist in the portal database, a `meta_cancer_type.txt` file is also mandatory. The meta cancer type file is made of the following fields:

Field	**Description**
`genetic_alteration_type`	CANCER_TYPE
`datatype`	CANCER_TYPE
`data_filename`	Data file name.

Table 6.2: cBioPortal meta cancer type file fields.

An example meta cancer type file would be:

```
genetic_alteration_type: CANCER_TYPE
datatype: CANCER_TYPE
data_filename: cancer_type.txt
```

The cancer type data file comprises the following tab-separated columns:

Field	**Description**
`type_of_cancer`	The cancer type abbreviation, e.g. "brca".

`name`	The name of the cancer type, e.g. “Breast Invasive Carcinoma”.
`clinical_trial_keywords`	A comma separated list of keywords used to identify this study, e.g. “breast,breast invasive”.
`dedicated_color`	CSS color name of the color associated with this cancer study, chosen according to the awareness ribbons color schema - list of awareness ribbon colors and associated causes regarding health and disability, e.g. “HotPink”.
`parent_type_of_cancer`	The `type_of_cancer` field of the cancer type of which this is a subtype, e.g. “Breast”.

Table 6.3: cBioPortal cancer data file columns.

An example record would be:

```
brca<TAB>Breast Invasive Carcinoma<TAB>breast,breast invasive<TAB>HotPink<TAB>
    Breast
```

Clinical data The clinical data file is used to capture both clinical attributes and the mapping between patient and sample ids. The software supports multiple samples per patient. The *sample* file is required, whereas the *patient* file is optional. The two clinical metadata files (or just one metadata file there is no *patient* file) contains the following fields:

Field	**Description**
`cancer_study_identifier`	Same value specified in meta study file.
`genetic_alteration_type`	CLINICAL
`datatype`	PATIENT_ATTRIBUTES or SAMPLE_ATTRIBUTES
`data_filename`	Data file name.

Table 6.4: cBioPortal clinical meta data file fields.

An example of sample metadata file would be:

```
cancer_study_identifier: brca_tcga_pub
genetic_alteration_type: CLINICAL
datatype: SAMPLE_ATTRIBUTES
data_filename: data_clinical_samples.txt
```

An example of patient metadata file would be:

```
cancer_study_identifier: brca_tcga_pub
genetic_alteration_type: CLINICAL
datatype: PATIENT_ATTRIBUTES
data_filename: data_clinical_patients.txt
```

For both patients and samples, the clinical data file is a two dimensional matrix with multiple clinical attributes. When the attributes are defined in the *patient* file they are considered to be patient attributes; when they are defined in the *sample* file they are considered to be sample attributes. The first four rows of the clinical data file contain tab-delimited metadata about the clinical attributes. These rows have to start with a '#' symbol. Each of the four rows contains different type of information regarding each of the attributes that are defined in the fifth row:

- Row 1- Display Name: the display name for each clinical attribute;

- Row 2 - Description: long(er) description of each clinical attribute;
- Row 3 - Datatype: the datatype of each clinical attribute (can be one of: STRING, NUMBER, BOOLEAN);
- Row 4 - Priority: a number which indicates the importance of each attribute. In the future, higher priority attributes will appear in more prominent places than lower priority ones on relevant pages. A lower number indicates a higher priority

Here is an example of the first 4 rows with the respective metadata for the attributes defined in the 5th row:

```
#Patient Identifier<TAB>Overall Survival Status<TAB>Overall Survival (Months)<
    TAB>Disease Free Status<TAB>Disease Free (Months)<TAB>...
#Patient identifier<TAB>Overall survival status<TAB>Overall survival in months
     since diagnosis<TAB>Disease free status<TAB>Disease free in months since
    treatment<TAB>...
#STRING<TAB>STRING<TAB>NUMBER<TAB>STRING<TAB>NUMBER<TAB>...
#1<TAB>1<TAB>1<TAB>1<TAB>1<TAB>
PATIENT_ID<TAB>OS_STATUS<TAB>OS_MONTHS<TAB>DFS_STATUS<TAB>DFS_MONTHS<TAB>...
....
```

Following the metadata rows comes a tab delimited list of clinical attributes (column headers). The sixth row is the first row to contain actual data.

The file containing the patient and the sample clinical data are free form documents. The patient data file has only one required field, named `PATIENT_ID` and containing a unique patient ID. The sample data file, instead, requires at least two columns: one named `PATIENT_ID` and one named `SAMPLE_ID`, containing, respectively, a unique patient ID, allowing cBioPortal to map the given sample to the corresponding patient, and a unique sample ID. One patient can be associated to multiple samples. In addition to the mandatory columns, clinical file headers allow to define attributes such as the patient overall survival status, gender or age, and, for the samples, the tumor site, the metastatic site and other attributes. The only thing to do is to add a new column to the data file and, after having specified the required metadata, to fill it with the corresponding sample or patient values.

Other files may be added to the cancer study folder, but they are not mandatory. For further details, please, refer to the cBioPortal official documentation[1].

Once all files are ready, they can be validated by a command line tool provided by the cBioPortal and, finally, imported into its local database.

6.1.2 cBioPortal Downloader

Once a dataset has been loaded into the cBioPortal, it can be queried through its graphical interface. The cBioPortal collects, from the cancer study, the data which satisfy the query constraints and computes some statistics; finally, it presents the results to the user, who can use some graphical tools to perform a certain number of visual analyses on the returned sample data (see Section 6.2). Once he is satisfied, the user can trigger the cBioPortal Downloader by clicking on its icon (Figure 6.2): the extension captures the data of interest directly from the cBioPortal views (see Section 7.1), stores them locally (see Section 7.3) and performs a cluster analysis on the samples, grouping them according to their functional profile (i.e. their genomic and clinical attribute values) (see Section 7.2) and assigning an arbitrary label to each cluster.

[1] `http://cbioportal-inodb.readthedocs.io/en/latest/File-Formats.html`

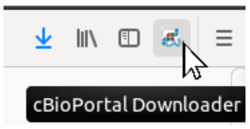


Figure 6.2: Browser extension icon.

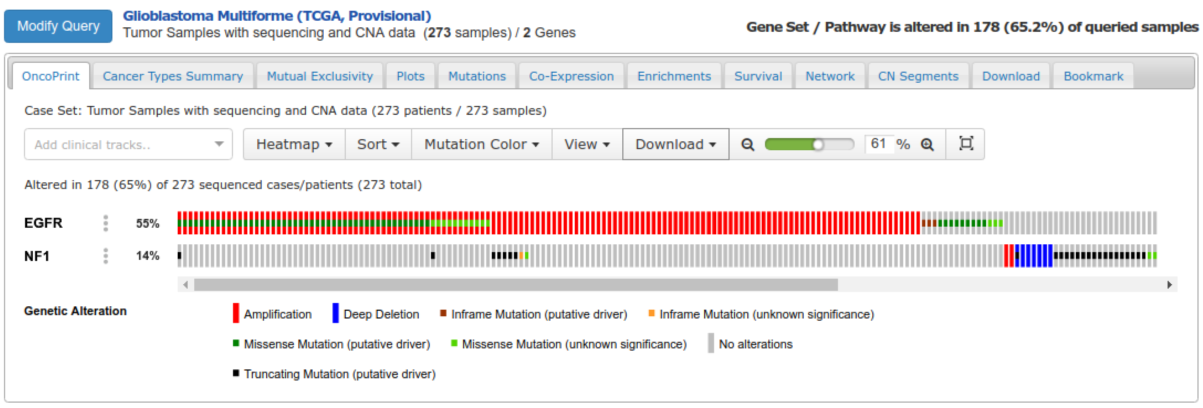
Then, it presents both the detailed data it has captured and the results of the cluster analysis into a new tab. Here the user can browse the sample data used by the cBioPortal to populate its graphical tools, analyze the cluster features and, eventually, customize the clusters; for instance, he can group the samples belonging to different clusters into a hybrid super-cluster, discard one or more cluster dimensions or flag some of the clusters as uninteresting (see Section 7.4). When the user is satisfied with the produced clusters, he can import them into the LAS platform, by clicking on a given button on the extension view.

6.1.3 LAS sample annotation

The cluster analysis results are used to enrich the LAS knowledge base; specifically, the cluster labels are used to produce new annotations to characterize the samples of the dataset which originated the workflow. The LAS system exposes a certain number of APIs which can be used both internally, to let the different components of the framework communicate between each other, and as an interface to the external world. In this case, the extension uses one of the LAS APIs to send, through a POST request, the samples of interest, grouped and labeled according to the cluster name and features they belong to. The LAS system unpacks the received data and enriches its knowledge base accordingly: specifically, it produces a new 'analysis' node, to state that a cluster analysis has taken place; after that, for each sample in the received data set, it generates a new 'annotation' pointing to a 'reference' node, specifying the cluster that sample belongs to (see Section 8).

6.2 cBioPortal functionalities

Section 4.2 explains the steps needed to build a cancer study query. Let us, now, analyze, in a detailed way, the functionalities the cBioPortal provides to its users.
First of all, the user has the possibility to select the "Download Data" tab which allows him to obtain the query results in text format, by pressing the "Download" button; notice that this option, allows to choose only one genomic profile at a time (e.g. mutations, copy-number alterations, etc.), differently from what happens when the query is submitted to produce the cBioPortal graphical views. The downloaded file is formatted as a tab-delimited matrix: the columns correspond to the samples, whose name is indicated in the first row of the file; the rows correspond to the genes, identified by their Entrez Gene ID and HUGO symbol in the first two columns of the matrix; finally, if the "Transpose data matrix" option is checked, the data matrix is transposed so that its columns correspond to the genes and its rows to the samples.
If the user selects the "Query" tab, the portal classifies each gene in each sample as altered or not altered, on the basis of the query criteria, and this classification is used for all analyses and visualizations in the portal, each of which is represented on a separate tab. Here the focus is put on the OncoPrint tab, since the framework that is being described takes advantage of its functionalities. Refer to the cBioPortal official documentation [37] to learn about the other tools it offers to the user.



Modify Query

Glioblastoma Multiforme (TCGA, Provisional)

Tumor Samples with sequencing and CNA data (273 samples) / 2 Genes

Gene Set / Pathway is altered in 178 (65.2%) of queried samples

OncoPrint | Cancer Types Summary | Mutual Exclusivity | Plots | Mutations | Co-Expression | Enrichments | Survival | Network | CN Segments | Download | Bookmark

Case Set: Tumor Samples with sequencing and CNA data (273 patients / 273 samples)

Add clinical tracks.. | Heatmap | Sort | Mutation Color | View | Download | 61 %

Altered in 178 (65%) of 273 sequenced cases/patients (273 total)

EGFR 55%

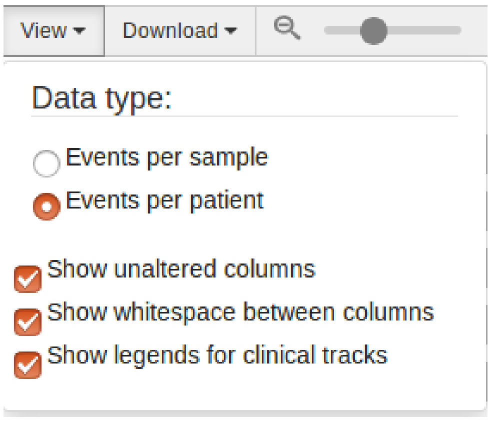
NF1 14%

Genetic Alteration

🟥 Amplification · 🟦 Deep Deletion · ■ Inframe Mutation (putative driver) · ■ Inframe Mutation (unknown significance) · 🟩 Missense Mutation (putative driver) · 🟩 Missense Mutation (unknown significance) · ▯ No alterations · ■ Truncating Mutation (putative driver)

Figure 6.3: OncoPrint view.

As mentioned in Chapter 4, the OncoPrint is, initially, made of a certain number of rows, each one representing one of the query genes; the columns, instead, represent samples (Figure 6.3). The initial view shows the sample data organized by patient: this means that all samples from a patient are merged into one column and all the statistics which are shown within the view are computed on a patient base. This behavior can be changed by selecting the option "Events per sample" from the dropdown menu, which appears when pressing the "View" bottom on the OncoPrint toolbar. In this way, each sample for each patient is in a separate column, and the statistics are computed accordingly. On the top of the gene bars, the OncoPrint declares the percentage of alterations found in the case set, computed as the ratio between the number of altered cases, over the cardinality of the case set. Additionally, an option, in the mentioned "View" dropdown menu, allows to discard unaltered cases (Figure 6.4). Each gene bar is pre-



View | Download

Data type:

Events per sample [✓ Events per patient]

[✓] Show unaltered columns

[✓] Show whitespace between columns

[✓] Show legends for clinical tracks

Figure 6.4: OncoPrint "View" dropdown menu.

ceded by a number, declaring the percentage of altered cases for that gene. Moreover, the color schema allows to get an idea of how the genomic profiles of interest are spread over the genes in the samples and to make comparisons among them (co-occurring or mutual exclusive genomic events can be spotted). Additional details are available by mousing over the gene column (Figure 6.5): they include detailed data about the genomic profiles of interest for that gene in the corresponding sample or patient and contain a link to the patient view page (Figure 6.6). In addition, the OncoPrint allows to enrich its visualization, by selecting, from a dropdown menu (Figure 6.7), one or more of the clinical attributes, among the ones declared into the clinical files (see Section 6.1.1) for the study of interest (Figure 6.8). To export the OncoPrint, the user has press the "Download" button on the toolbar and pick one format among PNG, SVG or PDF. Finally, if he wants to modify the query, he only has to choose "Modify Query" above the tabs

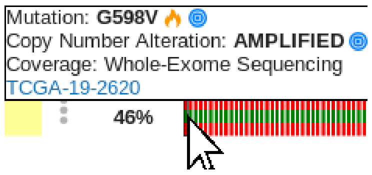
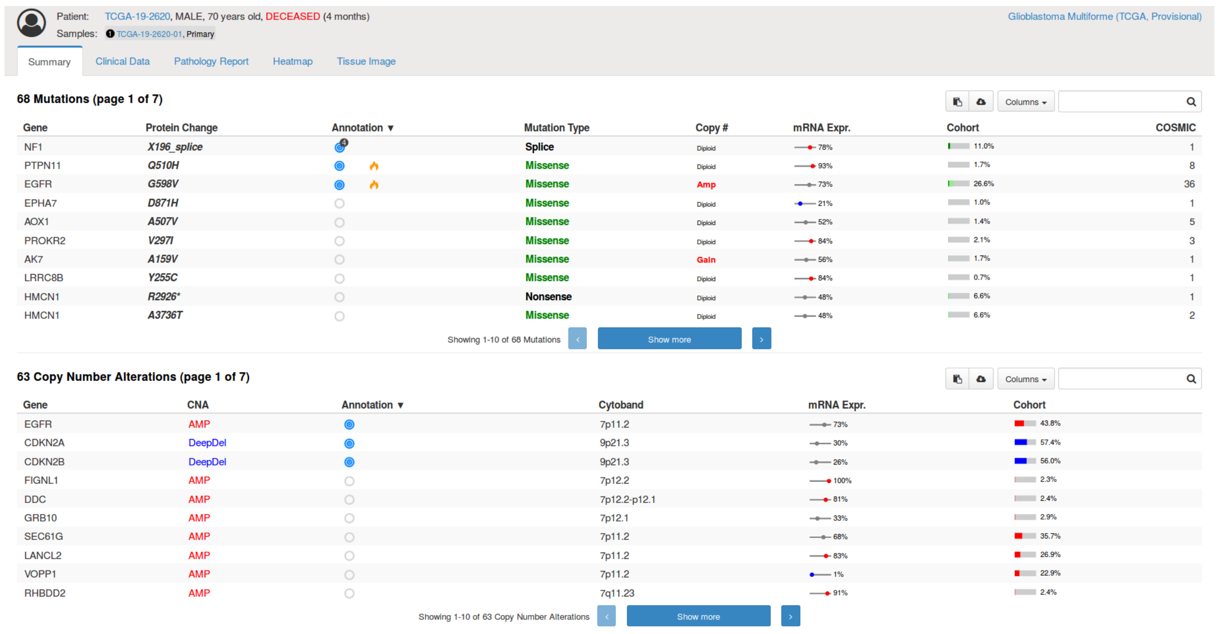

Mutation: **G598V** 🔥 ◎
Copy Number Alteration: **AMPLIFIED** ◎
Coverage: Whole-Exome Sequencing
TCGA-19-2620

Figure 6.5: OncoPrint patient details.


Patient: TCGA-19-2620, MALE, 70 years old, DECEASED (4 months)
Samples: ❶ TCGA-19-2620-01, Primary

Glioblastoma Multiforme (TCGA, Provisional)

68 Mutations (page 1 of 7)

Gene	Protein Change	Annotation ▼	Mutation Type	Copy #	mRNA Expr.	Cohort	COSMIC
NF1	*X196_splice*	◎	Splice	Diploid	78%	11.0%	1
PTPN11	*Q510H*	◎ 🔥	Missense	Diploid	93%	1.7%	8
EGFR	*G598V*	◎ 🔥	Missense	Amp	73%	26.6%	36
EPHA7	*D871H*	○	Missense	Diploid	21%	1.0%	1
AOX1	*A507V*	○	Missense	Diploid	52%	1.4%	5
PROKR2	*V297I*	○	Missense	Diploid	84%	2.1%	3
AK7	*A159V*	○	Missense	Gain	56%	1.7%	1
LRRC8B	*Y255C*	○	Missense	Diploid	84%	0.7%	1
HMCN1	*R2926**	○	Nonsense	Diploid	48%	6.6%	1
HMCN1	*A3736T*	○	Missense	Diploid	48%	6.6%	2

Showing 1-10 of 68 Mutations

63 Copy Number Alterations (page 1 of 7)

Gene	CNA	Annotation ▼	Cytoband	mRNA Expr.	Cohort
EGFR	AMP	◎	7p11.2	73%	43.8%
CDKN2A	DeepDel	◎	9p21.3	30%	57.4%
CDKN2B	DeepDel	◎	9p21.3	26%	56.0%
FIGNL1	AMP	○	7p12.2	100%	2.3%
DDC	AMP	○	7p12.2-p12.1	81%	2.4%
GRB10	AMP	○	7p12.1	33%	2.9%
SEC61G	AMP	○	7p11.2	68%	35.7%
LANCL2	AMP	○	7p11.2	83%	26.9%
VOPP1	AMP	○	7p11.2	1%	22.9%
RHBDD2	AMP	○	7q11.23	91%	2.4%

Showing 1-10 of 63 Copy Number Alterations

Figure 6.6: cBioPortal patient view.

and he is redirected to the query form, where he can provide new parameters according to his needs.

6.3 LAS annotations

The results of the analysis performed by means of the cBioPortal tools and, then, exported and aggregated by means of the cBioPortal Downloader, are integrated into the LAS semantic model: the new cluster analysis and annotation nodes not only introduce new concepts and relationships, but they also provide a new representation of the dataset which originated the workflow. This information can be obtained by simply querying the LAS graph database (Figure 6.9): when the user requests the data about a sample set, which has undergone the cBioPortal and cluster analyses, the sample nodes are returned with the associated cluster annotations and he can integrate this knowledge with all other relevant information which can be obtained though the returned dataset. This provides the researchers with a new layer of abstraction: genomic and clinical features, which were investigated individually, can now be considered as the components of functional profiles describing samples; and samples, which were previously represented as single entities, can be considered also as the members of given clusters, which allow to investigate genomic events co-occurrence/mutual-exclusivity and their correlation to clinical evidence (i.e. patient response to drug therapy).

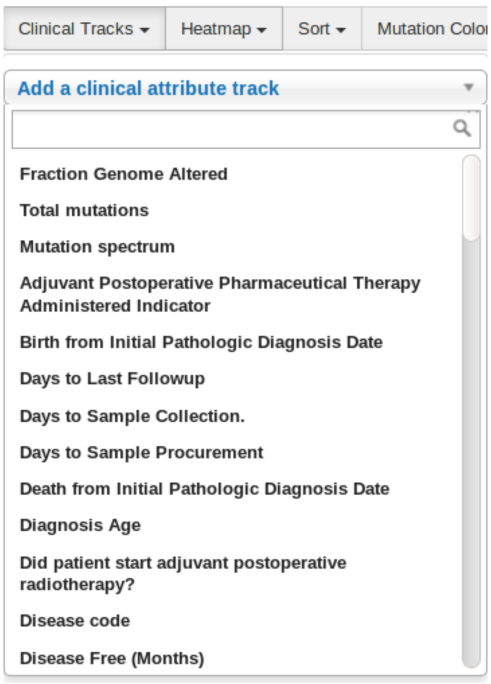

Clinical Tracks
Heatmap
Sort
Mutation Colo
Add a clinical attribute track
Fraction Genome Altered
Total mutations
Mutation spectrum
Adjuvant Postoperative Pharmaceutical Therapy Administered Indicator
Birth from Initial Pathologic Diagnosis Date
Days to Last Followup
Days to Sample Collection.
Days to Sample Procurement
Death from Initial Pathologic Diagnosis Date
Diagnosis Age
Did patient start adjuvant postoperative radiotherapy?
Disease code
Disease Free (Months)

Figure 6.7: OncoPrint "Clinical Tracks" dropdown menu.

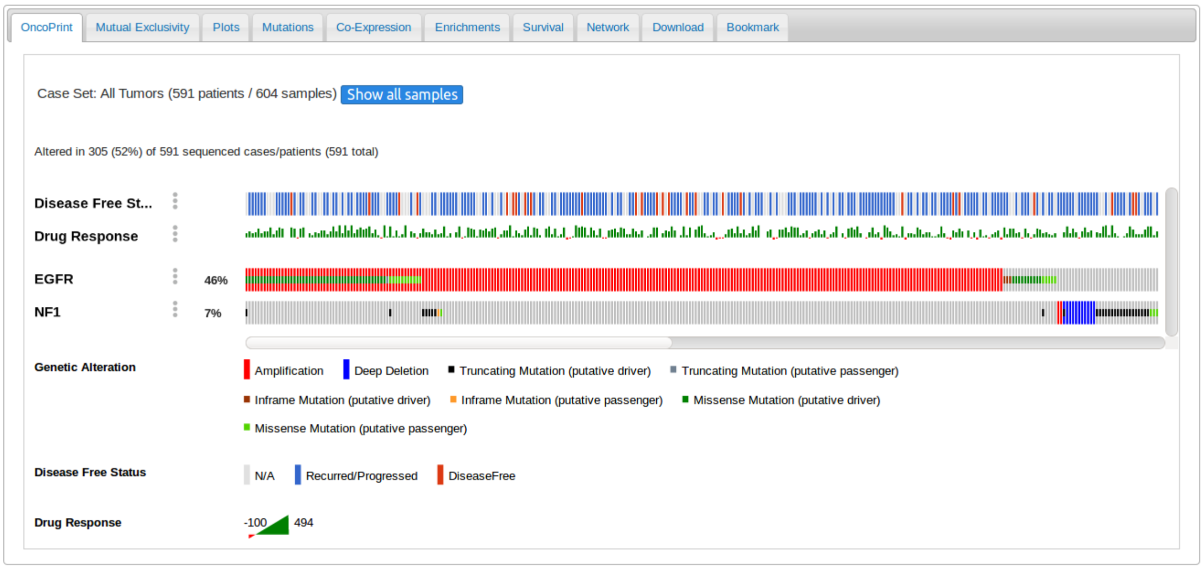

OncoPrint
Mutual Exclusivity
Plots
Mutations
Co-Expression
Enrichments
Survival
Network
Download
Bookmark
Case Set: All Tumors (591 patients / 604 samples)
Show all samples
Altered in 305 (52%) of 591 sequenced cases/patients (591 total)
Disease Free St...
Drug Response
EGFR
46%
NF1
7%
Genetic Alteration
Amplification
Deep Deletion
Truncating Mutation (putative driver)
Truncating Mutation (putative passenger)
Inframe Mutation (putative driver)
Inframe Mutation (putative passenger)
Missense Mutation (putative driver)
Missense Mutation (putative passenger)
Disease Free Status
N/A
Recurred/Progressed
DiseaseFree
Drug Response
-100
494

Figure 6.8: OncoPrint clinical track visualization.

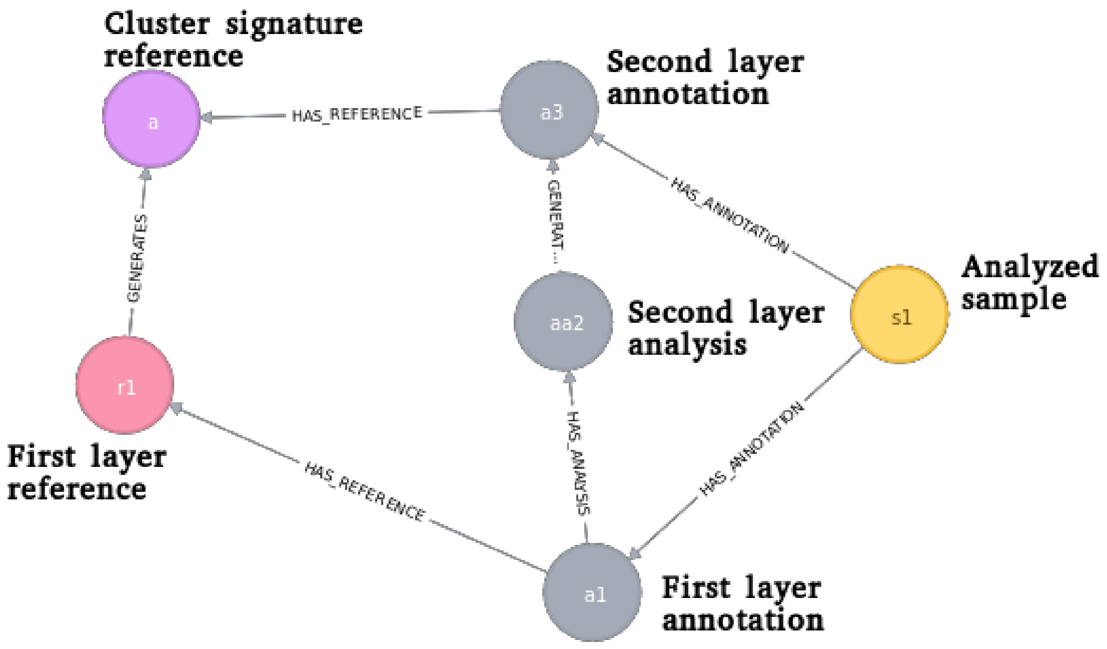


Figure 6.9: LAS cluster data representation.

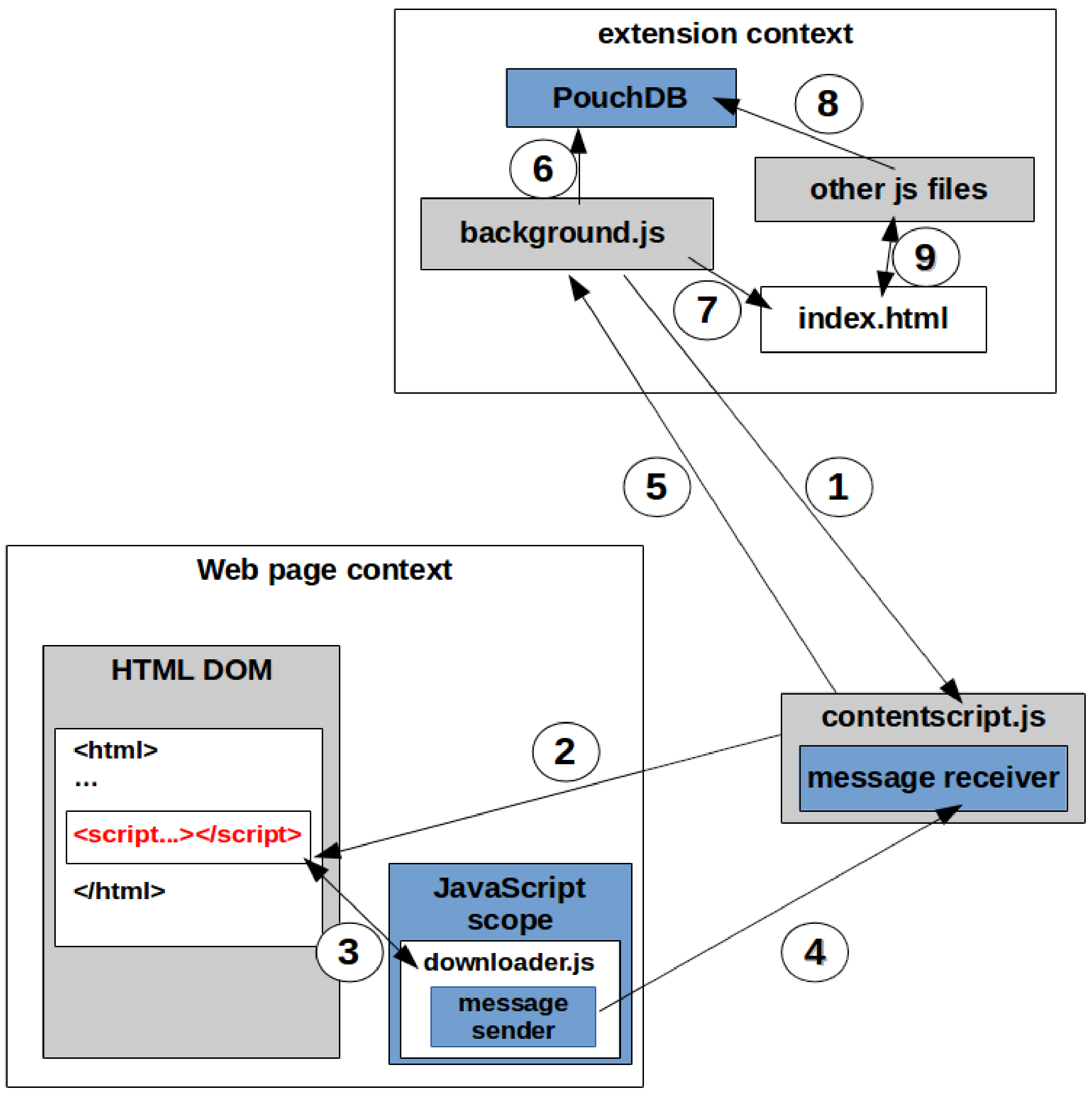
Chapter 7

cBioPortal Downloader

The core activities of the return dataflow, from the cBioPortal to the LAS platform, are handled by a browser extension, initially, developed for Google Chrome, and then ported to Mozilla Firefox. Browser extensions are plug-ins which allow to extended Web-application functionalities in a flexible and non-invasive way: since they are developed as separate components, they do not require the cares needed to hardcode the desired functionalities within a third-party software. In fact, it is possible to use different technologies and, even, programming languages; besides, they do not require to dig in someone else's code in order to integrate your own code; finally, if the third-party software is updated, you do not have to modify your own application (see Chapter 5). These are the reasons which led to the implementation of a browser extension, rather than a new cBioPortal module.

7.1 Data capture

The cBioPortal is a client-server application: the user interacts with the Web-client, to perform his queries; the Web-client, in its turn, queries a Web-server, which manages the database and responds by sending the requested data into a dynamic HTML page; the browser renders the received view and presents it to the user. Data, within the cBioPortal application, are managed by means of JavaScript variables and methods: the cBioPortal Downloader takes advantage of these structures to capture the results of the user analysis and export them. As Chapter 5 describes, browser extensions can interact with the hosting pages only by means of the content scripts, which are created and injected by the background script and have full access to the page DOM but not to its scope, which lives in an isolated environment with respect the the extension scope; so, the only possibility for an extension to execute some JavaScript code within the Web-page scope, is to dynamically add a new `<script>` tag within its DOM. This is what happens within this framework: when the user triggers the cBioPortal Downloader, its background script generates a content script (Figure 7.1, step 1) which injects a new `<script>` tag within the cBioPortal and loads an external script (Figure 7.1, steps 2 and 3). The external script can access the Web-portal scope and take advantage of its internal data structures: in this way, it can download the results of the user analysis (Figure 7.1, step 4) and make them available to the cBioPortal Downloader for its following computation. It is worth mentioning that, if the content scripts and the pages that host them wish to communicate, they must do it through the shared DOM, by means of an HTML5 functionality named cross-domain browser window messaging [43]: briefly the entity who needs to send information has to call `window.postMessage()`, which takes as parameter a JSON object containing the data to be sent; while the receiver page must register a handler for message events. This how the injected script sends the captured data to its parent content script (Figure 7.1, step 4). The content script, in its turn, takes advantage of the extension native massaging functionalities (see Section 5.1.6), to pass the received



1. The background script creates a content script.
2. The content script loads an external script within the page DOM.
3. The downloader script accesses the page scope and downloads data.
4. The downloader script sends data to the content script.
5. The content script sends data to the background script.
6. The backgroud script instantiates and populates a local PouchDB database.
7. The background script generates a new html page.
8. Some scripts associated to the html page retrieve and manipulate data from the PouchDB database
9. Populate the new tab panels.

Figure 7.1: cBioPortal data capture schema.

data to the background script (Figure 7.1, step 5), where they become available to the extension context. The background script, at this point, instantiates a PouchDB in-browser database and populates it with the cBioPortal analysis results (Figure 7.1, step 6); in conclusion, it creates a new HTML page (Figure 7.1, step 7), where some scripts operate to retrieve the sample data from the local database and, finally, present them to the user together with the results of a cluster analysis, performed on the same data, by the extension itself (Figure 7.1, steps 8 and 9).

7.2 Clustering

One fundamental functionality of the cBioPortal Downloader is the possibility to perform a cluster analysis on the sample data exported from the cBioPortal graphical tools. The clustering algorithm is based on a "conceptual" or "shared-property" approach, rather than a statistical computation, because data properties have a "conceptual" inherent meaning and aggregating them on the base of a statistic approach would have been meaningless. Specifically, the algorithm takes in consideration the sample categorical attributes as cluster dimensions and uses the *identity function* to measure sample similarity. The clustering process follows the following steps. First, all categorical attributes are identified by means of a statistical computation. The cardinality of the domain[1] of each sample attribute[2] is divided by the cardinality of the dataset: if the resulting ratio is a small number, and, hence, the attribute assumes a few distinct values all over the dataset, it is very likely that the attribute is a categorical one. The decision is made by comparing the resulting percentage with a reasonably low threshold, which, in this specific case has been set to a value of 1%.
Let us make an example. Genomic features, stating if a gene for a certain sample is mutated or wild type (i.e. non-mutated), have a domain made of only two items: "MUTATED" and "WILD TYPE". Let us suppose that a dataset contains 600 samples (typically, cancer studies collect data from some hundreds of patients):

$$\frac{domain\_cardinality}{dataset\_cardinality} \cdot 100 = \frac{2}{600} \cdot 100 = 0.33\% \tag{7.1}$$

As Equation 7.1 shows, genomic attributes, in this case, can assume a number of distinct values which is equal to the 0.33% of the dataset cardinality. So, since $0.33\% < 1\%$, the genomic features can be considered categorical attributes. This statement is assumed to be true for all genomic attributes and for any cancer study, since, according to our data model, they can always assume only two values ("MUTATED"/"WILD TYPE"). All other sample attributes are processed in the same way. Continuous attributes are, by now, neglected because dealing with them would have required to design and implement a functionality which should allow to divide their domain in ranges and assign them to categorical symbolic values; since the purpose of this project was to develop a simple and functional framework, this possibility has been, temporarily, neglected and could be one of the possible future developments (see Chapter 10).

Cluster dimensions are defined by the pattern of all categorical attributes selected by the user, as analysis dimensions, within the OncoPrint view. Clusters differ by the value that their dimensions assume: for each combination of these attribute values found in the dataset a cluster is initialized. The pattern of the cluster dimensions with the associated values can be named *cluster signature*. Let us consider a dataset made of 5 samples, characterized by three attributes:

[1]An attribute domain is the set of all possible distinct values that an attribute can assume

[2]Sample attributes correspond to the the tracks which the user used to build the OncoPrint. Typically, they correspond to the query genes and, eventually, to some clinical features which the user added to the OncoPrint. For a complete description of the data model which has been designed, please, refer to Section 7.3

"GENE1", which describes genomic alterations found for GENE1 in each sample and can assume the values ["MUT", "WT"]; "Drug Response", which describes the patient's response to the pharmacological treatment, measured in terms of relative growth of the tumor and which can assume the values ["CR", "PR", "SD", "PD"] (see Section 7.3); "Fraction Genome Altered", which tells the percentage of altered genome, for each sample, and can assume a value in the range [0,100]. Let us suppose that the dataset is made this way:

Sample ID	GENE1	Drug Response	Frac. Gen. Alt.
S1	MUT	PR	55%
S2	WT	SD	18%
S3	MUT	PR	78%
S4	MUT	PR	2%
S5	WT	SD	26%

Table 7.1: Dataset example.

This example dataset is characterized by two categorical attributes ("GENE1" and "Drug Response"), which can be used to define the cluster dimensions, and one continuous attribute ("Fraction Genome Altered"), which is discarded. Looking at the attribute values within Table 7.1, it is possible to identify two cluster signatures and, consequently, to initialize to clusters:

- patt1 = {"GENE1":"MUT", "Drug Response" : "PR"} [cluster C1]

- patt2 = {"GENE1":"WT", "Drug Response" : "SD"} [cluster C2]

Samples are associated to the cluster whose signature values are the same of the sample attribute ones. Figure 7.2 shows the output of such a cluster analysis performed on the dataset described by Table 7.1.

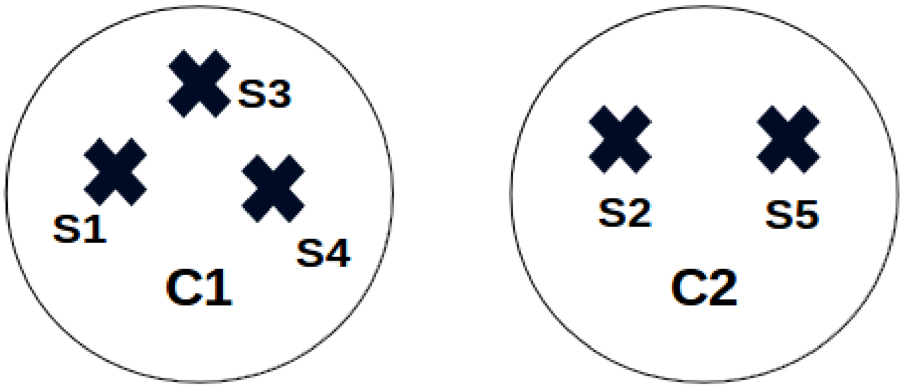


Figure 7.2: Cluster analysis result example.

7.3 Data model and storage

Data are modeled following the cBioPortal data representation: they are organized by sample or patient identifier, according to the data aggregation type selected by the user on the OncoPrint. Samples are characterized by a set of attributes: one attribute for each one of the tracks which compose the OncoPrint. Specifically, for each gene track a genomic attribute, identified by the gene HUGO symbol, is defined. It can assume two values: if at least one of the alterations listed in the queried genomic profiles have been found in a gene, on a certain sample, the corresponding sample attribute assumes the value "MUT" (abbreviation for "MUTATED"); on the contrary, if no one of the queried alterations have been found in that gene, in a certain sample, that sample attribute assumes the value "WT" (acronym for "WILD TYPE"). Also clinical tracks are mapped to sample attributes: they are identified by the track names and, for each sample, assume the value stored by the cBioPortal for that sample. The only exception is represented by the "Drug Response" data.

"Drug Response" clinical track keeps information about the response of the disease to the pharmacological therapy, in terms of relative tumor volume change. According to the *Response Evaluation Criteria In Solid Tumors* (RECIST) [44], the volume change must be measured in terms of percentage variation:

$$\Delta V_{\%} = \frac{V_f - V_i}{V_i} \cdot 100, \tag{7.2}$$

where V_i represents the initial tumor volume and V_f the final tumor volume.
According to the formula 7.2, the tumor volume percentage change can assume values in the range $[-100, +\infty[$. Depending on this percentage, the tumor is assigned to one of the following classes:

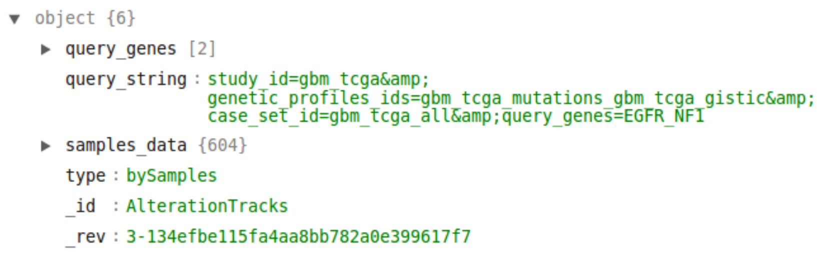
Category	**Tumor volume change interval**
Progressive disease (PD)	$]+30, +\infty[$
Stable disease (SD)	$]-100, -20]$
Partial response (PR)	$]-20, +30]$
Complete response (CR)	-100

Table 7.2: RECIST tumor change classification.

In conclusion, "Drug Response" clinical attribute, rather than being set with the tumor volume change, can assume one of the four values defined by the RECIST rules.

Finally, in order to make the downloaded data available to the extension context, they need to be stored. Rather than deploying a persistent database, here, data are kept in a **PouchDB** in-browser database [45]. PouchDB is a JavaScript implementation of CouchDB [46]. Its goal is to emulate the CouchDB API with near-perfect fidelity. It can be synchronized with a CouchDB database or it can just be exploited to generate an in-browser database. Like its elder brother, PouchDB is a NoSQL database, which allows to store unstructured documents rather than specifying a schema with rows, tables and all that jazz. The choice of deploying an in-browser database, without synchronization with CouchDB, has been motivated by the need of keeping this framework simple: a persistent database would have required to associate the stored query session results to the user who performed it and, consequently, to implement and manage users registration, log-ins and log-outs. For these reasons, by now, this functionality has been neglected. Anyhow, it may be one of the possible future developments (see Chapter 10).

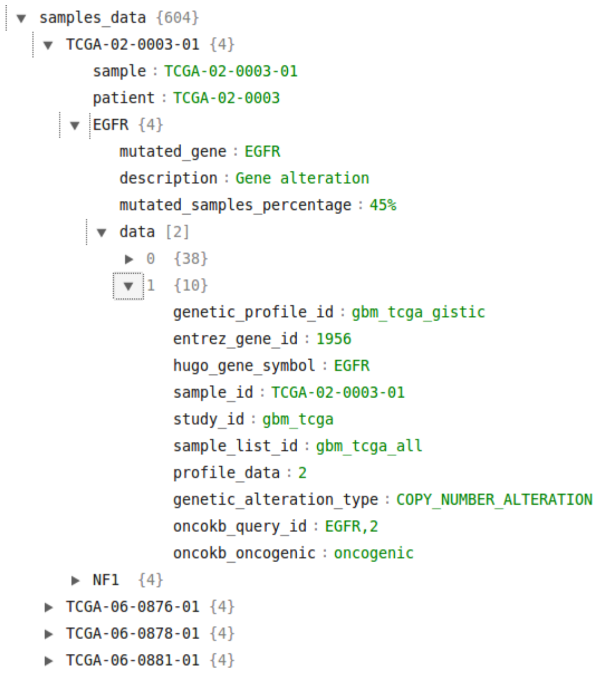
The sample data are organized in two hierarchical documents to be stored within the PouchDB database: one containing the genomic data and one containing the clinical data. Figure 7.3 shows the higher-level fields of the genomic data file. `samples_data` is the most relevant fea-



```
▼ object {6}
  ▶ query_genes [2]
    query_string : study_id=gbm_tcga&amp;
                   genetic_profiles_ids=gbm_tcga_mutations_gbm_tcga_gistic&amp;
                   case_set_id=gbm_tcga_all&amp;query_genes=EGFR_NF1
  ▶ samples_data {604}
    type : bySamples
    _id  : AlterationTracks
    _rev : 3-134efbe115fa4aa8bb782a0e399617f7
```

Figure 7.3: Genomic data file structure.

ture: it collects the genomic information about all samples, organized by sample identifier, externally, and gene symbol, internally. Each gene attribute, in its turn, contains a field, named `data`, storing the detailed information about the genomic alterations affecting that gene (Figure 7.4). (If the patient aggregation option is selected on the OncoPrint, this field is substituted by an analogous one, named `patients_data`, which collects patients' genomic information aggregated by patient identifier). The document contains, also, some meta-fields, containing in-

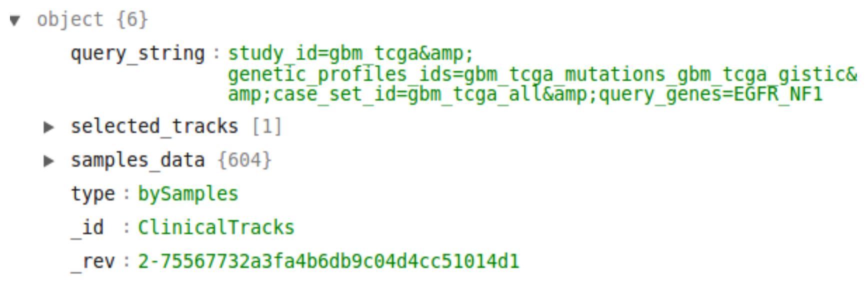


```
▼ samples_data {604}
  ▼ TCGA-02-0003-01 {4}
      sample : TCGA-02-0003-01
      patient : TCGA-02-0003
    ▼ EGFR {4}
        mutated_gene : EGFR
        description : Gene alteration
        mutated_samples_percentage : 45%
      ▼ data [2]
        ▶ 0  {38}
        ▼ 1  {10}
            genetic_profile_id : gbm_tcga_gistic
            entrez_gene_id : 1956
            hugo_gene_symbol : EGFR
            sample_id : TCGA-02-0003-01
            study_id : gbm_tcga
            sample_list_id : gbm_tcga_all
            profile_data : 2
            genetic_alteration_type : COPY_NUMBER_ALTERATION
            oncokb_query_id : EGFR,2
            oncokb_oncogenic : oncogenic
    ▶ NF1  {4}
  ▶ TCGA-06-0876-01 {4}
  ▶ TCGA-06-0878-01 {4}
  ▶ TCGA-06-0881-01 {4}
```

Figure 7.4: Genomic sample data.

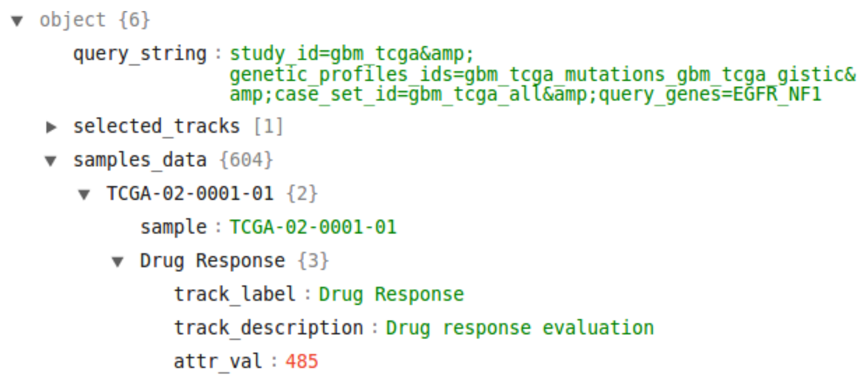
formation used by the extension to properly manage it: `query_genes` lists the genes investigated by the cBioPortal query; `query_string` holds a semantic string which concatenates all the query parameters and represents a signature of the current query session; `type` keeps track of the sample aggregation option the user selected (i.e. by samples or by patients); `_id` and `_rev`, finally, are fields which all PouchDB documents must contain.
The document storing the clinical data is analogous to the previous one (Figure 7.5), with only a couple of differences: the query gene list is replaced by a list of the investigated clinical tracks, named `selected_tracks`, and, clearly, each sample (or patient) object, under the `samples_data`

(or `samples_data`) field, stores the clinical data aggregated by track name (Figure 7.6).



```
▼ object {6}
    query_string : study_id=gbm_tcga&amp;
                   genetic_profiles_ids=gbm_tcga_mutations_gbm_tcga_gistic&
                   amp;case_set_id=gbm_tcga_all&amp;query_genes=EGFR_NF1
  ▶ selected_tracks [1]
  ▶ samples_data {604}
    type : bySamples
    _id  : ClinicalTracks
    _rev : 2-75567732a3fa4b6db9c04d4cc51014d1
```

Figure 7.5: Clinical data file structure.



```
▼ object {6}
    query_string : study_id=gbm_tcga&amp;
                   genetic_profiles_ids=gbm_tcga_mutations_gbm_tcga_gistic&
                   amp;case_set_id=gbm_tcga_all&amp;query_genes=EGFR_NF1
  ▶ selected_tracks [1]
  ▼ samples_data {604}
    ▼ TCGA-02-0001-01 {2}
        sample : TCGA-02-0001-01
      ▼ Drug Response {3}
          track_label : Drug Response
          track_description : Drug response evaluation
          attr_val : 485
```

Figure 7.6: Clinical sample data.

7.4 Data browsing

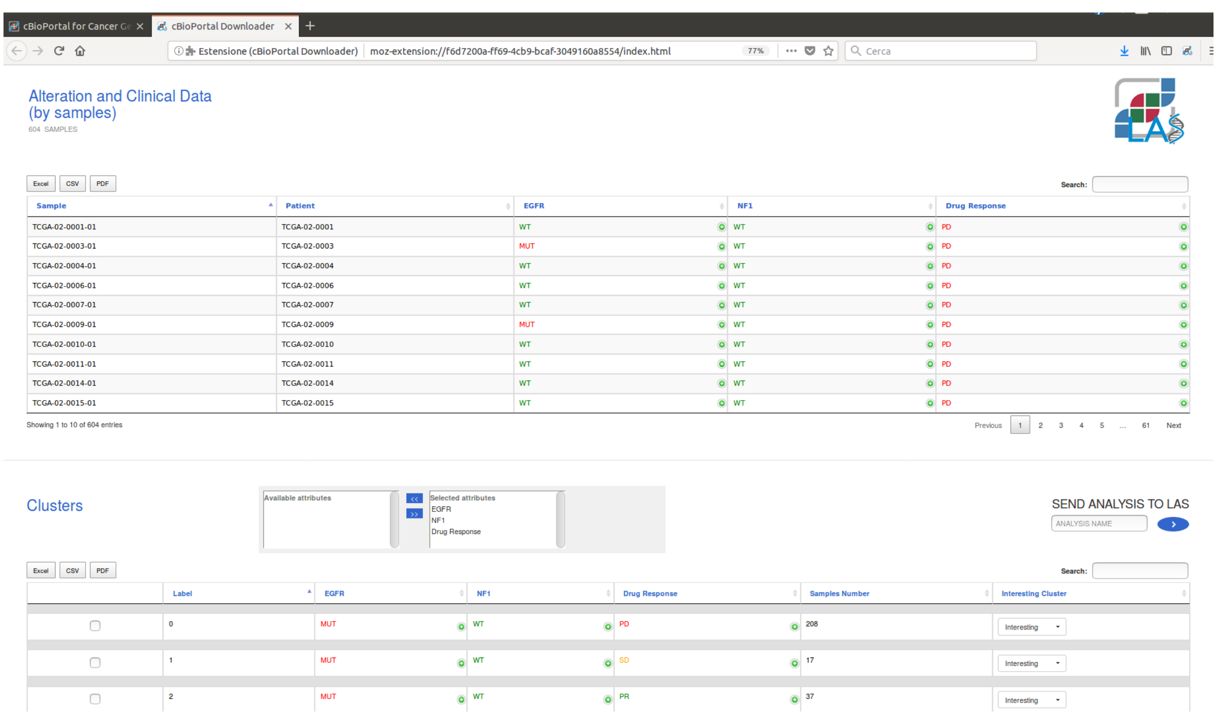
The cBioPortal Downloader presents data in a new tab, split into two panels, both organized in form of tables: the first one contains the genomic and clinical data exported from the cBioPortal, while the second one shows the results of the clustering process (Figure 7.7).

7.4.1 Sample data panel

The sample data panel shows the details of the data exported from the cBioPortal (Figure 7.8). Each one of its rows corresponds to one sample or one patient, depending on the visualization option selected by the user on the OncoPrint toolbar. The columns, instead, correspond to the sample attributes (see Section 7.3). Additionally, for some of the attributes additional details are available: genomic attributes, for example, are associated with the details of the single alterations found on each gene and sample. This information may be obtained by pressing the button appearing on the corresponding cell (Figure 7.9). Finally, there is also the possibility to download the sample data, as they appear in the table, in a text file, by pressing on one of the download buttons on the top on the table ("Excel", "CSV", "PDF").

7.4.2 Cluster panel

The second half of the page contains a table showing the details of the clusters produced by the clustering process (Figure 7.10). Rows correspond to the clusters, while columns correspond to their dimensions, plus a column for the cluster label and one for the cluster size. The labels, assigned by the application, are integer numbers starting from 0 and incremented by one for


cBioPortal for Cancer Ge
cBioPortal Downloader
Estensione (cBioPortal Downloader)
moz-extension://f6d7200a-ff69-4cb9-bcaf-3049160a8554/index.html
77%
Cerca

Alteration and Clinical Data
(by samples)
604 SAMPLES

Sample	Patient	EGFR	NF1	Drug Response
TCGA-02-0001-01	TCGA-02-0001	WT	WT	PD
TCGA-02-0003-01	TCGA-02-0003	MUT	WT	PD
TCGA-02-0004-01	TCGA-02-0004	WT	WT	PD
TCGA-02-0006-01	TCGA-02-0006	WT	WT	PD
TCGA-02-0007-01	TCGA-02-0007	WT	WT	PD
TCGA-02-0009-01	TCGA-02-0009	MUT	WT	PD
TCGA-02-0010-01	TCGA-02-0010	WT	WT	PD
TCGA-02-0011-01	TCGA-02-0011	WT	WT	PD
TCGA-02-0014-01	TCGA-02-0014	WT	WT	PD
TCGA-02-0015-01	TCGA-02-0015	WT	WT	PD

Showing 1 to 10 of 604 entries

Clusters

Available attributes

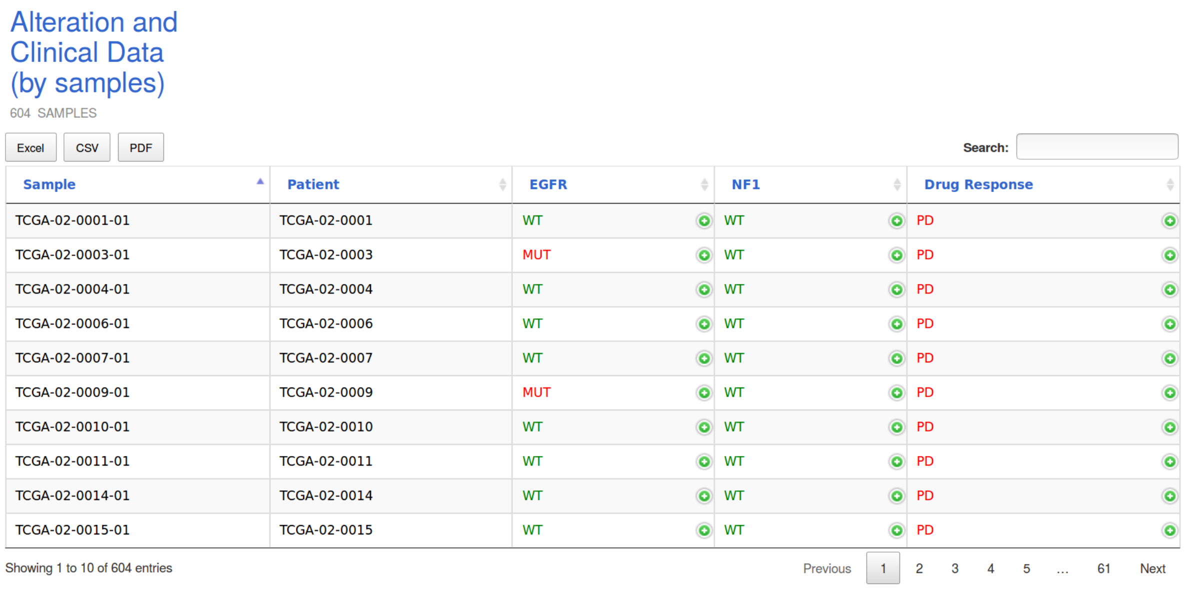
Selected attributes
EGFR
NF1
Drug Response

SEND ANALYSIS TO LAS

	Label	EGFR	NF1	Drug Response	Samples Number	Interesting Cluster
☐	0	MUT	WT	PD	208	Interesting
☐	1	MUT	WT	SD	17	Interesting
☐	2	MUT	WT	PR	37	Interesting

Figure 7.7: cBioPortal Downloader page.

every new cluster. Similarly to the sample data table, the cluster table provides the possibility to obtain additional details by pressing on a button on the table cells; in this way, the complete list of the samples belonging to the cluster corresponding to the cell row is produced: by selecting one of the sample identifiers, it is possible to obtain the value assumed by the attribute corresponding to the cell column, in that sample (Figure 7.11). It is possible to customize clusters. First of all, a checkbox in the first cell of each row, allows, when selected, to merge two or more clusters and build a "hybrid super-cluster" (Figure 7.12). The new super-cluster requires a label: the system proposes a default one which can be changed through an apposite textbox. Anyhow, even when merged, clusters with different attributes are kept in separate rows, which are only re-organized so that clusters with the same label are shown one close to the other. This allows to un-merge them by simply checking the checkbox, again, and providing a new and different label. Furthermore, this checkbox also offers the possibility to simply change one cluster label. A double select list at the top of the table allows to discard or include one or more cluster attributes and change their dimensionality and representation. Additionally, a dropdown menu in the last cell of each row allows to mark a cluster as "un-interesting" and, in case, to make it "interesting" again; a super-cluster collects all not interesting classes of sample sets (Figure 7.13). If the user wants to bring the clusters back to their initial representation, he only needs to click on a button to undo all changes. Moreover, also cluster data can be downloaded in text format by pressing one of the buttons on the top of the table. Finally, when the cluster analysis is over, the user can export the results to the LAS platform, by providing an analysis name and pressing on the given button on the top of the table (Figure 7.14): the extension builds data structure containing only the clusters which have been flagged as "interesting" and where clusters with the same label are actually merged together; then, it submits an HTTP POST, containing the just build object, to the LAS API and, finally, stops working.


Alteration and Clinical Data (by samples)

604 SAMPLES

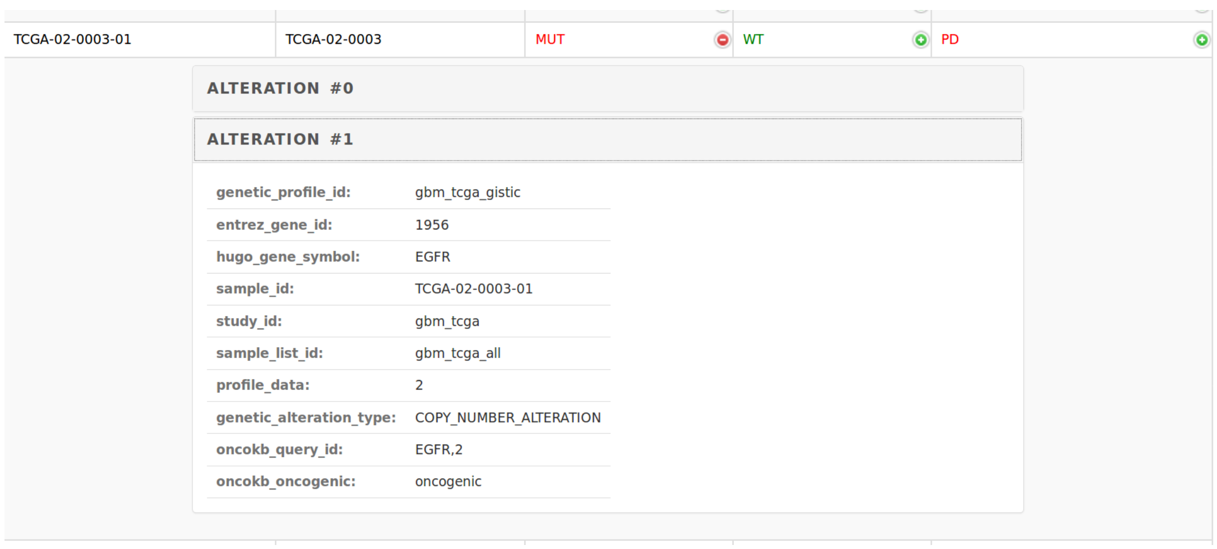
Excel CSV PDF Search:

Sample	Patient	EGFR	NF1	Drug Response
TCGA-02-0001-01	TCGA-02-0001	WT	WT	PD
TCGA-02-0003-01	TCGA-02-0003	MUT	WT	PD
TCGA-02-0004-01	TCGA-02-0004	WT	WT	PD
TCGA-02-0006-01	TCGA-02-0006	WT	WT	PD
TCGA-02-0007-01	TCGA-02-0007	WT	WT	PD
TCGA-02-0009-01	TCGA-02-0009	MUT	WT	PD
TCGA-02-0010-01	TCGA-02-0010	WT	WT	PD
TCGA-02-0011-01	TCGA-02-0011	WT	WT	PD
TCGA-02-0014-01	TCGA-02-0014	WT	WT	PD
TCGA-02-0015-01	TCGA-02-0015	WT	WT	PD

Showing 1 to 10 of 604 entries

Previous 1 2 3 4 5 … 61 Next

Figure 7.8: cBioPortal Downloader sample data panel.

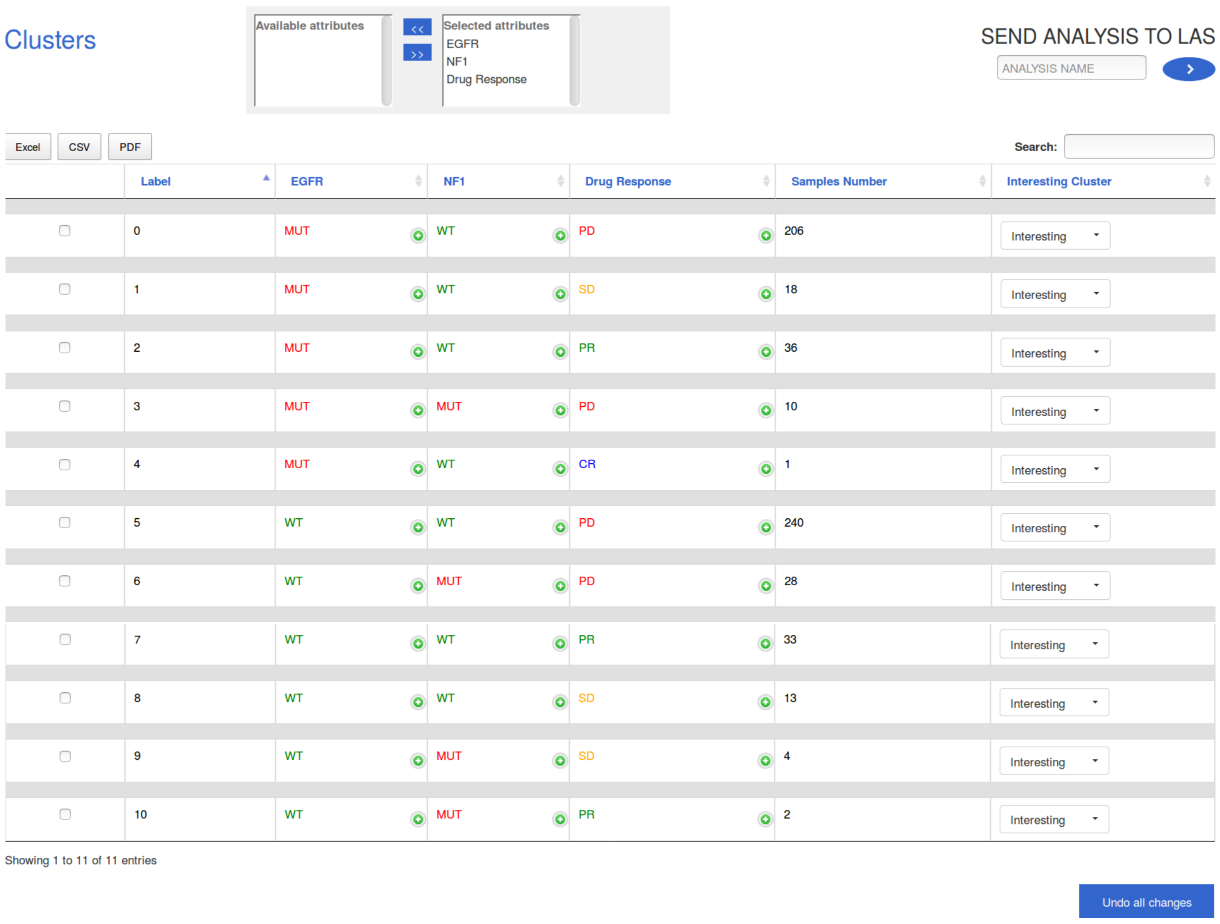

TCGA-02-0003-01	TCGA-02-0003	MUT	WT	PD

ALTERATION #0

ALTERATION #1

genetic_profile_id:	gbm_tcga_gistic
entrez_gene_id:	1956
hugo_gene_symbol:	EGFR
sample_id:	TCGA-02-0003-01
study_id:	gbm_tcga
sample_list_id:	gbm_tcga_all
profile_data:	2
genetic_alteration_type:	COPY_NUMBER_ALTERATION
oncokb_query_id:	EGFR,2
oncokb_oncogenic:	oncogenic

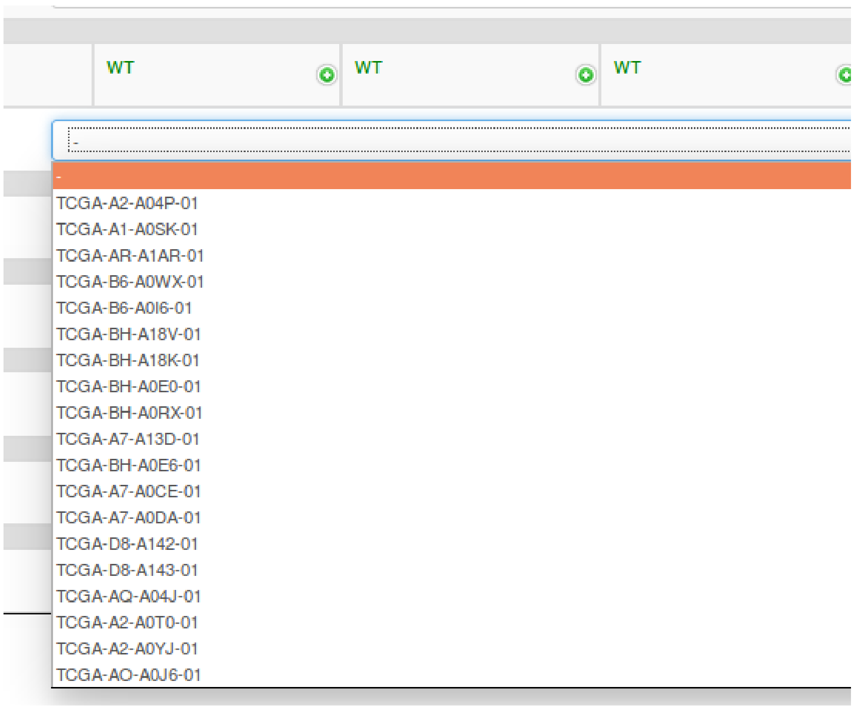
Figure 7.9: cBioPortal downloader: CNA details for sample TCGA-02-0003-01 on gene EGFR.



	Label	EGFR	NF1	Drug Response	Samples Number	Interesting Cluster
☐	0	MUT	WT	PD	206	Interesting
☐	1	MUT	WT	SD	18	Interesting
☐	2	MUT	WT	PR	36	Interesting
☐	3	MUT	MUT	PD	10	Interesting
☐	4	MUT	WT	CR	1	Interesting
☐	5	WT	WT	PD	240	Interesting
☐	6	WT	MUT	PD	28	Interesting
☐	7	WT	WT	PR	33	Interesting
☐	8	WT	WT	SD	13	Interesting
☐	9	WT	MUT	SD	4	Interesting
☐	10	WT	MUT	PR	2	Interesting

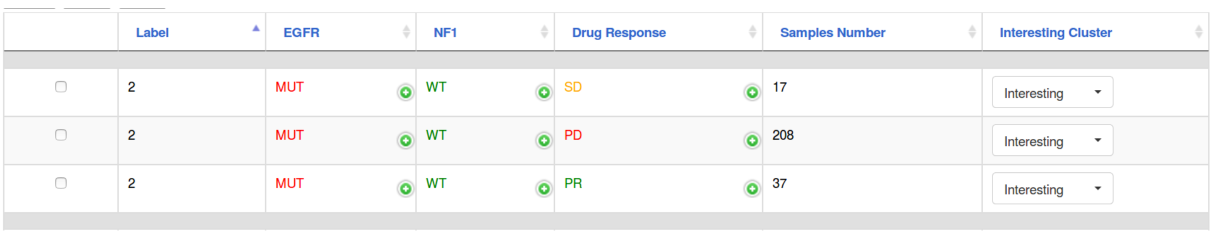
Showing 1 to 11 of 11 entries

Figure 7.10: cBioPortal Downloader cluster panel.



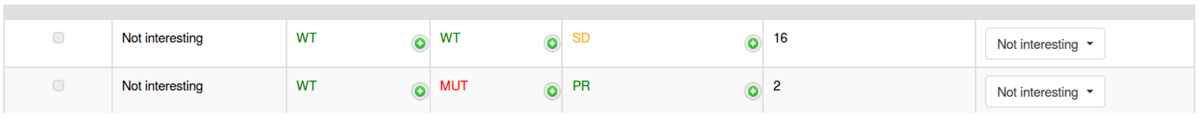
TCGA-A2-A04P-01
TCGA-A1-A0SK-01
TCGA-AR-A1AR-01
TCGA-B6-A0WX-01
TCGA-B6-A0I6-01
TCGA-BH-A18V-01
TCGA-BH-A18K-01
TCGA-BH-A0E0-01
TCGA-BH-A0RX-01
TCGA-A7-A13D-01
TCGA-BH-A0E6-01
TCGA-A7-A0CE-01
TCGA-A7-A0DA-01
TCGA-D8-A142-01
TCGA-D8-A143-01
TCGA-AQ-A04J-01
TCGA-A2-A0T0-01
TCGA-A2-A0YJ-01
TCGA-AO-A0J6-01

Figure 7.11: cBioPortal Downloader: cluster sample list.



	Label	EGFR	NF1	Drug Response	Samples Number	Interesting Cluster
▢	2	MUT	WT	SD	17	Interesting
▢	2	MUT	WT	PD	208	Interesting
▢	2	MUT	WT	PR	37	Interesting

Figure 7.12: cBioPortal Downloader: hybrid cluster.



▢	Not interesting	WT	WT	SD	16	Not interesting
▢	Not interesting	WT	MUT	PR	2	Not interesting

Figure 7.13: cBioPortal Downloader: interesting/un-interesting clusters.

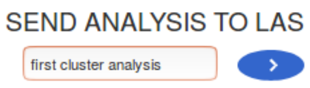


Figure 7.14: cBioPortal Downloader: send analysis to LAS.

Chapter 8

LAS annotation

The final objective of this framework is to integrate a new layer of knowledge within the LAS platform where biological and clinical data are aggregated to provide to the researchers the possibility to uncover interesting trends and correlations among them.

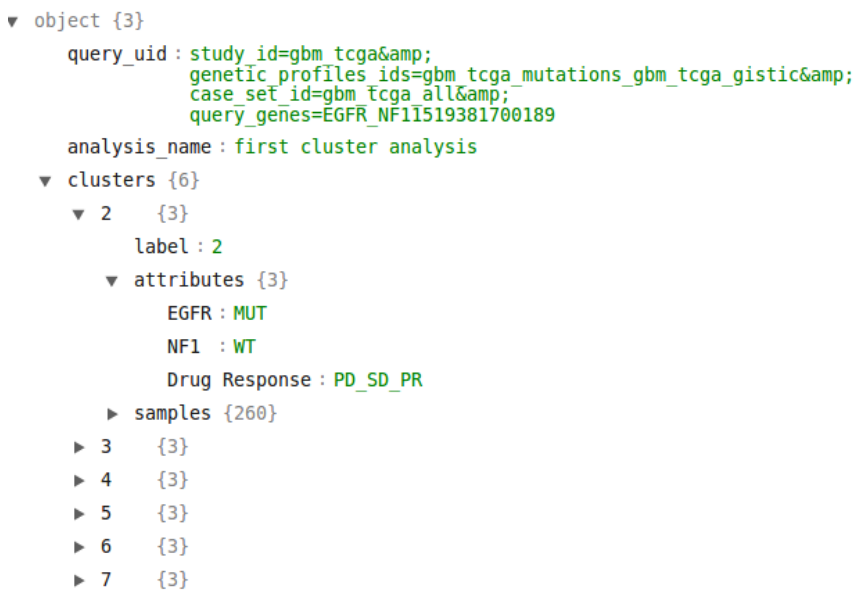
This process is managed by a new API which scans the data structure received from the cBioPortal Downloader (Figure 8.1 and Figure 8.2) and populates the LAS graph database with the information generated by the analyses performed throughout the overall dataflow. Specifically, first, it retrieves the export log file, generated when the dataset of interest has been exported from the LAS platform to the cBioPortal (see Section 8.2): this log file contains, for each sample, all layer-one annotation identifiers associated to the properties which have been investigated within the cBioPortal analysis. Then, it creates a new analysis node, setting a "name" property, which is assigned with the name the user provided at export time (see Section 7.4.2) and associates it to the layer-one annotation nodes. After that, for each one of the clusters it received, it generates a cluster reference node, with a "label" property whose value is put equal to the cluster label. This node is pointed by the layer-one reference nodes, which are correlated to the layer-one source annotations corresponding to its own dimensions. Finally, the API scans all the samples it received and for each one of them, it creates an annotation node, correlating it to the reference node describing the cluster it belongs to.

In order to manage this workflow, the LAS annotation model has been extended (see Section 8.1) and a data export log file has been designed (see Section 8.2).

8.1 Data model

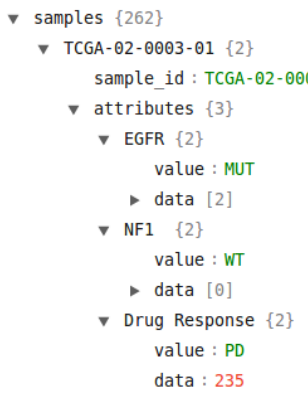
As already mentioned, a cluster analysis and the annotations it produces represent a second layer of knowledge, where the first layer is made of the laboratory experiments and the associated annotations. Figure 8.3 shows how the two layers have been integrated.

Analysis node Conceptually, an analysis is a process which investigates one or more features on a given input dataset and, as a result, produces an output dataset. This concept applies to LAS analyses, both to the first-layer and to the second-layer ones. First-layer analyses are the laboratory experiments, executed on a given sample set, to investigate specific sample genomic or clinical features and resulting in an annotation process, which establishes a relationship between the samples and the investigated features. Concretely, they are modeled and represented within the graph database by means of analysis nodes (L1_A), which are pointed to by their source data document (RAW) and point to the annotations (L1_AN) they generated. In the



```
▼ object {3}
    query_uid : study_id=gbm_tcga&amp;
                genetic_profiles_ids=gbm_tcga_mutations_gbm_tcga_gistic&amp;
                case_set_id=gbm_tcga_all&amp;
                query_genes=EGFR_NF11519381700189
    analysis_name : first cluster analysis
  ▼ clusters {6}
    ▼ 2    {3}
        label : 2
      ▼ attributes {3}
          EGFR : MUT
          NF1  : WT
          Drug Response : PD_SD_PR
      ▶ samples {260}
    ▶ 3    {3}
    ▶ 4    {3}
    ▶ 5    {3}
    ▶ 6    {3}
    ▶ 7    {3}
```

Figure 8.1: Cluster data structure.



```
▼ samples {262}
  ▼ TCGA-02-0003-01 {2}
      sample_id : TCGA-02-0003-01
    ▼ attributes {3}
      ▼ EGFR {2}
          value : MUT
        ▶ data [2]
      ▼ NF1  {2}
          value : WT
        ▶ data [0]
      ▼ Drug Response {2}
          value : PD
          data : 235
```

Figure 8.2: Cluster data structure: sample detail.

second layer, the conceptual model still holds, what changes are the entities which are concretely correlated: a cluster analysis analyzes a sample set on the basis of the genomic and clinical attributes which some laboratory experiments have already pointed out and annotated on samples. That is why cluster analysis nodes (L2_A) are pointed to by the set of the layer-one annotation nodes (L1_AN) and point to the second-layer annotations (L2_AN) they generated.

Annotation node Within the LAS genomic alteration model, every annotation is a semantic statement establishing a relationship, expressed by means of a predicate, between a biological sample (the subject of the statement) and a concept (the object of the statement), such as a genetic mutation. It is represented within the graph database as a node of type "annotation" with an incoming edge linking it to the biological sample (S) and an outgoing edge linking it the reference node in the knowledge base. Besides, annotation nodes are also pointed to by the node representing the process which generated them. First-layer and second-layer annotation nodes are identical; the only difference between them is the semantic meaning of the nodes with which they are related: at the first level, annotation nodes (L1_AN) are pointed to by a laboratory experiment node (L1_A) and point to a "feature" reference node (L1_R); at the second level, the analysis node represents a cluster analysis (L2_A), while the reference node

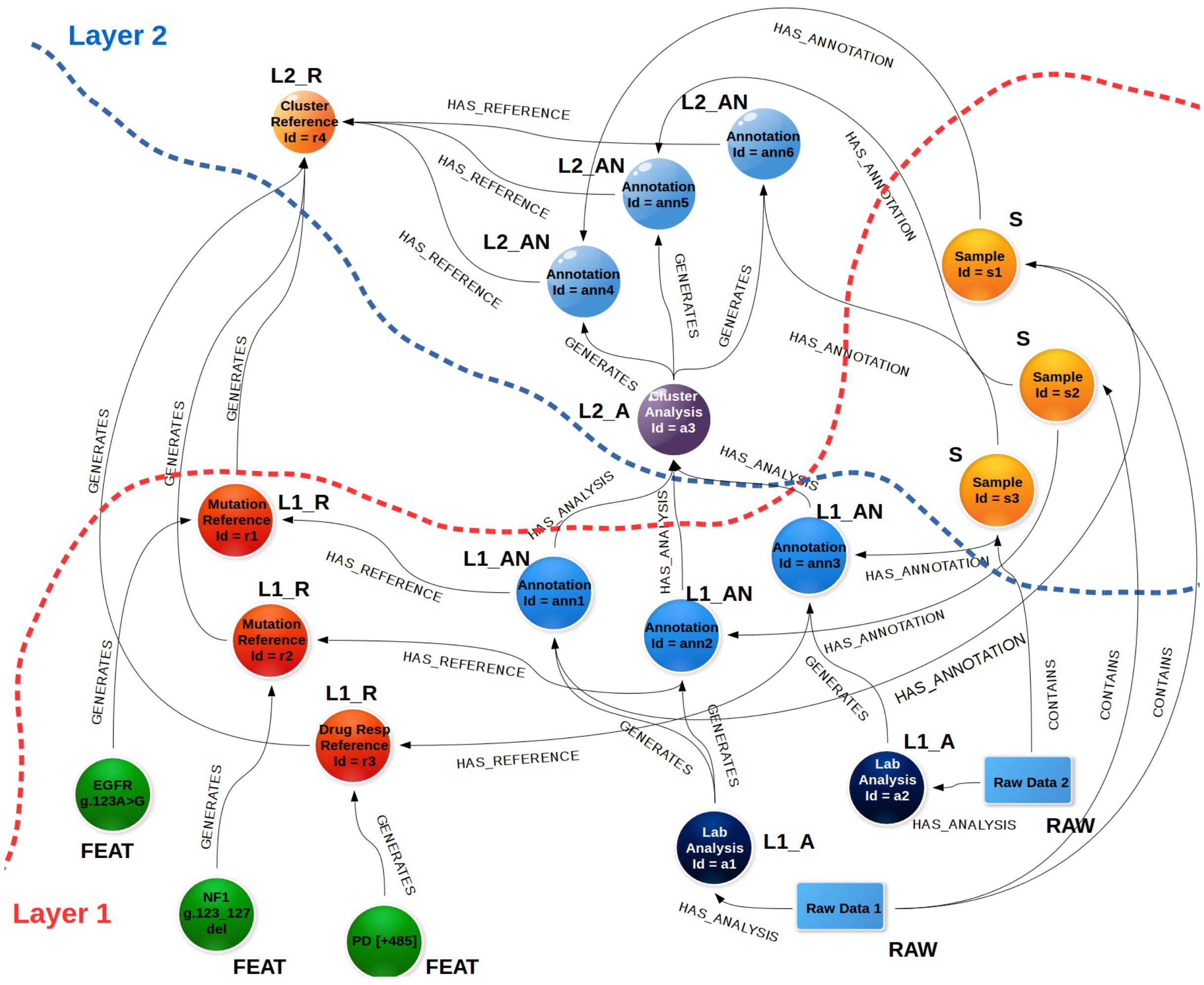


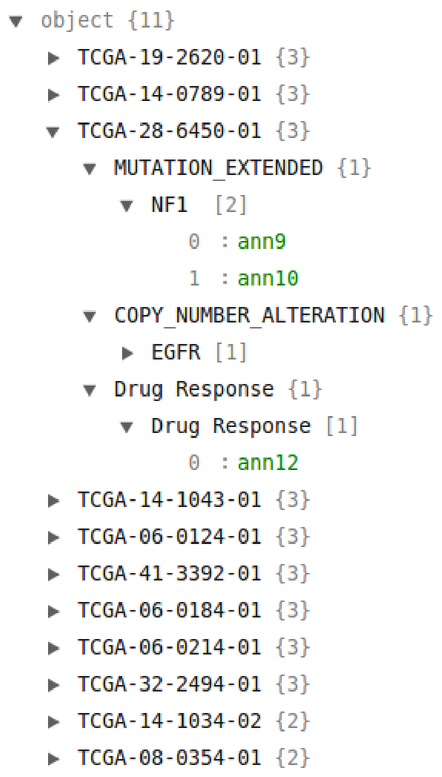
Figure 8.3: Layer 2 annotation model.

(L2_R) expresses a cluster signature. So, at the second layer, annotations establish a correlation among a cluster analysis, the analyzed samples and the features (label and dimensions) of the cluster they belong to.

Reference node Annotations are abstract concepts saying that a given sample has an undefined property: the property is specified by the reference node. Reference nodes actually express a sample feature by means of one or more relationships to the values which has been found for a certain property by a given analysis. First-layer reference nodes (L1_R) are pointed to by the genomic and clinical features (FEAT) which have been uncovered by a certain laboratory experiment; in the second layer, instead, reference nodes (L2_R) define a cluster signature for each one of the clusters found by the analysis by means of a set of links relating them to the level-one reference nodes (L1_R) corresponding to their own dimensions.

8.2 Export log

The API which receives the clusters from the cBioPortal Downloader, needs to identify the dataset and the sample annotations which originated the analysis. This task is accomplished by means of a data export log file. The log file is produced together with the file set needed to import the dataset into the cBioPortal. It is a JSON document and its name is equal to the dataset identifier: in this way, it can be immediately located into the file system, by parsing the cluster data structure `query_uid` field (Figure 8.1), which contains the cBioPortal query parameters, including the study identifier. Figure 8.4 shows the structure of an example export



```
▼ object {11}
  ▶ TCGA-19-2620-01 {3}
  ▶ TCGA-14-0789-01 {3}
  ▼ TCGA-28-6450-01 {3}
    ▼ MUTATION_EXTENDED {1}
      ▼ NF1  [2]
          0  : ann9
          1  : ann10
    ▼ COPY_NUMBER_ALTERATION {1}
      ▶ EGFR [1]
    ▼ Drug Response {1}
      ▼ Drug Response [1]
          0  : ann12
  ▶ TCGA-14-1043-01 {3}
  ▶ TCGA-06-0124-01 {3}
  ▶ TCGA-41-3392-01 {3}
  ▶ TCGA-06-0184-01 {3}
  ▶ TCGA-06-0214-01 {3}
  ▶ TCGA-32-2494-01 {3}
  ▶ TCGA-14-1034-02 {2}
  ▶ TCGA-08-0354-01 {2}
```

Figure 8.4: Example export log file structure.

log file. It appears as a list of sample identifiers, corresponding to the dataset samples, and, for each one of them, lists all the genomic and clinical profiles which have been exported as sample attributes within the cBioPortal; for each one of these profiles, it lists the unique IDs which identify the corresponding annotations within the LAS system. In this way, it is possible to trace back the annotations which originated the analysis and to perform the second-layer annotation process. Once again, the reason which motivated this choice is the need of keeping this framework simple. In the future, this mechanism may be replaced by more sophisticated one, such as a session identifier to be kept as a reference to the original dataset, throughout the dataflow (see Chapter 10).

Chapter 9

Use case

The purpose of this chapter is to present a real use-case scenario, to illustrate how the framework works.

Once the cBioPortal has been locally deployed, it is possible to import one or more LAS datasets into its database. We load two public cancer studies: "Bladder Urothelial Carcinoma (TCGA, Provisional)" and "Glioblastoma Multiforme (TCGA, Provisional)". As explained in Section 6.1.1, data to be imported must be organized in a set of properly formatted files and put into a single folder. Let us look into the "Glioblastoma Multiforme" import folder.

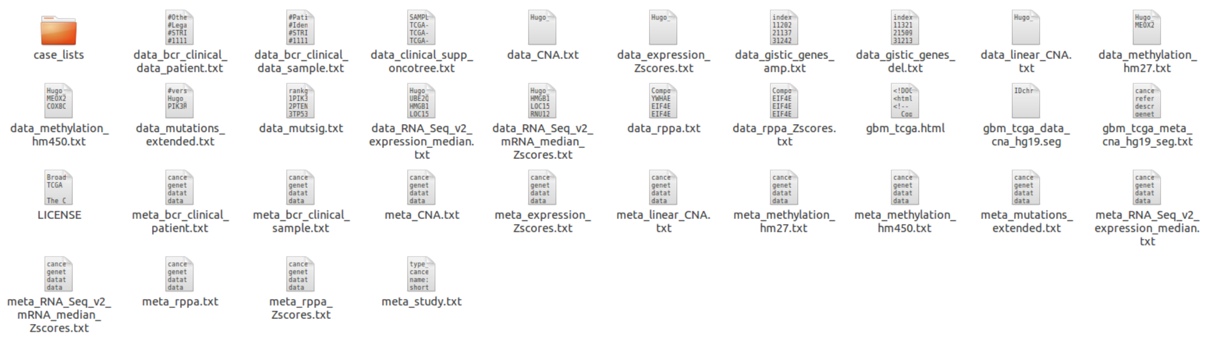

case_lists
data_bcr_clinical_data_patient.txt
data_bcr_clinical_data_sample.txt
data_clinical_supp_oncotree.txt
data_CNA.txt
data_expression_Zscores.txt
data_gistic_genes_amp.txt
data_gistic_genes_del.txt
data_linear_CNA.txt
data_methylation_hm27.txt
data_methylation_hm450.txt
data_mutations_extended.txt
data_mutsig.txt
data_RNA_Seq_v2_expression_median.txt
data_RNA_Seq_v2_mRNA_median_Zscores.txt
data_rppa.txt
data_rppa_Zscores.txt
gbm_tcga.html
gbm_tcga_data_cna_hg19.seg
gbm_tcga_meta_cna_hg19_seg.txt
LICENSE
meta_bcr_clinical_patient.txt
meta_bcr_clinical_sample.txt
meta_CNA.txt
meta_expression_Zscores.txt
meta_linear_CNA.txt
meta_methylation_hm27.txt
meta_methylation_hm450.txt
meta_mutations_extended.txt
meta_RNA_Seq_v2_expression_median.txt
meta_RNA_Seq_v2_mRNA_median_Zscores.txt
meta_rppa.txt
meta_rppa_Zscores.txt
meta_study.txt

Figure 9.1: Glioblastoma Multiforme import folder.

As Figure 9.1 shows, apart from the mandatory files (`meta_study.txt`, `data_bcr_clinical_data_patient.txt`, `data_bcr_clinical_data_sample.txt`, `meta_bcr_clinical_data_patient.txt`, `meta_bcr_clinical_data_sample.txt`), there is a bunch of additional data files describing the genomic profiles available for this study (DNA methylation, copy number alterations, sequence alterations, RNA and protein expression information etc.). All of these files are organized in a matrix format, where the columns correspond to the sample identifiers and the rows to the genomic profile name. These are the files which are downloaded when the "Download" tab is selected (see Section 6.2). Moreover, there is a meta file, for each one of the data files, describing its content. Finally, there is a sub-folder named "case_list" which contains a set of files listing the case/patient set corresponding to each genomic profile available for these study.

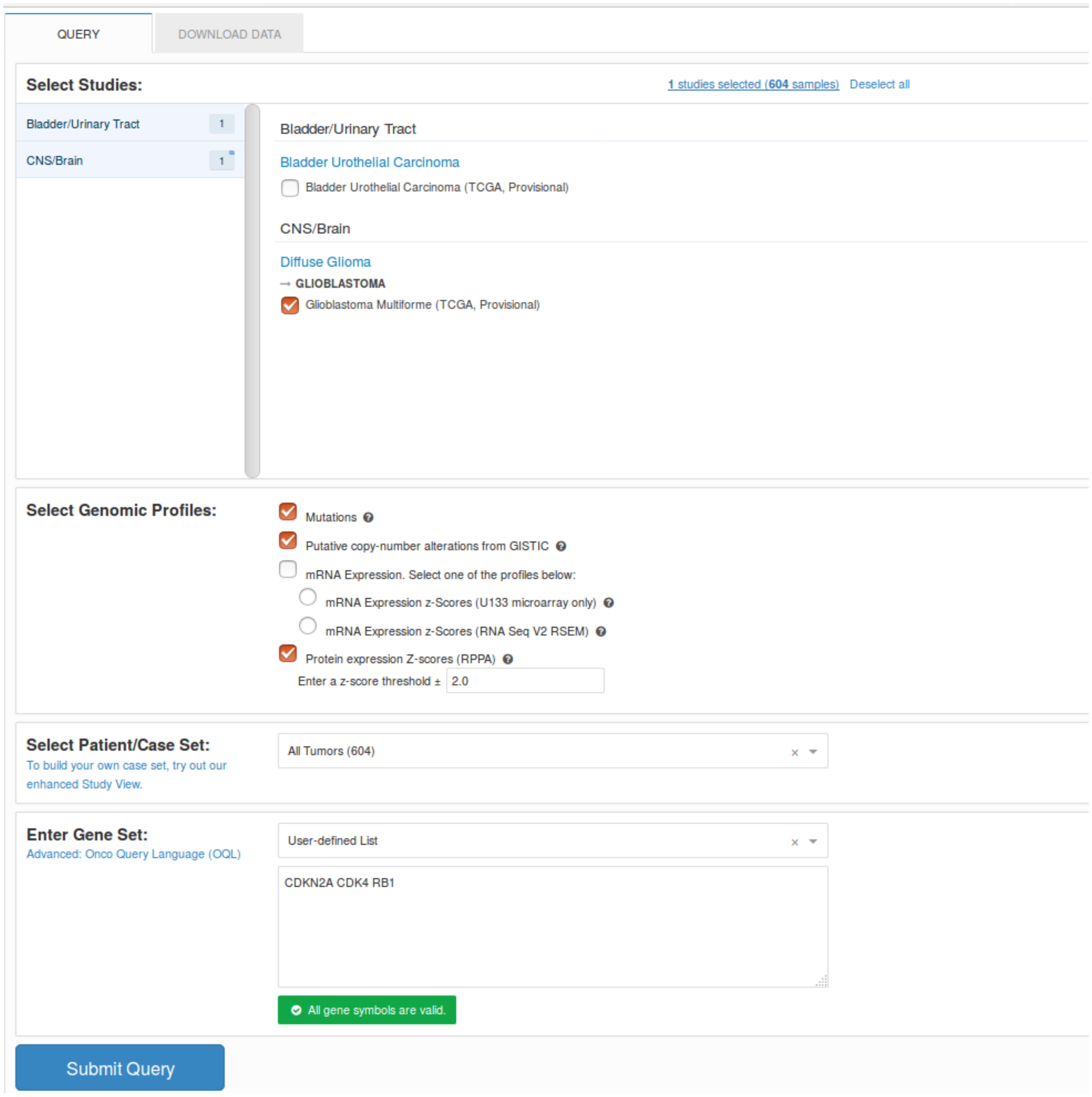
We are ready to perform our first query (Figure 9.2). Let us perform the four needed steps:

1. Select a cancer study: "Glioblastoma Multiforme (TCGA, Provisional)".

2. Select the genomic profiles: "Mutations" and "Putative copy-number alterations". Note

that these profiles are selected by default. For RNA and protein data, they can be selected when available and the default z-score threshold[1] can be optionally modified by the user.

3. Select patient/case from the dropdown menu or build a custom patient set: "All Tumors (604)". Note that to enter a user-defined case list, this option must be selected from the dropdown menu and enter the case IDs separated by a space.
4. Enter genes of interest manually or by selecting from predefined lists: CDKN2A CDK4 RB1. Note that the gene set may be refined my means of the Onco Query Language (see the dedicated paragraph in Section 4.2).



Select Studies: 1 studies selected (604 samples) Deselect all

Bladder/Urinary Tract 1
CNS/Brain 1

Bladder/Urinary Tract
Bladder Urothelial Carcinoma
☐ Bladder Urothelial Carcinoma (TCGA, Provisional)

CNS/Brain
Diffuse Glioma
→ GLIOBLASTOMA
[✓ Glioblastoma Multiforme (TCGA, Provisional)]

Select Genomic Profiles:
[✓ Mutations]
[✓ Putative copy-number alterations from GISTIC]
☐ mRNA Expression. Select one of the profiles below:
○ mRNA Expression z-Scores (U133 microarray only)
○ mRNA Expression z-Scores (RNA Seq V2 RSEM)
[✓ Protein expression Z-scores (RPPA)]
Enter a z-score threshold ± 2.0

Select Patient/Case Set: All Tumors (604)
To build your own case set, try out our enhanced Study View.

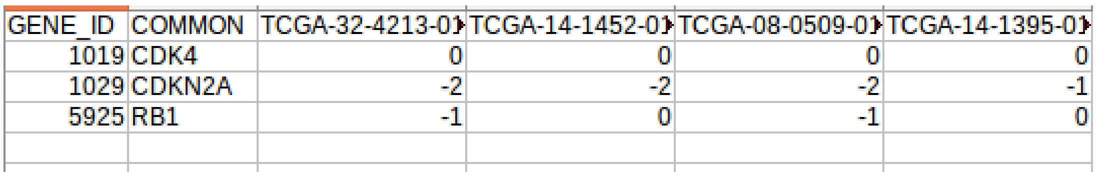
Enter Gene Set: User-defined List
Advanced: Onco Query Language (OQL)
CDKN2A CDK4 RB1
All gene symbols are valid.

Figure 9.2: cBioPortal query (use-case).

As first operation, we select the "Download" tab and download copy number alteration data in text format. Figure 9.3 shows the first columns of the file: the first two columns list the

[1]A z-score threshold indicates the number of standard deviations away from the mean of expression in the reference.

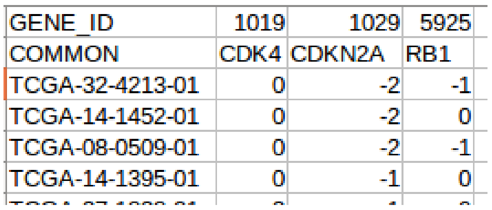
queried genes, identified both by their HUGO symbol and by their Entrez Gene identifier. The following four columns contain CNA information about four samples, whose name is indicated in the the first row. It can be noticed that CNA values are expressed by means of positive or negative integer numbers: -2 = homozygous deletion; -1 = hemizygous deletion; 0 = neutral/no change; 1 = gain; 2 = amplification.



GENE_ID	COMMON	TCGA-32-4213-01	TCGA-14-1452-01	TCGA-08-0509-01	TCGA-14-1395-01
1019	CDK4	0	0	0	0
1029	CDKN2A	-2	-2	-2	-1
5925	RB1	-1	0	-1	0

Figure 9.3: cBioPortal download file with CNA data (use-case).

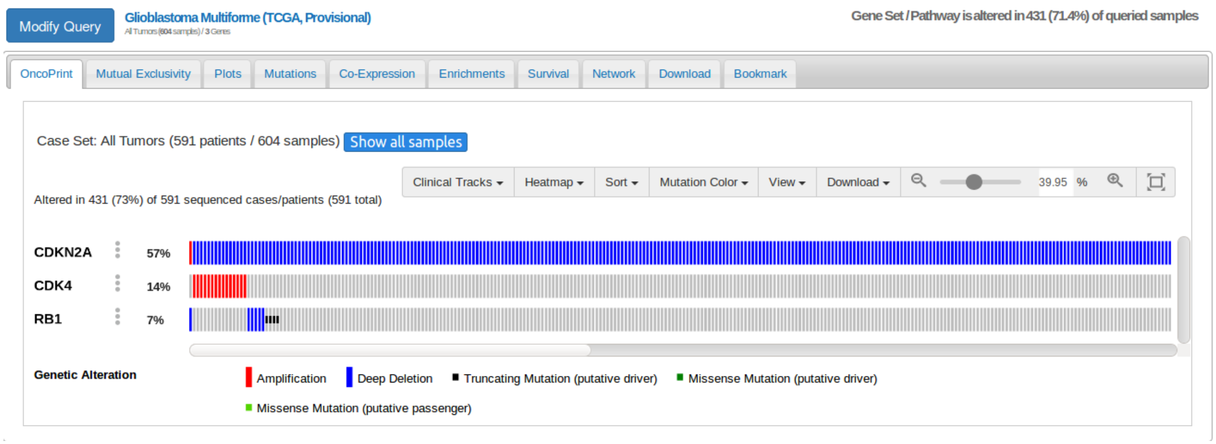
If the "Transpose data matrix" option is checked, the data matrix is transposed so that its columns correspond to the genes and its rows to the samples (Figure 9.4).



GENE_ID	1019	1029	5925
COMMON	CDK4	CDKN2A	RB1
TCGA-32-4213-01	0	-2	-1
TCGA-14-1452-01	0	-2	0
TCGA-08-0509-01	0	-2	-1
TCGA-14-1395-01	0	-1	0

Figure 9.4: cBioPortal transposed download file with CNA data (use-case).

Let us, now, select the "Query" tab: the portal performs its computation and shows the results in separate tabs. Let us analyze the content of the OncoPrint tab (Figure 9.5).



Glioblastoma Multiforme (TCGA, Provisional)

All Tumors (604 samples) / 3 Genes

Gene Set / Pathway is altered in 431 (71.4%) of queried samples

Case Set: All Tumors (591 patients / 604 samples)

Altered in 431 (73%) of 591 sequenced cases/patients (591 total)

CDKN2A	57%
CDK4	14%
RB1	7%

Genetic Alteration: 🟥 Amplification; 🟦 Deep Deletion; ■ Truncating Mutation (putative driver); 🟩 Missense Mutation (putative driver); 🟩 Missense Mutation (putative passenger)

Figure 9.5: cBioPortal oncoprint patient visualization (use-case).

The OncoPrint resulting from the example query is, initially, made of three rows: one for each of the query genes representing sample data aggregated by patient. The portal returned 604 samples, corresponding to 591 patients: alterations have been found in 431 (73%) of 591 cases. For CDKN2A (altered in 57% of the cases), most alterations are deep deletions; CDK4 (altered in 14% of the cases) is mainly interested by amplification events; for RB1 (altered in 7% of the cases), the most recurrent variations are, again, deep deletions. As for mutations, they are a few for CDKN2A and many for RB1, while there is any mutation for CDK4. The alterations in these three genes are distributed in a nearly mutually exclusive way across samples. Figure 9.6 shows how the visualization and the statistics change if we press the "View" button on the toolbar and select the "Events per sample" option. Additional details are available by

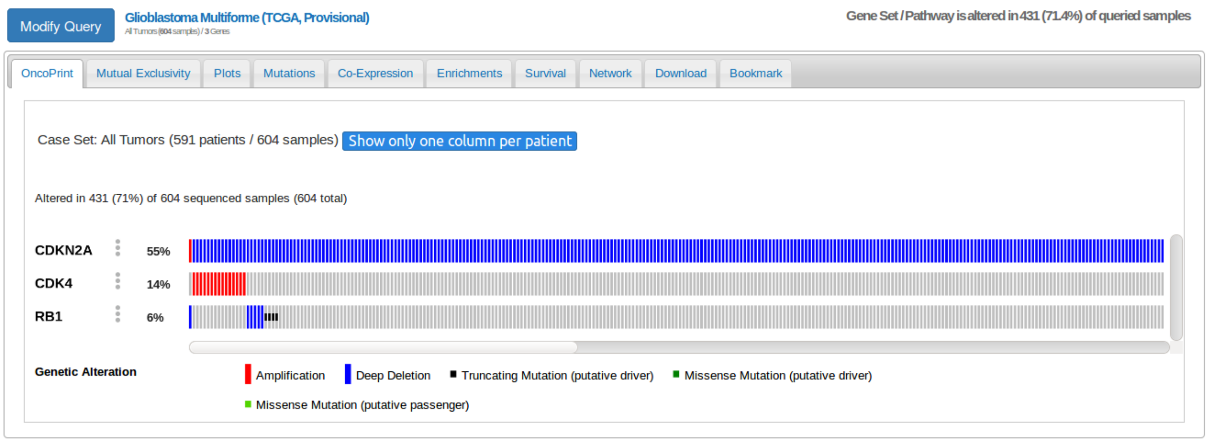

Modify Query
Glioblastoma Multiforme (TCGA, Provisional)
All Tumors (604 samples) / 3 Genes
Gene Set / Pathway is altered in 431 (71.4%) of queried samples
OncoPrint
Mutual Exclusivity
Plots
Mutations
Co-Expression
Enrichments
Survival
Network
Download
Bookmark
Case Set: All Tumors (591 patients / 604 samples)
Show only one column per patient
Altered in 431 (71%) of 604 sequenced samples (604 total)
CDKN2A
55%
CDK4
14%
RB1
6%
Genetic Alteration
Amplification
Deep Deletion
Truncating Mutation (putative driver)
Missense Mutation (putative driver)
Missense Mutation (putative passenger)

Figure 9.6: cBioPortal oncoprint sample visualization (use-case).

mousing over an event indicated on a gene. For example, let us mouse over the first event on the CDKN2A row: a temporary window appears, telling that this event corresponds to a CNA event and, specifically, to an amplification of the corresponding gene on the sample whose ID is TCGA-06-0146-01 (Figure 9.7). If we click on the sample identifier, the portal redirects us the

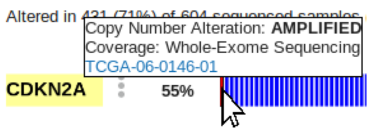
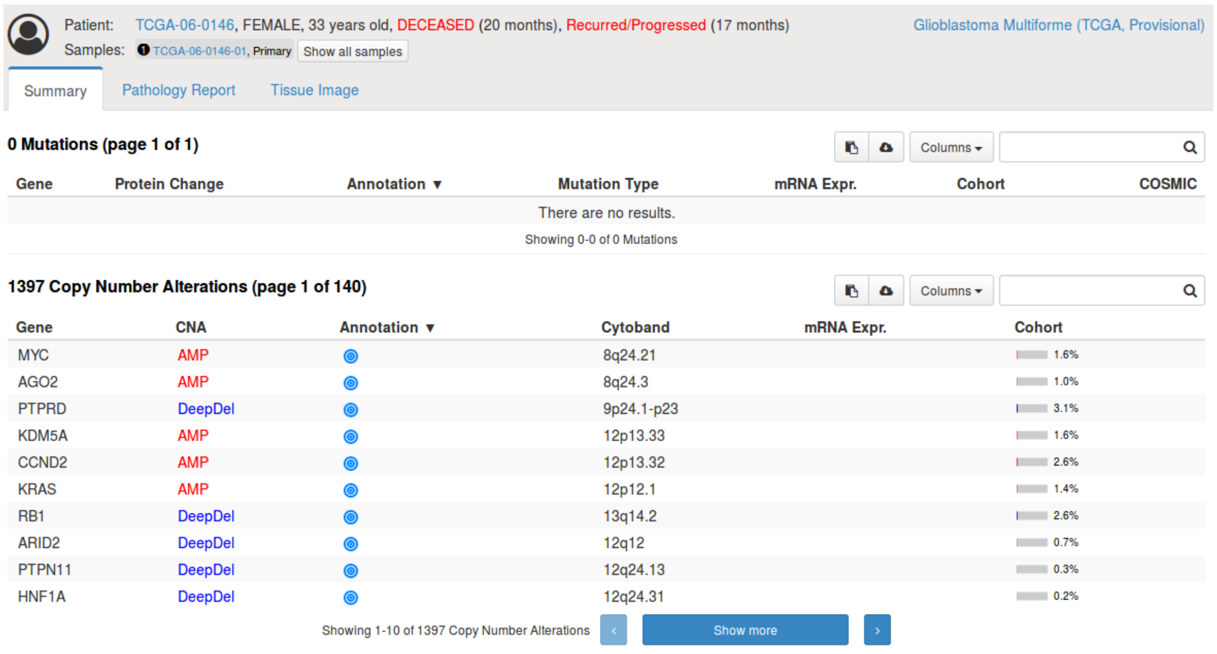

Altered in 431 (71%) of 604 sequenced samples
Copy Number Alteration: AMPLIFIED
Coverage: Whole-Exome Sequencing
TCGA-06-0146-01
CDKN2A
55%

Figure 9.7: cBioPortal CNA event details (use-case).

the patient view: in this way, we find out that this sample comes from a 33-years old woman, identified within the platform by the ID TCGA-06-0146; she deceased after 20 months from the initial diagnosis and that, for a period of 17 months, the disease progressed. Additionally, we know that this is the only sample associated to this patient; finally, two tables present mutation and CNA information about the entire gene set investigated by this study. Let us, now, go back to the OncoPrint to add a couple of clinical tracks; specifically, we add "Drug Response", describing the patient response to drug therapy in terms of tumor volume change, and "Karnofsky Performance Score", representing the functional capabilities of a person. According to the legend, both attributes can a assume values in a certain range; this value is represented by variable-height bars: small bars correspond to small values and, vice-versa, high bars correspond to large values. This visualization offer the possibility to visually correlate clinical track variations to genomic events in the corresponding samples.

If we trigger the cBioPortal Downloader, by pressing on its icon, a new tab opens up (Figure 9.10), showing the sample data collected by the cBioPortal, filtered according to the query criteria and enriched with the clinical tracks we added; furthermore, the new tab contains also the result of the cluster analysis performed on the exported samples. The data panel (Figure 9.11) collects all the data presented within the OncoPrint, organized into a table: the 604 rows correspond to the investigated samples, while the columns correspond to the genomic and the clinical tracks. If we ask for the alteration details for the gene CDKN2A in sample TCGA-02-0001-01, which appears to be mutated, we can read that on that gene a copy number alteration and, specifically, a deep deletion (`profile_data` = -2) has been found (Figure 9.12). Similarly we could press on the given button for all other expansible cells to obtain additional


Patient: TCGA-06-0146, FEMALE, 33 years old, DECEASED (20 months), Recurred/Progressed (17 months) Glioblastoma Multiforme (TCGA, Provisional)

Samples: TCGA-06-0146-01, Primary Show all samples

Summary Pathology Report Tissue Image

0 Mutations (page 1 of 1)

Gene	Protein Change	Annotation ▼	Mutation Type	mRNA Expr.	Cohort	COSMIC
			There are no results.			

Showing 0-0 of 0 Mutations

1397 Copy Number Alterations (page 1 of 140)

Gene	CNA	Annotation ▼	Cytoband	mRNA Expr.	Cohort
MYC	AMP	◎	8q24.21		1.6%
AGO2	AMP	◎	8q24.3		1.0%
PTPRD	DeepDel	◎	9p24.1-p23		3.1%
KDM5A	AMP	◎	12p13.33		1.6%
CCND2	AMP	◎	12p13.32		2.6%
KRAS	AMP	◎	12p12.1		1.4%
RB1	DeepDel	◎	13q14.2		2.6%
ARID2	DeepDel	◎	12q12		0.7%
PTPN11	DeepDel	◎	12q24.13		0.3%
HNF1A	DeepDel	◎	12q24.31		0.2%

Showing 1-10 of 1397 Copy Number Alterations

Figure 9.8: cBioPortal patient view (use-case).

details about the presented information. The cluster panel (Figure 9.13) shows, through a table, the result of the cluster analysis it has been performed on the exported sample data. As it is possible to read in the text appearing below the table ("Showing 1 to 16 of 16 entries"), the algorithm has found 16 clusters identified by four dimensions: the query genes ("CDKN2A", "CDK4", "RB1") and "Drug Response"; "Karnofsky Performance Score" has been discarded since it assumes continuous values in the range [1, 100]. As for the data table, also here it is possible to obtain additional information by pressing the button on one of the cells: in this case, first, a list of the cluster samples is presented and, then, by selecting one of the sample identifiers, the details about the attribute corresponding to the cluster dimension mapped to that column, for that sample, are shown. In this case, we ask for the details about CDKN2A alterations for the Cluster 0 samples (Figure 9.14) and, then, we select the sample TCGA-06-0146-01 (Figure 9.15): in this way, we find out that on that gene, in this sample, a copy number alteration, specifically, an amplification (`profile_data` = 2), has been found. And this is in accordance with what is specified by the dimension value, for this cluster, which is set as "MUT". Now, we remove RB1 from the cluster dimensions; as a result clusters are rebuilt and, as expected, their number is reduced: they become 12 (Figure 9.16). Finally, we merge together clusters "0", "1" and "3", assigning to all of them the label "3"; similarly, we combine also the clusters "4", "5" and "6", to build one super-cluster, named "6"; finally, we flag clusters "11" and "12" as not interesting. Figure 9.17 shows how the cluster table appears, now. We have 6 interesting clusters, two of which are hybrid ones, obtained by merging some of the "pure" clusters resulting from the clustering process; while two clusters have been flagged as "non-interesting". Finally, we are ready to send the result of the analyses performed throughout the framework dataflow to the LAS platform, by pressing the given button.

Once data have been imported into the LAS, we can access the Web-platform and, through, a Neo4j instance, explore the set of new nodes added to the graph database. Here, it is not possible to show the result of the overall cluster analysis, as it are appended to the graph, since it would require to show some hundreds of nodes. So we focus on the cluster "2", which is a

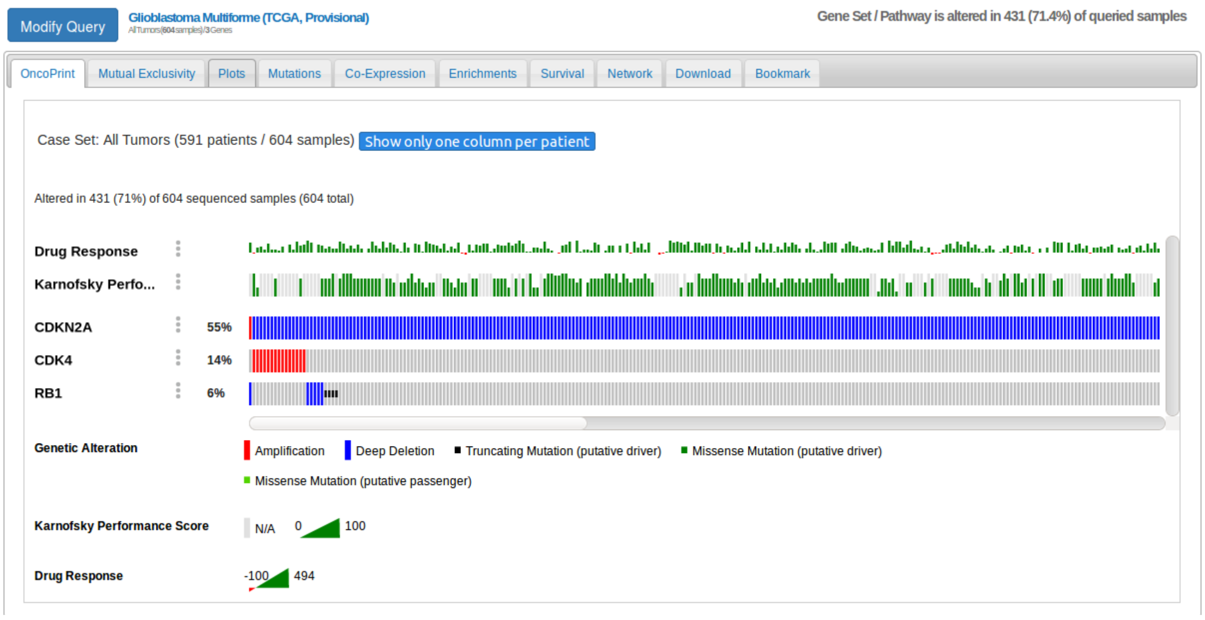
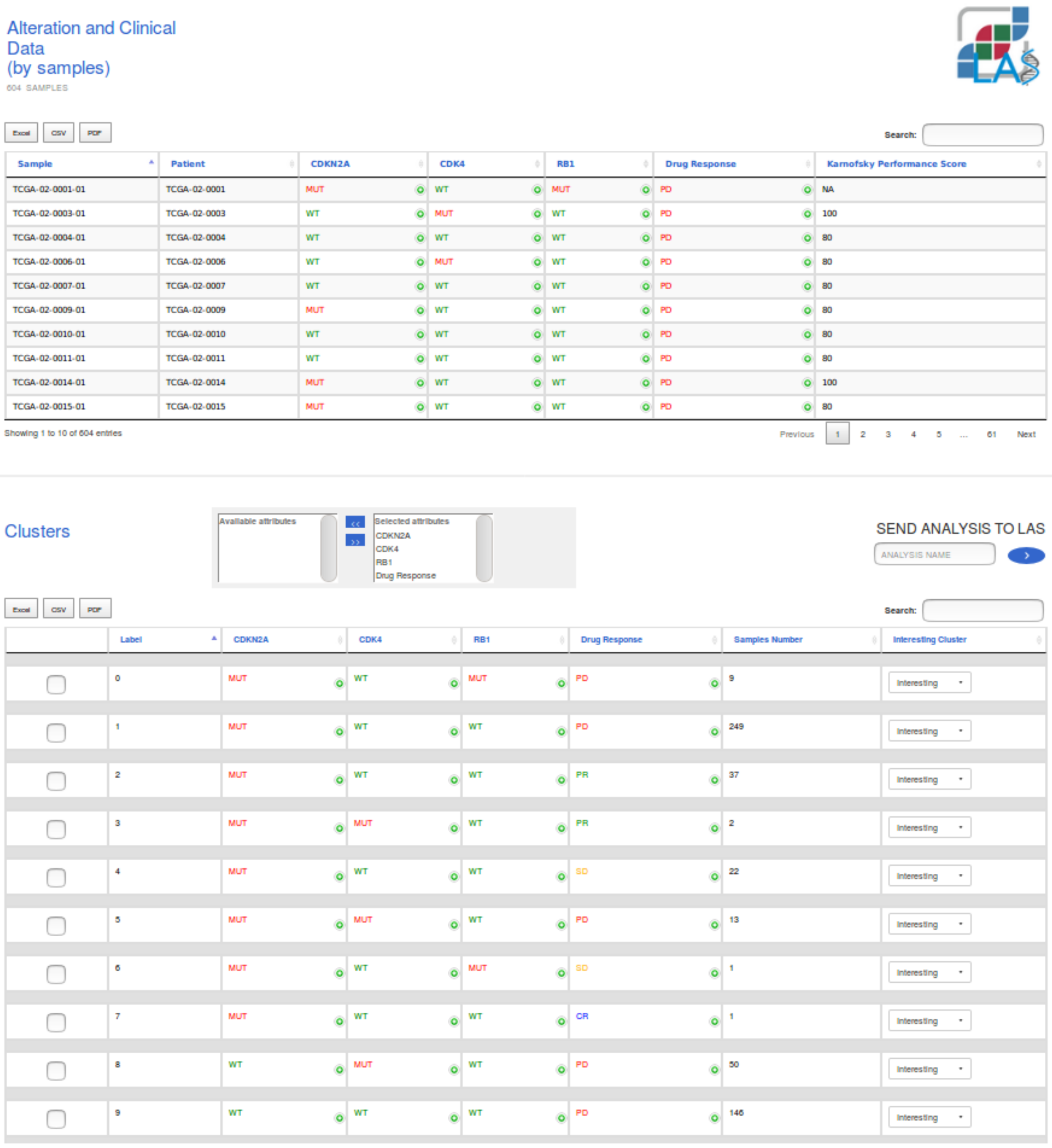

Modify Query
Glioblastoma Multiforme (TCGA, Provisional)
All Tumors (604 samples) / 3 Genes
Gene Set / Pathway is altered in 431 (71.4%) of queried samples
OncoPrint
Mutual Exclusivity
Plots
Mutations
Co-Expression
Enrichments
Survival
Network
Download
Bookmark
Case Set: All Tumors (591 patients / 604 samples)
Show only one column per patient
Altered in 431 (71%) of 604 sequenced samples (604 total)
Drug Response
Karnofsky Perfo...
CDKN2A
55%
CDK4
14%
RB1
6%
Genetic Alteration
Amplification
Deep Deletion
Truncating Mutation (putative driver)
Missense Mutation (putative driver)
Missense Mutation (putative passenger)
Karnofsky Performance Score
N/A
0
100
Drug Response
-100
494

Figure 9.9: cBioPortal OncoPrint with clinical tracks (use-case).

small one: it is made, only of two samples. As Figure 9.17 shows, this cluster is characterized by the following signature: "CDKN2A" = "MUT"; "CDK4" = "MUT"; "Drug Response" = "PR". This means that for each of the samples, three layer-one annotations have been used for this analysis and, all of them, point to the new analysis node. In this case, we consider only the subset of the six layer-one annotations, attached to the two cluster samples. As Figure 9.18 shows, for each one of the cluster samples, a layer-two annotation is produced and each one of them point to a single reference node, representing the cluster the samples belong to. Finally, the cluster reference node is pointed to by the six layer-one reference nodes, corresponding to the genomic and clinical features described by the layer-one annotations and corresponding to the cluster dimensions.


Alteration and Clinical Data (by samples)

604 SAMPLES

Sample	Patient	CDKN2A	CDK4	RB1	Drug Response	Karnofsky Performance Score
TCGA-02-0001-01	TCGA-02-0001	MUT	WT	MUT	PD	NA
TCGA-02-0003-01	TCGA-02-0003	WT	MUT	WT	PD	100
TCGA-02-0004-01	TCGA-02-0004	WT	WT	WT	PD	80
TCGA-02-0006-01	TCGA-02-0006	WT	MUT	WT	PD	80
TCGA-02-0007-01	TCGA-02-0007	WT	WT	WT	PD	80
TCGA-02-0009-01	TCGA-02-0009	MUT	WT	WT	PD	80
TCGA-02-0010-01	TCGA-02-0010	WT	WT	WT	PD	80
TCGA-02-0011-01	TCGA-02-0011	WT	WT	WT	PD	80
TCGA-02-0014-01	TCGA-02-0014	MUT	WT	WT	PD	100
TCGA-02-0015-01	TCGA-02-0015	MUT	WT	WT	PD	80

Showing 1 to 10 of 604 entries

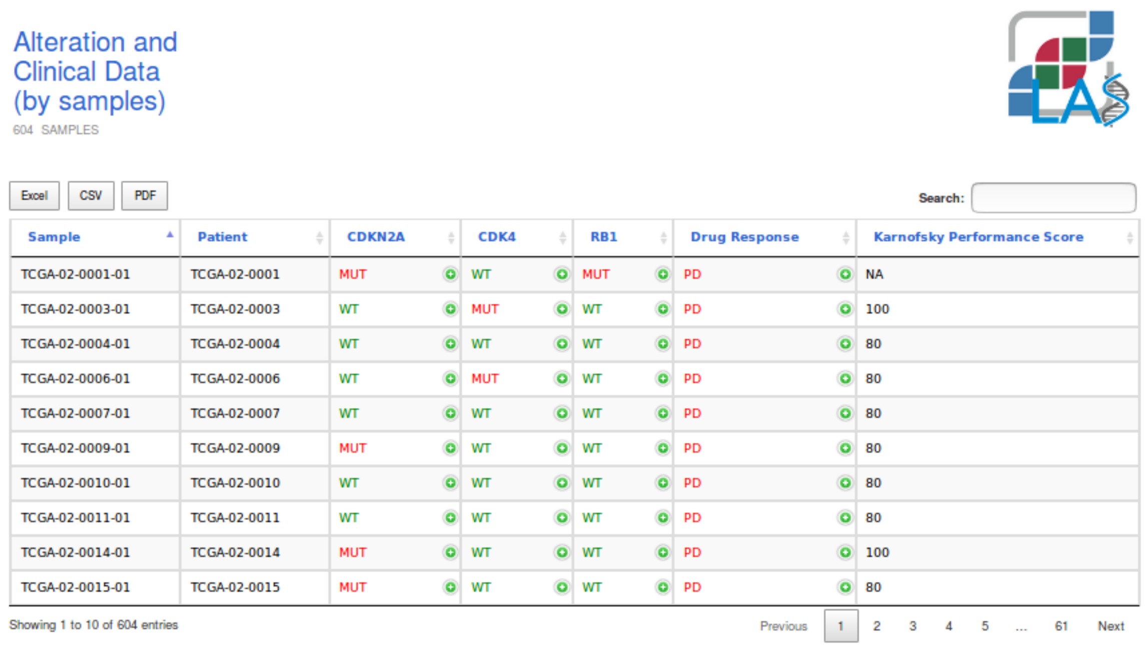
Clusters

CDKN2A
CDK4
RB1
Drug Response

SEND ANALYSIS TO LAS

	Label	CDKN2A	CDK4	RB1	Drug Response	Samples Number	Interesting Cluster
☐	0	MUT	WT	MUT	PD	9	Interesting
☐	1	MUT	WT	WT	PD	249	Interesting
☐	2	MUT	WT	WT	PR	37	Interesting
☐	3	MUT	MUT	WT	PR	2	Interesting
☐	4	MUT	WT	WT	SD	22	Interesting
☐	5	MUT	MUT	WT	PD	13	Interesting
☐	6	MUT	WT	MUT	SD	1	Interesting
☐	7	MUT	WT	WT	CR	1	Interesting
☐	8	WT	MUT	WT	PD	50	Interesting
☐	9	WT	WT	WT	PD	146	Interesting

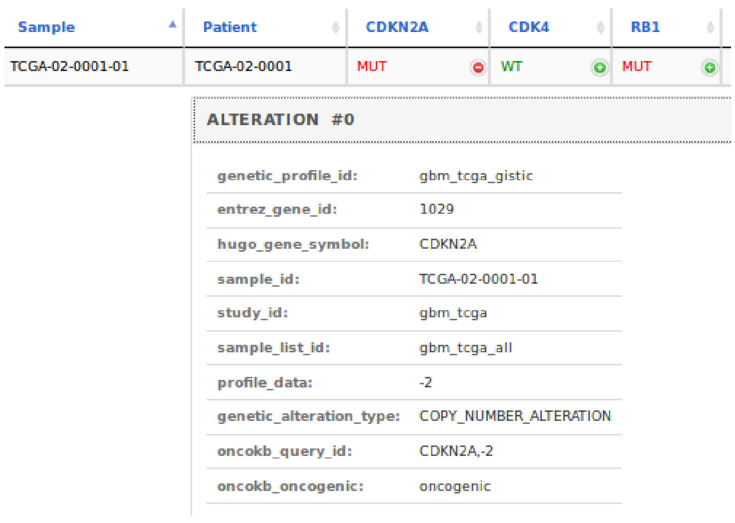
Figure 9.10: cBioPortal Downloader view (use-case).



Sample	Patient	CDKN2A	CDK4	RB1	Drug Response	Karnofsky Performance Score
TCGA-02-0001-01	TCGA-02-0001	MUT	WT	MUT	PD	NA
TCGA-02-0003-01	TCGA-02-0003	WT	MUT	WT	PD	100
TCGA-02-0004-01	TCGA-02-0004	WT	WT	WT	PD	80
TCGA-02-0006-01	TCGA-02-0006	WT	MUT	WT	PD	80
TCGA-02-0007-01	TCGA-02-0007	WT	WT	WT	PD	80
TCGA-02-0009-01	TCGA-02-0009	MUT	WT	WT	PD	80
TCGA-02-0010-01	TCGA-02-0010	WT	WT	WT	PD	80
TCGA-02-0011-01	TCGA-02-0011	WT	WT	WT	PD	80
TCGA-02-0014-01	TCGA-02-0014	MUT	WT	WT	PD	100
TCGA-02-0015-01	TCGA-02-0015	MUT	WT	WT	PD	80

Showing 1 to 10 of 604 entries

Figure 9.11: cBioPortal Downloader data panel (use-case).

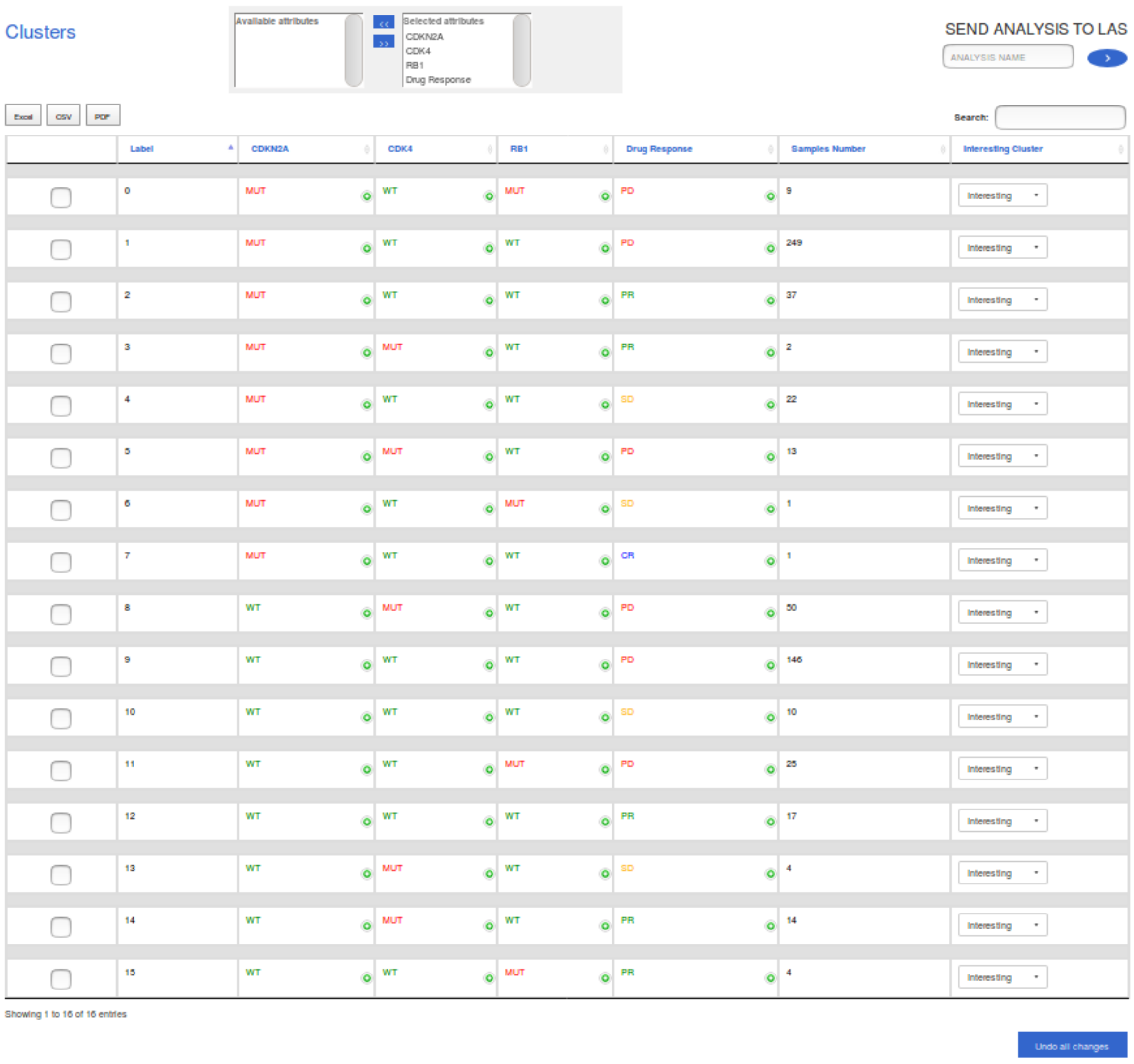


Sample	Patient	CDKN2A	CDK4	RB1
TCGA-02-0001-01	TCGA-02-0001	MUT	WT	MUT

ALTERATION #0

genetic_profile_id:	gbm_tcga_gistic
entrez_gene_id:	1029
hugo_gene_symbol:	CDKN2A
sample_id:	TCGA-02-0001-01
study_id:	gbm_tcga
sample_list_id:	gbm_tcga_all
profile_data:	-2
genetic_alteration_type:	COPY_NUMBER_ALTERATION
oncokb_query_id:	CDKN2A,-2
oncokb_oncogenic:	oncogenic

Figure 9.12: cBioPortal Downloader data panel: alteration details (use-case).


Clusters

Available attributes

Selected attributes
CDKN2A
CDK4
RB1
Drug Response

SEND ANALYSIS TO LAS
ANALYSIS NAME

Excel CSV PDF

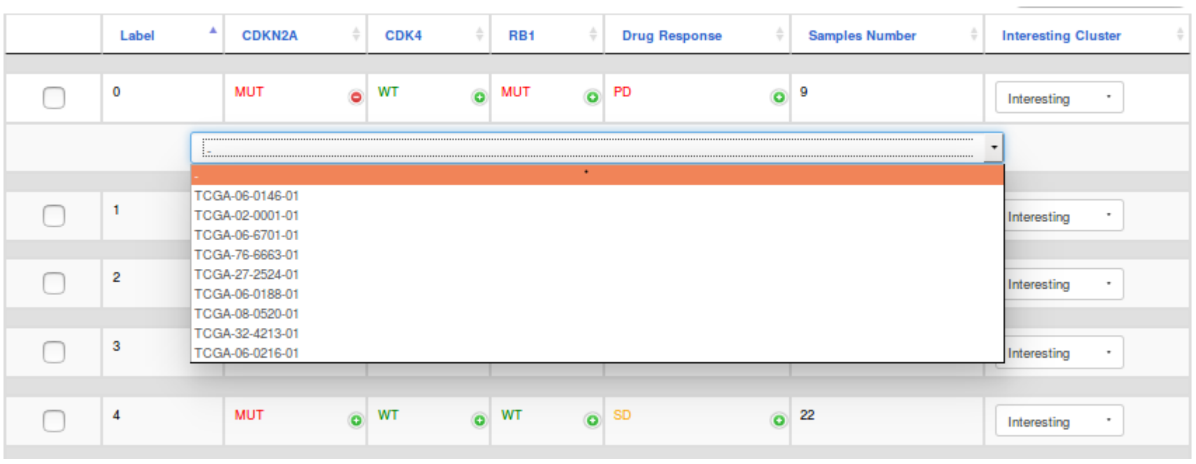
Search:

	Label	CDKN2A	CDK4	RB1	Drug Response	Samples Number	Interesting Cluster
	0	MUT	WT	MUT	PD	9	Interesting
	1	MUT	WT	WT	PD	249	Interesting
	2	MUT	WT	WT	PR	37	Interesting
	3	MUT	MUT	WT	PR	2	Interesting
	4	MUT	WT	WT	SD	22	Interesting
	5	MUT	MUT	WT	PD	13	Interesting
	6	MUT	WT	MUT	SD	1	Interesting
	7	MUT	WT	WT	CR	1	Interesting
	8	WT	MUT	WT	PD	50	Interesting
	9	WT	WT	WT	PD	146	Interesting
	10	WT	WT	WT	SD	10	Interesting
	11	WT	WT	MUT	PD	25	Interesting
	12	WT	WT	WT	PR	17	Interesting
	13	WT	MUT	WT	SD	4	Interesting
	14	WT	MUT	WT	PR	14	Interesting
	15	WT	WT	MUT	PR	4	Interesting

Showing 1 to 16 of 16 entries

Undo all changes

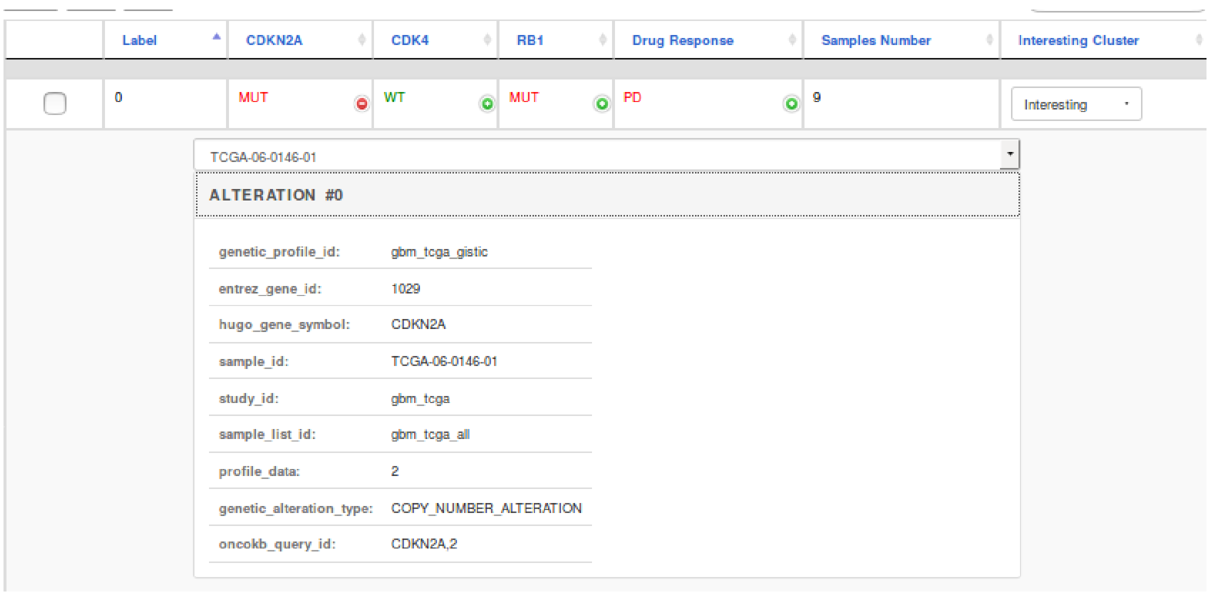
Figure 9.13: cBioPortal Downloader cluster panel (use-case).



	Label	CDKN2A	CDK4	RB1	Drug Response	Samples Number	Interesting Cluster
	0	MUT	WT	MUT	PD	9	Interesting
	1						Interesting
	2						Interesting
	3						Interesting
	4	MUT	WT	WT	SD	22	Interesting

TCGA-06-0146-01
TCGA-02-0001-01
TCGA-06-6701-01
TCGA-76-6663-01
TCGA-27-2524-01
TCGA-06-0188-01
TCGA-08-0520-01
TCGA-32-4213-01
TCGA-06-0216-01

Figure 9.14: cBioPortal Downloader cluster panel: cluster sample list (use-case).



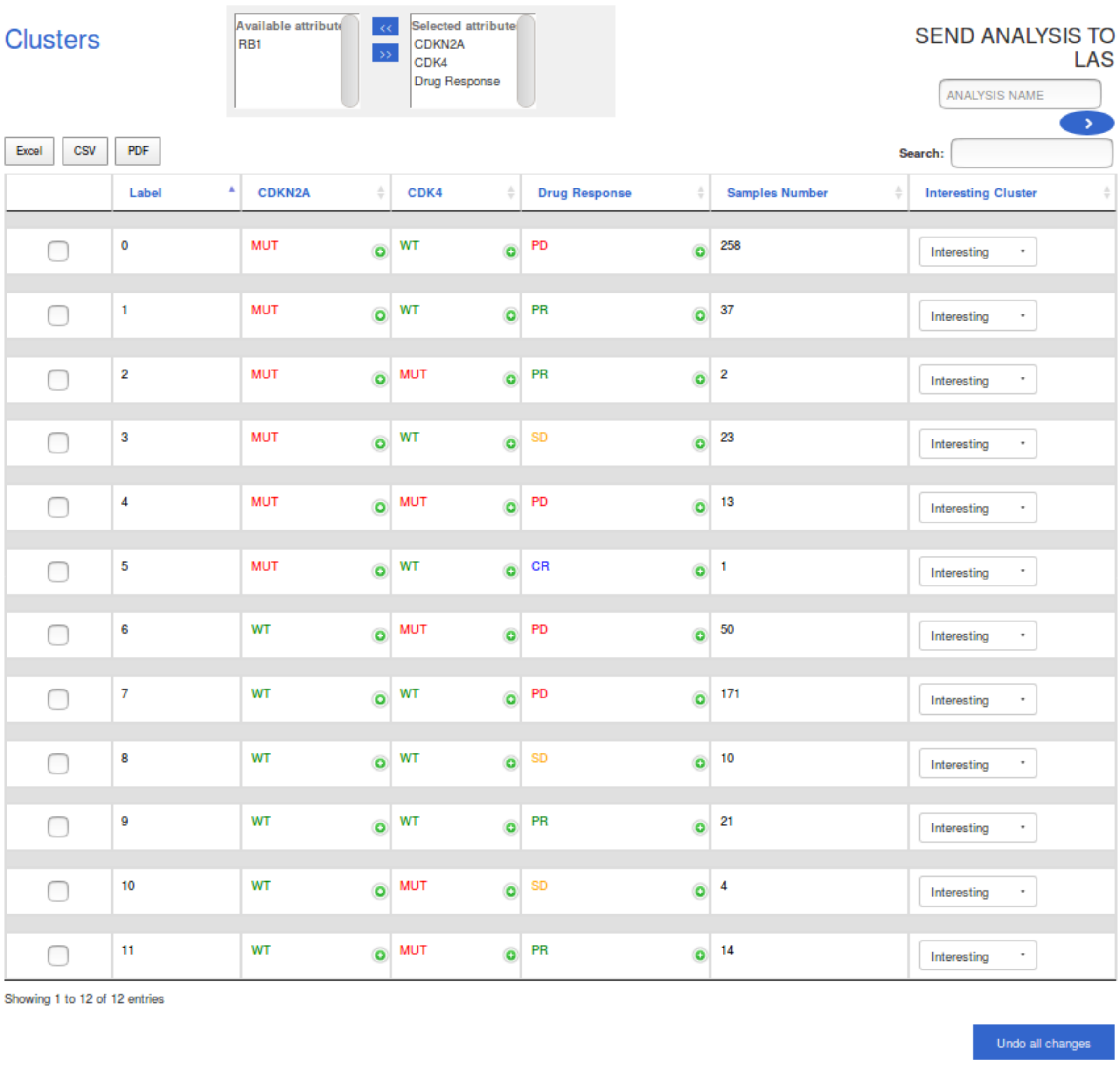
	Label	CDKN2A	CDK4	RB1	Drug Response	Samples Number	Interesting Cluster
☐	0	MUT	WT	MUT	PD	9	Interesting

TCGA-06-0146-01

ALTERATION #0

genetic_profile_id:	gbm_tcga_gistic
entrez_gene_id:	1029
hugo_gene_symbol:	CDKN2A
sample_id:	TCGA-06-0146-01
study_id:	gbm_tcga
sample_list_id:	gbm_tcga_all
profile_data:	2
genetic_alteration_type:	COPY_NUMBER_ALTERATION
oncokb_query_id:	CDKN2A,2

Figure 9.15: cBioPortal Downloader cluster panel: cluster sample alteration (use-case).



Clusters

Available attribute | RB1

Selected attribute | CDKN2A | CDK4 | Drug Response

SEND ANALYSIS TO LAS

ANALYSIS NAME

Excel | CSV | PDF

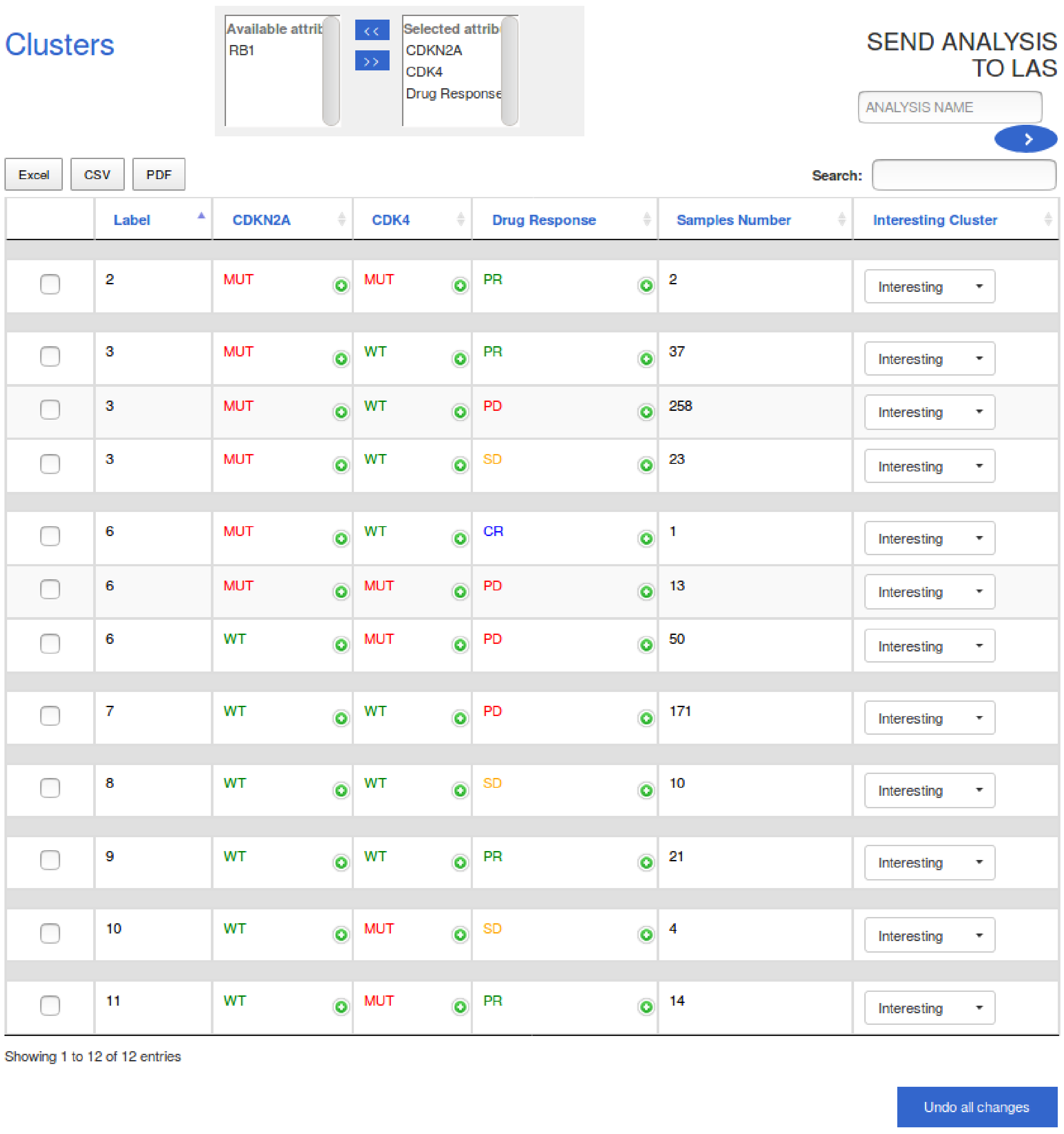
Search:

	Label	CDKN2A	CDK4	Drug Response	Samples Number	Interesting Cluster
☐	0	MUT	WT	PD	258	Interesting
☐	1	MUT	WT	PR	37	Interesting
☐	2	MUT	MUT	PR	2	Interesting
☐	3	MUT	WT	SD	23	Interesting
☐	4	MUT	MUT	PD	13	Interesting
☐	5	MUT	WT	CR	1	Interesting
☐	6	WT	MUT	PD	50	Interesting
☐	7	WT	WT	PD	171	Interesting
☐	8	WT	WT	SD	10	Interesting
☐	9	WT	WT	PR	21	Interesting
☐	10	WT	MUT	SD	4	Interesting
☐	11	WT	MUT	PR	14	Interesting

Showing 1 to 12 of 12 entries

Undo all changes

Figure 9.16: cBioPortal Downloader cluster panel: dimension removal (use-case).



	Label	CDKN2A	CDK4	Drug Response	Samples Number	Interesting Cluster
☐	2	MUT	MUT	PR	2	Interesting
☐	3	MUT	WT	PR	37	Interesting
☐	3	MUT	WT	PD	258	Interesting
☐	3	MUT	WT	SD	23	Interesting
☐	6	MUT	WT	CR	1	Interesting
☐	6	MUT	MUT	PD	13	Interesting
☐	6	WT	MUT	PD	50	Interesting
☐	7	WT	WT	PD	171	Interesting
☐	8	WT	WT	SD	10	Interesting
☐	9	WT	WT	PR	21	Interesting
☐	10	WT	MUT	SD	4	Interesting
☐	11	WT	MUT	PR	14	Interesting

Showing 1 to 12 of 12 entries

Figure 9.17: cBioPortal Downloader cluster panel: cluster merging and uninteresting cluster flagging (use-case).

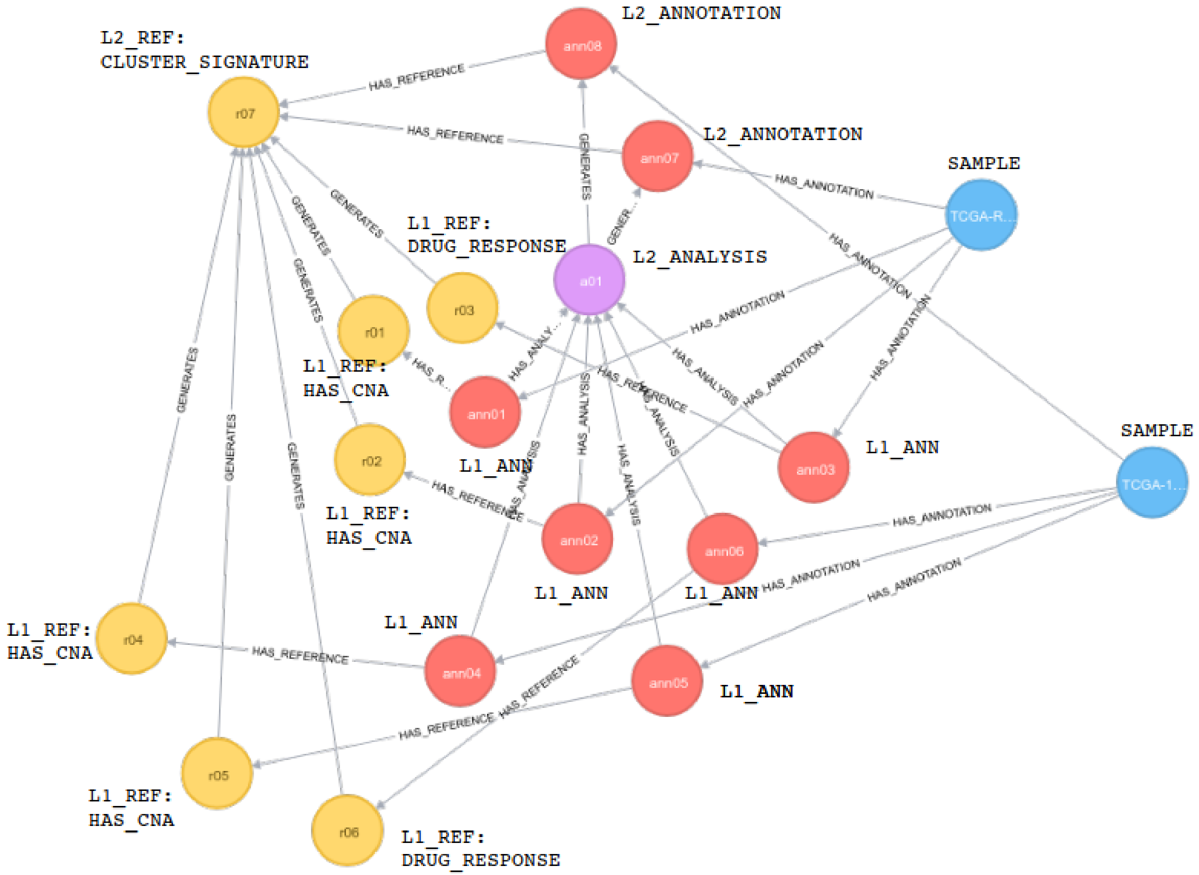


Figure 9.18: Layer 2 annotations for cluster 2 (use-case).

Part III

Conclusions

Chapter 10

Deployment and future development

Software and data integration are critical tasks, especially when applied to complex and non-standardized domain knowledge. This made the developing of this project challenging. Its goal was to design and implement a framework to integrate two platforms for cancer data storage and analysis, the cBioPortal for Cancer Genomics and the LAS platform. Specifically, it was required to (i) build a system which could allow to import LAS datasets into the cBioPortal; (ii) exploit the cBioPortal functionalities to perform visual mining and statistical computation on the imported data; (iii) aggregate the results of the cBioPortal analyses by means of a conceptual clustering process and, (iv) finally, re-import the knowledge produced through this dataflow into the LAS platform, in form of new annotations on samples. The result is a framework, implemented using a light and portable browser extension, which provides a new data aggregation and knowledge abstraction. The framework architecture is made of three components: the LAS platform, an instance of the cBioPortal, deployed by means of a Docker container, and a browser extension, named cBioPortal Downloader, which enables the communication and the data exchange among the two systems. The dataflow starts with the dataset export from the LAS system to the cBioPortal. Once the user is satisfied with the cBioPortal visual analysis, he can trigger the cBioPortal Downloader to capture and export the analysis result. Once the data reach the extension context they are used to generate a new tab made of two panels: a data panel, which allows to explore the exported data, and a cluster panel, which shows the results of the clustering process and allows to customize the built clusters. When the clusters are ready, they can be exported into the LAS platform, where their labels become new annotations on the originating samples.

As it has been already pointed out during this dissertation, the goal of this project was to develop a simple and effective framework, so its functionalities have been kept to their most basic implementation. Anyhow, it represents only the start point of a broader project and there is space for future improvements. First of all, the PouchDB local database may be synchronized with a persistent CouchDB database: this would allow to keep the history of mining sessions and use them at a later time. It would require to implement a user sign-up and log-in mechanism to associate, in a persistent database, the session data to the user which performed the query. Moreover, the LAS data export may be refined: a session identifier may be added as a study attribute in one of the files to be loaded into the cBioPortal. This would allow to trace back the studies when they come back to the LAS in form of cluster labels. Additionally, the clustering algorithm should take in consideration also continuous features of data; one possibility is to let the user divide their domain into ranges and assign them to categorical symbolic values. The cluster labeling model, based on integer labels, should be substituted by a more expressive one. One option is to label clusters according to a certain clinical evidence, such as treatment response and this would imply to build a new ontology describing biological, pharmacological and clinical knowledge modeled by the cBioPortal, and use it to label clusters; another possibility is to use semantic strings obtained by concatenating all cluster dimensions associated with their values and, in order to disambiguate the information, specifies also which attributes don't characterize each cluster. In conclusion, the new ontology may be employed to automatically generate cluster high-level annotations for un-labeled samples. So, this tool would represent the start point of a process which, from a supervised cluster analysis, would lead to an unsupervised classification of data.

Nowadays, the scientific research field claims for automatic tools which can help in performing every day laboratory activities; moreover, the need to represent complex knowledge in a simple way, so that it can be used by these instruments, has made data standardization an urgent issue. The framework which has been presented tries to meet this requests using a pluggable solution and a semantic approach to integrate data. As already stated, it is the working nucleus of a bigger project and it represents a valuable support instrument for biomedical scientists who every day investigate the cancer origins, with the aim of finding new diagnosis approaches, cures and therapies.

Bibliography

[1] Yimin Bao and Ellis Horowitz. "Integrating Through User Interface: A Flexible Integration Framework for Third-party Software". In: ().

[2] *3 Things You Need to Know About an Integration vs Interface*. URL: `http://www.lokisys.com/2015/01/integration-vs-interface/`.

[3] Lapatas et al. "Data integration in biological research: an overview". In: *Journal of Biological Research* (2015).

[4] Chris Merrick. *9 Reasons Data Warehouse Projects Fail*. URL: `https://blog.rjmetrics.com/2014/12/04/10-common-mistakes-when-building-a-data-warehouse/`.

[5] Carole Goble and Robert Stevens. "State of the nation in data integration for bioinformatics". In: *Journal of Biomedical Informatics* (Oct. 2008).

[6] *Federated database system*. URL: `https://en.wikipedia.org/wiki/Federated_database_system`.

[7] Belleau F et al. "Bio2RDF: towards a mashup to build bioinformatics knowledge systems". In: *Journal of Biomedical Informatics* (2008).

[8] *Resource Description Framework*. URL: `https://it.wikipedia.org/wiki/Resource_Description_Framework`.

[9] *RDF Schema 1.1. W3C Recommendation 25 February 2014*. URL: `http://www.w3.org/TR/rdf-schema/`.

[10] Byoung-Ha Yoon, Seon-Kyu Kim, and Seon-Young Kim. "Use of Graph Database for the Integration of Heterogeneous Biological Data". In: *Genomics & Informatics* (2017).

[11] Renzo Angles and Claudio Gutierrez. "An introduction to Graph Data Management". In: ().

[12] *Neo4j: la guida*. URL: `http://www.html.it/guide/neo4j-la-guida/`.

[13] *neo4j - DEVELOPER MANUAL*. URL: `https://neo4j.com/docs/developer-manual/current/cypher/#cypher-intro`.

[14] *AllegroGraph 6.4.0 Documentation*. URL: `https://franz.com/agraph/support/documentation/current/index.html`.

[15] *SPARQL Query Language for RDF. W3C Recommendation 15 January 2008*. URL: `https://www.w3.org/TR/rdf-sparql-query/`.

[16] *What is SPARQL?* URL: `https://ontotext.com/knowledgehub/fundamentals/what-is-sparql/`.

[17] *The MongoDB 3.6 Manual*. URL: `https://docs.mongodb.com/manual/`.

[18] Franck Michel, Catherine Faron-Zucker, and Johan Montagnat. "A Mapping-based Method to Query MongoDB Documents with SPARQL". In: ().

[19] Chandrasekaran B., Josephson JR., and Benjamins VR. “What are ontologies, and why do we need them?” In: *IEEE INTELLIGENT SYSTEMS* (1999).

[20] Robert Arp, Barry Smith, and Andrew D. Spear. *Building Ontologies with Basic Formal Ontology*.

[21] *OWL Web Ontology Language Overview. W3C Recommendation 10 February 2004*. URL: `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

[22] *Tutorial 4: Introducing RDFS & OWL*. URL: `http://www.linkeddatatools.com/introducing-rdfs-owl`.

[23] Barry Smith et al. “The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration”. In: *Nature Biotechnology* (Nov. 2007). URL: `http://dx.doi.org/10.1038/nbt1346`.

[24] *Introduction to the GO resource*. URL: `http://www.geneontology.org/page/introduction-go-resource`.

[25] *The Sequence Ontology*. URL: `http://www.sequenceontology.org/`.

[26] Bandrowski A et al. “The Ontology for Biomedical Investigations”. In: *PLoS One* (Apr. 2016).

[27] *Foundational Model of Anatomy*. URL: `http://si.washington.edu/projects/fma`.

[28] Warren A. Kibbe et al. “Disease Ontology 2015 update: an expanded and updated database of human diseases for linking biomedical knowledge through disease data”. In: *Nucleic Acids Research* (2015).

[29] Robert A. Weinberg. *The Biology of Cancer*. Garland Science, 2007.

[30] Cecie Starr. *Biologia A- I meccanismi della vita*. De Agostini Scuola, 2006.

[31] Den Dunnen et al. “HGVS recommendations for the description of sequence variants: 2016 update.” In: *Human Mutation* (2016).

[32] *Laboratory Assistant Suite*. URL: `https://las.ircc.it/las/laslogin/`.

[33] *Laboratory Assistant Suite. Technical report*. Istituto di Candiolo - IRCCS. 2005.

[34] *What is genome annotation?* URL: `https://support.ncbi.nlm.nih.gov/link/portal/28045/28049/Article/755/What-is-genome-annotation`.

[35] *COSMIC*. URL: `http://cancer.sanger.ac.uk/cosmic`.

[36] Alberto Grand. “The LAS Molecular Annotation Model”.

[37] Cerami et al. “The cBio Cancer Genomics Portal: An Open Platform for Exploring Multi-dimensional Cancer Genomics Data”. In: *Cancer Discovery* (May 2012).

[38] *cBioPortal for Cancer Genomics, FAQ*. URL: `http://www.cbioportal.org/faq.jsp`.

[39] Gao et al. “Integrative analysis of complex cancer genomics and clinical profiles using the cBioPortal”. In: *Science Signaling* (2013).

[40] *Google Chrome Extensions documentation*. URL: `https://developer.chrome.com/extensions/`.

[41] *Firefox Extensions documentation*. URL: `https://developer.mozilla.org/it/Add-ons/WebExtensions`.

[42] *cBioPortal Documentation. Data Loading*. URL: `http://cbioportal-inodb.readthedocs.io/en/latest/Data-Loading.html`.

[43] *Window.postMessage() - Web APIs | MDN*. URL: `https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage`.

[44] "New response evaluation criteria in solid tumours: Revised RECIST guideline (version 1.1)". In: *European Journal of Cancer* (2009).

[45] *PouchDB. API Reference*. URL: `https://pouchdb.com/api.html`.

[46] *Apache CouchDB 2.1 Documentation*. URL: `http://docs.couchdb.org/en/2.1.1/`.