Final Year Project Report

Full Unit – Final Report

An extendible clustering package for bioinformatics

Karan Alreja

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Susnas Sourjah

Department of Computer Science

Karan Alreja, 2025

Royal Holloway, University of London

April 15, 2025

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 10,204

Student Name: Karan Alreja

Date of Submission: 15th April 2025

Signature: Karan Alreja

Table of Contents

Abstract

This project presents the design and implementation of an extensible clustering environment for bioinformatics, aimed at reducing the steep learning curve often associated with existing bioinformatics tools. By using Java's platform independence and a healthy plugin architecture, the software allows researchers to work with multiple biological data types—such as microarray and sequence data—without requiring in-depth knowledge of underlying libraries or complex setup procedures. The system is built on a Model-View-Controller (MVC) framework and features a JavaFX-based user interface that enables users to easily select datasets, choose clustering algorithms, and view results through intuitive visualizations such as scatter plots and dendrograms.

The core of the application contains implementations of two widely used clustering algorithms—Kmeans and hierarchical clustering—which have been designed for dynamic configuration through plug-in methods. This approach develops flexibility and also provides an easy mechanism for integrating new clustering and visualization methods. The plugin framework is supported by dynamic configuration dialogs that allow each algorithm or visualization tool to specify its parameter requirements, thereby streamlining the process of tailoring the analysis to specific datasets.

In addition to its core functionality, the project provides thorough documentation describing the plugin architecture and instructions on how to extend the system. This section is designed to facilitate further development by researchers or developers who wish to add new algorithms or data processing techniques with minimal overhead.

Comprehensive testing—including unit and integration tests—shows that the system meets the functional requirements while maintain high standards of stability and performance. The project aim to move a significant step forward in providing bioinformatics researchers with a flexible, user-friendly tool for advanced data analysis, reducing the technical barriers often associated with existing software solutions.

Project Specification

1. Functional Requirements

The primary objective of this project is to develop an extensible clustering environment for bioinformatics applications. The system is required to do the following:

Data Handling:

- Work with at least two types of biological data, specifically microarray data (e.g., SOFT files) and sequence data (CSV files).
- Provide parsers that transform raw biological datasets into structured data objects that can be processed by the clustering algorithms.

Clustering Algorithms:

- Implement at least two clustering techniques: K-means clustering and hierarchical clustering.

- Allow users to configure each clustering algorithm dynamically through a user-friendly configuration dialog that accepts parameters (such as the number of clusters, distance metrics, and maximum iterations).

Visualization and Assessment:

- Support at least one visualization technique for the clustering results, with initial implementations including scatter plots for K-means and dendrograms for hierarchical clustering.
- Enable the assessment of clusters by providing clear, interpretable visual outputs and, potentially, summarizing quantitative assessments in future extensions.

Plugin Architecture:

- Integrate a flexible plugin framework that allows for new clustering algorithms, visualization methods, and parsers to be added without modifying the core system.
- Ensure that plugins define their configuration requirements (via a method like getParameters()) and expose a unified interface for execution, thus providing a consistent user experience.

User Interface:

- Provide an intuitive JavaFX-based graphical user interface (GUI) that minimizes the learning curve.
- The interface should allow users to easily select datasets, choose among available algorithms, configure parameters through interactive dialogs, and visualize results.

2. Non-Functional Requirements

The system must also meet several quality criteria:

Extensibility and Modularity:

- The software design should follow a Model-View-Controller (MVC) pattern to ensure a clean separation of concerns.
- The plugin architecture must facilitate easy addition or replacement of functional components (clustering, visualization, parsing) with minimal impact on the existing codebase.

Usability:

- The application should have a user-friendly interface that is accessible to non-specialists. Users should be able to run clustering analyses without having extensive knowledge about programming or bioinformatics libraries.
- Clear error handling, informative status messages, and dynamic configuration dialogs should provide a smooth user experience.

Performance and Stability:

- The system must be efficient in parsing, clustering, and visualizing large biological datasets.

- Error handling and complete testing (unit tests and integration tests) are required to ensure that the system remains stable under various conditions.

Maintainability:

- The code should be well-documented using Javadoc and inline comments.
- The project should employ good version control practices (e.g., using Git with proper branches, commits, and tags).

3. Technical Requirements

The technical specifications for the project include:

Programming Language:

- The system is developed in Java, which offers platform independence ("write once, run anywhere") and robust object-oriented design support.

User Interface:

- JavaFX is used for building the GUI, ensuring high-quality and responsive visualizations that are easy to update and extend.

Plugin Architecture:

- The project relies on Java's plugin technology to dynamically load external modules. This is implemented via a PluginManager that scans designated directories for JAR files, dynamically loads them into the application, and exposes them through standardized interfaces (ParserPlugin, ClusteringPlugin, VisualizationPlugin).

4. Constraints and Assumptions

- The core implementation currently focuses on microarray data (SOFT files) and CSVformatted sequence data.
- The system assumes that end users will have a Java-compatible environment (e.g., a recent version of Java and JavaFX) and minimal technical expertise.
- While the design allows for extensive future extensions (including additional clustering algorithms, additional data types, and enhanced visualization or assessment techniques), only a subset of functionalities are implemented in this initial release.

5.Deliverables

The final deliverables for the project include:

- A fully functional, extensible clustering environment that integrates multiple clustering algorithms (K-means and hierarchical) and visualization methods.
- Complete source code with a comprehensive Git repository reflecting robust software engineering practices.
- Detailed documentation including a final report, user and installation manuals, and a README file explaining the directory structure.

- A demonstration video showcasing the software in use.
- UML diagrams and supplementary materials included in the appendices of the report.

Chapter 1: Introduction

1.1 Aim and objective of the project

Bioinformatics is an interdisciplinary field at the intersection of biology and computer science. It originated from the understanding that both disciplines deal with large amounts of complex data that need systematic/algorithmic organization and analysis. In biology, data such as DNA sequences, protein structures, and gene expression profiles contain important information about living organisms, while computer science offers tools and methods to process this data efficiently. (Jeff Gauthier, 2019)

The role of computers in bioinformatics is important because of their ability to handle vast datasets, perform rapid computations, and find patterns that are otherwise difficult to find manually. By using algorithms, visualization techniques, and modeling, bioinformatics enables researchers to gain insights into biological phenomena, such as understanding genetic variations, predicting protein functions, and evaluating evolutionary relationships (Lesk, 2019). This combination of computational power and biological research forms the foundation of modern bioinformatics.

This project aims to use computational resources to develop a useful and extensible software application for clustering biological datasets. By integrating clustering algorithms, datasets, and visualization techniques within a plug-in framework, the software ensures adaptability and scalability.

This allows researchers to extend the system in vital ways, such as incorporating new datasets, visualization methods, and integrating additional clustering algorithms. Which creates use of new functionalities, providing a versatile solution to meet the ever-increasing needs of bioinformatics research.

1.2 Motivation for the Project

Bioinformatics has emerged as an important topic in modern biological research, where computational tools are crucial for analyzing complex biological datasets, such as gene expression profiles and protein sequences (Lesk, 2019). However, existing bioinformatics systems often present significant barriers to entry due to their complexity and the specialized knowledge required to use them effectively (Steiper, 2005). Researchers frequently encounter challenges with these tools, as they require advanced programming skills or specific domain expertise to customize or extend their functionalities, as highlighted in tools like **Bioconductor** (Gentleman, 2008). These barriers can limit the accessibility and usability of bioinformatics platforms for those without specialized backgrounds, slowing down the research process. (Anna Niarakis, 2022)

Prominent systems such as **GenePattern** (Reich, 2006), **Cytoscape** (Shannon, 2003) and **Galaxy** (Goecks, 2010) provide significant features for data analysis and visualization, but they frequently necessitate a high learning curve for customization and integration of new functionality. Many of these systems use complicated scripting languages or highly specialized interfaces that require users to have extensive technical knowledge. As a result, users are often restricted by the scope of the pre-existing toolsets and must invest significant time in learning the underlying frameworks.

In contrast, the proposed software aims to reduce this knowledge barrier by providing an intuitive **graphical user interface (GUI)** and an **extensible plug-in framework**. The use of a GUI will simplify user interactions, making it easier for researchers to use clustering algorithms, upload new datasets, and integrate new visualization techniques without needing to understand underlying code. Furthermore, the plug-in architecture will ensure that the software can be easily expanded in vital ways. This approach will dramatically reduce the technical hurdles and offer a more flexible, user-friendly experience, allowing researchers to customize and extend the software according to their individual needs.

This software will allow a broader audience of academics to do advanced analysis on biological data without requiring substantial programming expertise. With a focus on adaptability, scalability, and usability, the project seeks to provide a valuable tool for bioinformatics researchers, making data analysis more accessible and effective. (Spjuth, 2007)

1.3 List of Key Milestones achieved

1.3.1Term 1 Week

1-2: Project plan was worked on and submitted.

Week 3-6: Initial code was worked on setting up SOFT parser and K-means algorithm, while implementing SE principles(open/close, Strategy, Plug-in, observer for interface)

Week 7-9: Bug-fixes and rudimentary Interface was designed and presented.

Week 10-11: Work on interim report and presentation.

1.3.2 Term 2

Week 1-4: Read up on design patterns(mentioned below), plug-in architecture, best practices for bioinformatics data handling, biological databases, read up on more clustering algorithms.

Week 5-8: Implementation of all the requirements took place during this period which included but not limited to: Clustering(hierarchal), parsing(CSV), SE design principle implementation for these(open/close, observer, adapter, strategy, plug-in,)

Week 9: Final reviews and interface was drastically improved using the plug-in discovery method, described in it’s relevant section below.

Week 10-11: Work on final report and presentation.

Chapter 2: Background and Theory

In the development of bioinformatics software, several key theories and methodologies have shaped the direction of this project. Drawing from principles of object-oriented software engineering, bioinformatics, and computational biology, the following literature provides a strong foundation for the decisions made in the design and implementation of the software. I will explain their importance and how they have helped me shape this project.

2.1 Books:

Design Patterns: Elements of Reusable Object-Oriented Software (Erich Gamma, 1994)**Eclipse Plug-ins, Third Edition** (Eric Clayberg, 2008) have been important for shaping my understanding of software architecture, especially for plug-in frameworks and modularity. These books are the industry standard in their respective fields for object-oriented design patterns and how they can be applied to create reusable, flexible, and scalable software. I keep revisiting them to ensure that I am following proper Software Engineering (SE) principles. The concepts covered, such as the use of abstract classes, interfaces, and the principles of loose coupling and high cohesion, helped me learn and improve the way I make the software, ensuring maintainability and extensibility. **Eclipse Plug-ins** (Eric Clayberg, 2008) has been very important for reference on how to design a modular system that allows for easy integration of new functionalities, which is fundamental to my bioinformatics project. By frequently referring to these texts, I try to apply the best SE principles possible.

Effective Java (Bloch, 2017) has good insights into best practices for Java programming, espically in terms of object-oriented design and optimization techniques. It provides concrete guidelines for writing clean, efficient, and maintainable code, which are essential for developing robust software. I have focused on the sections concerning concerning memory management, performance optimization, and the use of common Java libraries. Bloch's emphasis on effective use of Java's features, such as collections, concurrency, and exceptions, is beneficial for my project so that it operates smoothly and efficiently. This ensures that the software is functional and maintainable.

Building Bioinformatics Solutions (Bessant, 2009) has been a big help for my understanding of bioinformatics tools and methods, which are important for my project. The book covers a wide range of topics related to bioinformatics, such as, handling biological datasets, apply clustering algorithms, and integrate databases and visualization tools. This book taught me crucial topics including the several types of datasets commonly used in bioinformatics, such as gene expression data, protein sequences, and microarray data. It also helped me understand the different algorithms commonly employed, such as K-means clustering and hierarchical clustering, which I am using in my project. Additionally, it provided a foundation for understanding the role of databases and visualization tools in bioinformatics, offering guidelines on how to integrate these components effectively. Even though I have not read every chapter, Building Bioinformatics Solutions has been essential in shaping my understanding of what bioinformatics software should do and how it can be developed to handle complex biological data efficiently.

2.2 Articles/Journals:

The Central Dogma (CRICK, 1970) established fundamental concepts in molecular biology, particularly the processes of transcription and translation, which convert DNA into RNA and then into proteins. This fundamental concept is crucial for understanding bioinformatics datasets, particularly those containing genetic and protein sequences. This study provided a conceptual framework for analyzing and interpreting biological data, such as gene expression patterns and protein sequences, which are critical in bioinformatics research, by recognizing how genetic information moves throughout a cell. This has helped me understand the various types of data utilized in bioinformatics and how they interact in biological systems

A Brief History of Bioinformatics (Jeff Gauthier, 2019) helped shape my understanding of bioinformatics, its origins, and its goals. The article discusses the development of the field, starting with the early days of computational biology and its progression into the sophisticated tools and systems used today. This historical context helped me understand bioinformatics as an interdisciplinary field, bringing together biology, computer science, and mathematics to address complex biological questions. It emphasised the importance of computational tools in processing and analyzing biological data, which laid the foundation for my approach in developing a software system that integrates these elements to facilitate bioinformatics research.

Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks (Shannon, 2003) and **Galaxy** (Goecks, 2010) offer important understandings of bioinformatics software systems. These papers describe widely used bioinformatics platforms that provide powerful features for data analysis and visualization. Cytoscape is known for visualizing molecular interaction networks, while Galaxy provides a web-based platform for data analysis and workflow management. These have guided my understanding of the types of functionalities a bioinformatics software system should have, especially when it comes to integrating data analysis with visualization techniques. Both papers highlight the importance of creating user-friendly platforms that allow researchers to work with complex biological data, which directly influences the design of my software system's interface and extensibility.

Addressing Barriers in Bioinformatics (Anna Niarakis, 2022) discusses obstacles in bioinformatics software, such as comprehensiveness, accessibility, reusability, interoperability, and reproducibility. This paper identified common barriers that users face when working with bioinformatics tools, particularly for those without much programming knowledge. This paper's findings have helped shape the design of my software, particularly in terms of making the platform more accessible, expandable, and user-friendly. Understanding the barriers stated in the study, I prioritized simplifying the user interface and providing a plug-in structure to make the software flexible to varied user needs, which matches with the goal of enhancing bioinformatics software usability.

Computational Analysis of Microarray (Quackenbush) This article provides an in-depth review of the methodologies used in the computational analysis of microarray data, with a focus on normalization techniques, clustering methods, and interpretation of gene expression profiles. The understandings gained from this work have made an impact for the design of clustering algorithms for this project, understanding it's working. This also helped in understanding the challenges inherent in analyzing high-dimensional biological data, helped in selecting and adapting K-means and hierarchical clustering algorithms to better suit the needs of bioinformatics research.

2.3 Websites:

Baeldung's article on the K-means Clustering Algorithm (Ali Dehghani, 2024)was very helpful in my implementation of K-means clustering. The article provided clear insights into how K-means works, and practical guidance was essential for implementing the algorithm correctly in the context of bioinformatics data. The article helped clarify key details about choosing the right distance metric, managing datasets, and optimizing the clustering process, all of which are critical for achieving meaningful results in biological data analysis.

2.4 Technology choices

This project is implemented in Java, using JavaFX and figma for the user interface, PlantUML for UML and class diagrams, and Git for version control.

Programming Language: Java **Why**

Java?

- **Platform Independence:** Java's "write once, run anywhere" philosophy ensures the application can work on a variety of systems without requiring significant modifications. This is essential for ensuring accessibility across diverse user environments.

- **Rich Ecosystem for Plug-in Architectures:** Java's object-oriented structure and widespread libraries (like the Java Plug-in Framework) makes it a lucrative choice for building extensible software systems, a core aspect of the project. (Shannon, 2003)
- **Community Support and Reliability**: Java has a mature ecosystem, extensive documentation, and robust community support. Its proven stability in enterprise and academic projects makes it a reliable choice for bioinformatics applications (TIOBE, n.d.)
- **Integration with Modern UI and Testing Frameworks:** Java integrates seamlessly with JavaFX to build rich, responsive user interfaces. Tools like TestFX further enhance the testing process by providing a framework for automating JavaFX interface tests—ensuring that the GUI is both robust and user-friendly.

Comparison to Alternatives:

- **Python:** While Python is widely used in bioinformatics due to libraries like Biopython, it is interpreted, which can lead to slower execution times for large-scale or real-time applications. Java, being compiled, offers better performance for computation-heavy tasks.
- **C++:** Though faster in execution, C++ has a steeper learning curve and lacks the built-in support for GUI development and cross-platform functionality provided by Java. *User Interface Design: JavaFX / Figma* **Why JavaFX?**

- **Integration with Java:** JavaFX is native to Java, making it easier to develop cohesive applications where the GUI and backend work seamlessly.
- **Rich GUI Capabilities:** JavaFX supports advanced visualizations, animations, and a wide range of controls, which are ideal for presenting complex data such as biological clustering outputs.
- **Ease of Customization:** It allows for the creation of modern, dynamic interfaces, essential for usability in bioinformatics tools.

Why Figma?

- Figma can be used for prototyping and designing the user interface before implementation, ensuring a user-centered approach. This reduces the risk of usability issues during the development phase.

Comparison to Alternatives:

- **Tkinter (Python):** Although simpler, Tkinter lacks the sophistication needed for modern GUIs and isn't as suitable for high-quality, data-intensive visualizations.
- **Swing (Java):** While Swing is another Java-based option, it is considered outdated compared to JavaFX, which offers better functionality and design flexibility.

Why PlantUML?

- **Text-Based and Lightweight:** Allows you to create, modify, and version UML diagrams via simple text files; Easily integrates with version control systems for tracking design evolution.

- **Versatility in Diagram Types:** Supports a wide range of UML diagrams (class, sequence, activity, etc.). Exports to formats such as PNG, SVG, and PDF for inclusion in reports.
- **Alignment with Software Engineering Principles:** Facilitates quick iteration and collaborative design, which is crucial for a modular, plugin-based project. (Citations: PlantUML Documentation, 2020; Gamma et al., 1994) **Comparison to Alternatives:**
- **Open Source and Community-Driven:** Benefits from community support and frequent updates. Compared to graphical tools like Visio or Enterprise Architect, PlantUML is more agile and cost-effective.

Why Git?

- **Distributed and Flexible:** Each developer works with a complete local repository, allowing for offline work and safe experimentation. Supports robust branching and merging strategies to manage complex plugin integrations.
- **Facilitates Collaboration:** Integrates seamlessly with platforms such as GitHub and GitLab for code reviews, continuous integration, and collaborative development. Enables thorough tracking of changes and historical versions of the codebase
- **Widely Adopted and Reliable:** Industry-standard tool with extensive documentation, supporting smooth conflict resolution. Provides a strong foundation for maintaining clean, well-structured code throughout the project. (Chacon &

Straub, 2014; Bird et al., 2009) **Comparison to alternatives:**

- Git is a distributed system offering unmatched flexibility and performance with powerful branching/merging, robust community support, and seamless integration with modern tools—features that set it apart from centralized alternatives like Subversion or CVS.

Chapter 3: Implementation so far

This section is organized into several subsections, each describing a key component of the project's development. We begin by describing the parsing of biological datasets, including our implementations of the SoftParser and CSVParser. Next, we discuss the implementation of clustering algorithms, with a focus on the K-Means and Hierarchical clustering methods. Following that, we detail the plugin architecture and how dynamic configuration is integrated to facilitate extensibility. The subsequent subsection covers the design and implementation of the user interface using JavaFX, while the testing and validation section outlines our methodologies for ensuring robust performance and error handling. Finally, we present an overview of our code quality measures and version control practices using Git.

All the UML pictures code snippet will be provided in an Appendix for easier rendering.

3.1 Parsing Layer

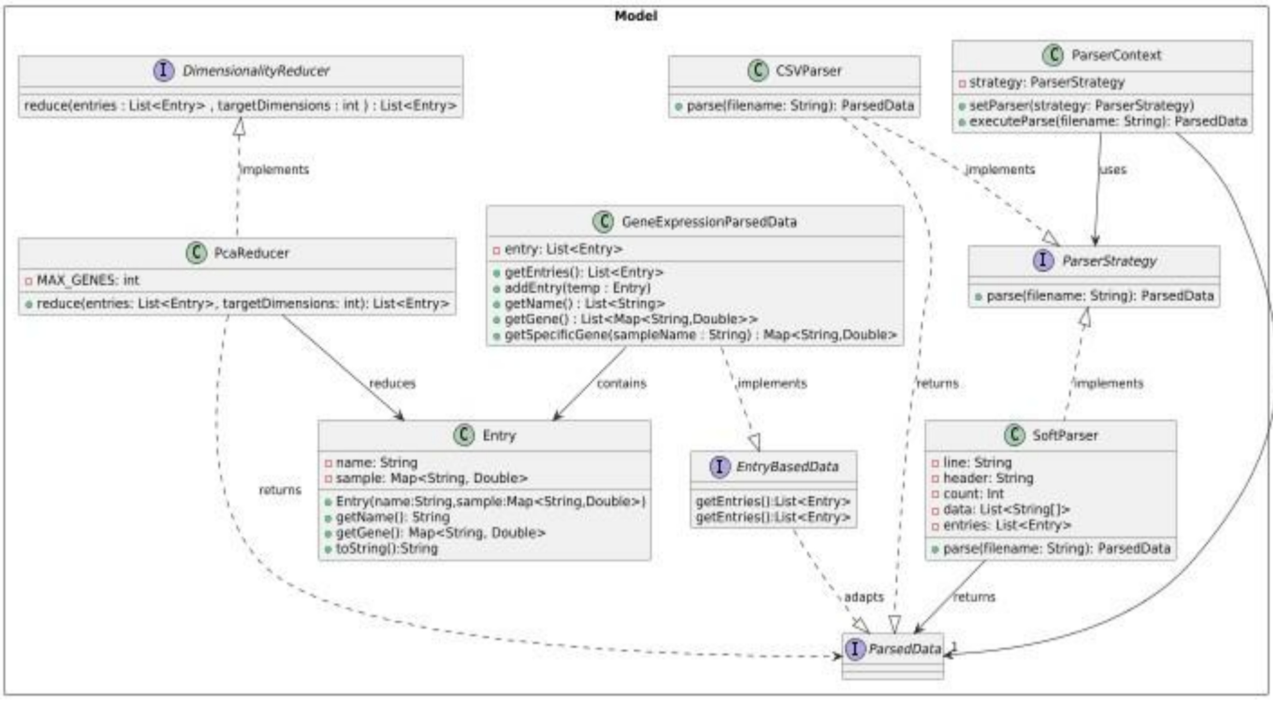

Model
DimensionalityReducer
reduce(entries : List<Entry> , targetDimensions : int) : List<Entry>
implements
PcaReducer
MAX_GENES: int
reduce(entries: List<Entry>, targetDimensions: int): List<Entry>
CSVParser
parse(filename: String): ParsedData
ParserContext
strategy: ParserStrategy
setParser(strategy: ParserStrategy)
executeParse(filename: String): ParsedData
implements
uses
ParserStrategy
parse(filename: String): ParsedData
GeneExpressionParsedData
entry: List<Entry>
getEntries(): List<Entry>
addEntry(temp : Entry)
getName() : List<String>
getGene() : List<Map<String,Double>>
getSpecificGene(sampleName : String) : Map<String,Double>
reduces
contains
implements
returns
implements
returns
Entry
name: String
sample: Map<String, Double>
Entry(name:String,sample:Map<String,Double>)
getName(): String
getGene(): Map<String, Double>
toString():String
EntryBasedData
getEntries():List<Entry>
getEntries():List<Entry>
SoftParser
line: String
header: String
count: int
data: List<String[]>
entries: List<Entry>
parse(filename: String): ParsedData
returns
adapts
returns
ParsedData
1

Figure 1. This shows the relationship between the classes for our parsing layer.

To ensure that our parsing algorithms are interchangeable at runtime, we have encapsulated them using the Strategy design pattern. This approach allows us to dynamically select and swap the parsing method—whether it's SoftParser for SOFT files or CSVParser for CSV files—through the ParserContext; classes that need to get their data parsed need only call this class, this class holds references to concrete subclasses. Each parser implementation adheres to the ParserStrategy interface and returns a ParsedData object this is a general interface that allows for polymorphism for any data that is parsed, with our current focus on GeneExpressionParsedData via an adapter interface (EntryBasedData). This design promotes polymorphism and a highly general architecture, enabling us to add new parsing methods for other biological data types with minimal changes; any new parser simply needs to extend ParsedData to let the program know that the specific class is a kind of

ParsdData for classes further upstream to pick it up, these classes can also implement their own adapter interface if they need(recommended for different kinds of biological data). We also illustrate these design choices using UML and class diagrams for clarity in further subsections.

3.1.1 SoftParser

The SoftParser was developed to address the task of parsing SOFT files from the NCBI GEO repository, changing them into a structured internal format suitable for clustering analysis. To achieve this, I used the Strategy design pattern—SoftParser implements the ParserStrategy interface, making it interchangeable with other parser implementations like CSVParser. The ParserContext class serves as the strategy context, dynamically selecting and invoking SoftParser based on the file type during runtime keeping the system modular and extensible.

In terms of code logic, SoftParser first reads the header to capture metadata and then iterates through each subsequent line to extract sample identifiers and gene expression values. Each data row is converted into an Entry object, and these entries are aggregated into a GeneExpressionParsedData instance. This encapsulation ensures that downstream components, such as the clustering algorithms, have a consistent and optimized data structure to work with.

The need for SoftParser arose from the complexity and size of SOFT files, which are rich in hierarchical biological data. Supporting classes like Entry and GeneExpressionParsedData play a critical role by representing individual samples and collections of samples respectively. This modular design improves maintainability and reusability while it also allows for easy extension: if new file formats or parsing strategies are required in the future, they can be integrated with minimal changes to the core system.

3.1.2 CSVparser

The CSVParser was made as an alternative parser to handle CSV-formatted biological datasets, complementing the SoftParser for SOFT files. Like SoftParser, CSVParser implements the ParserStrategy interface, enabling it to be easily integrated into the ParserContext. This adherence to the Strategy pattern allows the system to select the appropriate parser based on the file type, ensuring flexibility and modularity.

The design of CSVParser is centered on reading a CSV file where the first row is used as the header, identifying column names with the first column representing the sample identifier. Each row is then processed by splitting the data on commas, converting the values into numerical gene expression data, and mapping them to their corresponding gene names. Each processed row is encapsulated into an Entry object, and these entries are aggregated within a GeneExpressionParsedData instance, providing a consistent internal representation for downstream clustering and analysis.

Error handling in CSVParser is implemented to manage typical CSV issues such as column count mismatches and improper numerical formatting. This ensures that only valid data is parsed and fed into the clustering pipeline. Supporting classes, particularly Entry and GeneExpressionParsedData, are reused to maintain uniformity in data representation across different parsers, which aids in achieving a robust and maintainable architecture.

Overall, CSVParser's integration into the system demonstrates a flexible approach to handling multiple data formats, reinforcing the project's commitment to extensibility and modular design in bioinformatics software.

3.1.3 Dimensionality Reducer

The Principal Component Analysis (PCA) dimensionality reducer plays a important role in rationalizing high-dimensional biological data by transforming gene expression profiles into a lower-dimensional space. Operating directly on ParsedData objects, the PCA reducer processes the list of Entry instances to compute a set of principal components that capture the maximum variance in the data. By reducing the dimensionality, PCA not only speeds up subsequent clustering operations by simplifying data complexity but also improves visualization by presenting clusters in a more interpretable form. This modular component fits naturally within our software's design— thanks to the use of common interfaces—and allows for easy substitution or extension. It ensures that even if new types of biological data are introduced, the same efficient, consistent method is applied to prepare the data for analysis.

3.2 Clustering Layer

The clustering layer is designed to analyze parsed biological data(ParsedData) by grouping similar data points using clustering algorithms. At its core, this layer relies on the **Strategy design pattern**—implemented via the `ClusterStrategy` interface—which defines a standard method `fit(data, config)` that returns a `ClusteredData` object(later used by the views to display data). This abstraction allows different clustering algorithms (like K-means and Hierarchical clustering) to be interchanged dynamically without affecting the rest of the system. Any clustering algorithm needs to return a ClussteredData Object so that upstream analysis can be done by any view that can process this ClusteredData. Each algorithm also provides a `getParameters()` method, which exposes default configuration values and required settings, enabling dynamic dialogs to collect user-specific adjustments.

The **K-means algorithm** uses this pattern by generating random centroids, assigning data points to the nearest centroid based on Euclidean distance, and iteratively updating the centroids until convergence. It packages its results into a `FlatClusteredData` instance, conforming to the `ClusteredData` interface so that upstream visualization components can interact with the results uniformly.

Hierarchical clustering operates on a similar foundation but uses a recursive merging process. It constructs an evolving tree structure from individual data points via intermediate `ClusterNode` objects and ultimately produces a `HierarchicalClusteredData` object, which can be visualized, for example, as a dendrogram.

Furthermore, the **Observer pattern** is used by the `ClusterContext`, which manages the clustering strategy and notifies registered observers (such as visualization classes) once new clustering data becomes available. This ensures that any changes in clustering output automatically propagate to the user interface.

Together, these components—through clearly defined interfaces, dynamic configuration via `getParameters()`, and a consistent output defined by `ClusteredData`—form a modular, extensible clustering layer that meets the demands of diverse bioinformatics datasets and supports future algorithm integration

3.2.1 K-means Clustering

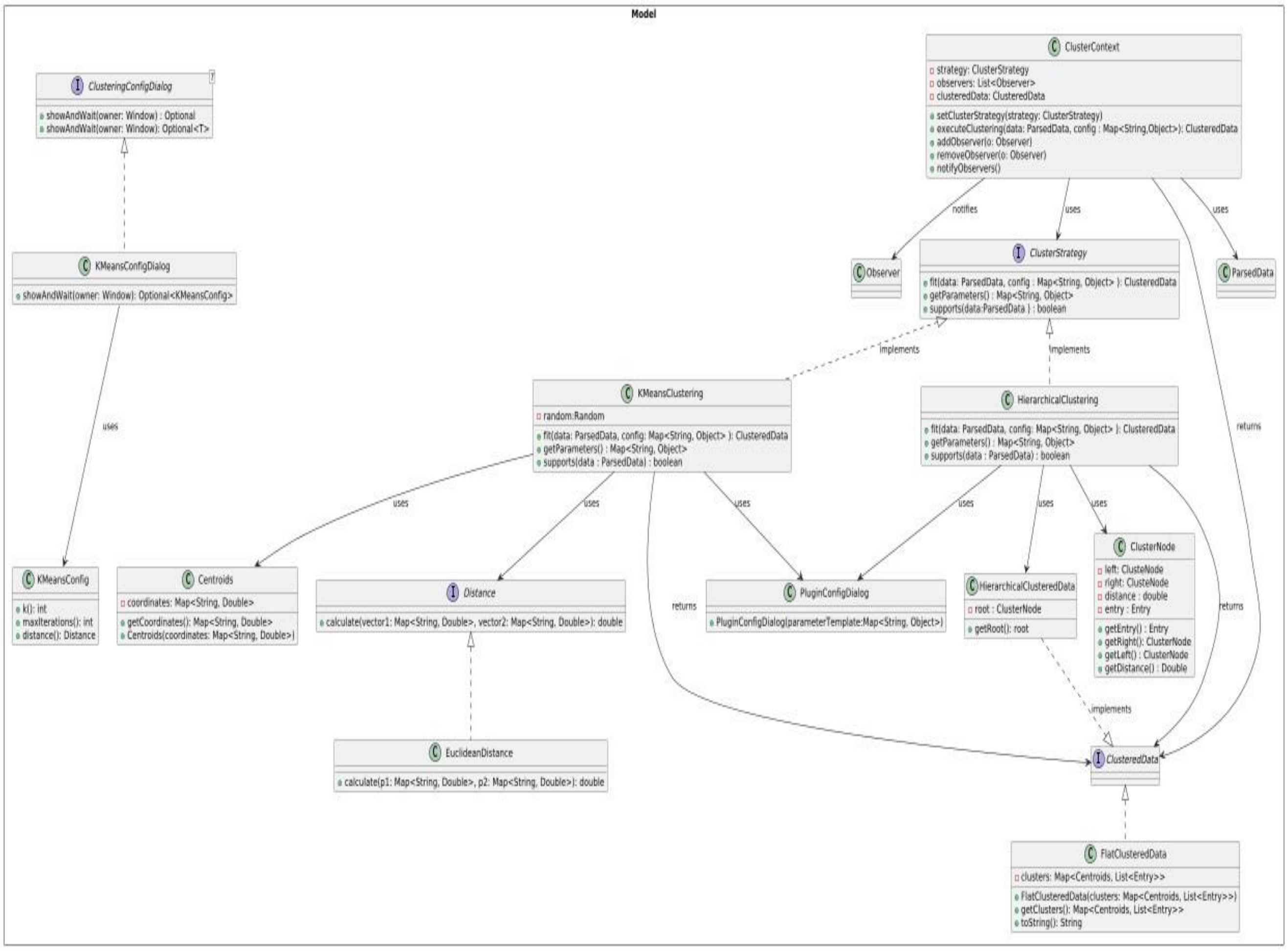

Model
ClusteringConfigDialog
showAndWait(owner: Window) : Optional
showAndWait(owner: Window): Optional<T>
KMeansConfigDialog
showAndWait(owner: Window): Optional<KMeansConfig>
uses
ClusterContext
strategy: ClusterStrategy
observers: List<Observer>
clusteredData: ClusteredData
setClusterStrategy(strategy: ClusterStrategy)
executeClustering(data: ParsedData, config : Map<String,Object>): ClusteredData
addObserver(o: Observer)
removeObserver(o: Observer)
notifyObservers()
notifies
uses
uses
Observer
ClusterStrategy
fit(data: ParsedData, config : Map<String, Object>): ClusteredData
getParameters() : Map<String, Object>
supports(data:ParsedData) : boolean
ParsedData
implements
implements
KMeansClustering
random:Random
fit(data: ParsedData, config: Map<String, Object>): ClusteredData
getParameters() : Map<String, Object>
supports(data : ParsedData) : boolean
HierarchicalClustering
fit(data: ParsedData, config: Map<String, Object>): ClusteredData
getParameters() : Map<String, Object>
supports(data : ParsedData) : boolean
returns
uses
uses
uses
uses
uses
uses
KMeansConfig
k(): int
maxIterations(): int
distance(): Distance
Centroids
coordinates: Map<String, Double>
getCoordinates(): Map<String, Double>
Centroids(coordinates: Map<String, Double>)
Distance
calculate(vector1: Map<String, Double>, vector2: Map<String, Double>): double
returns
PluginConfigDialog
PluginConfigDialog(parameterTemplate:Map<String, Object>)
HierarchicalClusteredData
root : ClusterNode
getRoot(): root
ClusterNode
left: ClusteNode
right: ClusteNode
distance : double
entry : Entry
getEntry() : Entry
getRight(): ClusterNode
getLeft() : ClusterNode
getDistance() : Double
returns
EuclideanDistance
calculate(p1: Map<String, Double>, p2: Map<String, Double>): double
implements
ClusteredData
FlatClusteredData
clusters: Map<Centroids, List<Entry>>
FlatClusteredData(clusters: Map<Centroids, List<Entry>>)
getClusters(): Map<Centroids, List<Entry>>
toString(): String

Figure 2. Shows the relationship between Clustering Algorithms

The K-means clustering implementation is built following the Strategy design pattern, summarized in the KMeansClustering class which implements the ClusterStrategy interface. This interface defines a standard method, fit(data, config), that enables the algorithm to be configured at runtime via a configuration map. The configuration parameters—such as the number of clusters (k), the maximum number of iterations, and the chosen distance metric—are exposed through the getParameters() method. These parameters are then dynamically collected using the configuration dialog (KMeansConfigDialog).

Supporting Classes and Flow:

- **Entry and GeneExpressionParsedData:** The data parsed from biological datasets (e.g., from SOFT or CSV files) is structured as a list of Entry objects. Each Entry encapsulates a sample and its gene expression values. This data is aggregated within a GeneExpressionParsedData instance, matching to the ParsedData interface, to maintain a consistent data model.

- **Distance Calculation:** The algorithm calculates the Euclidean distance between each data point and the cluster centroids using the EuclideanDistance class (implementing the Distance interface), an essential operation for grouping data points effectively.
- **Centroid Management:** Random centroids are initially generated. Data entries are then assigned to the nearest centroid, and new centroids are computed as the mean of the entries assigned to each cluster. This iteration continues until convergence.
- **Output Format:** Once the algorithm stabilizes, the final clustering results are packaged as a FlatClusteredData object, which implements the ClusteredData interface. This uniform output enables integration with visualization components in the system.

3.2.2 Hierarchical Clustering

The Hierarchical Clustering implementation uses a recursive, agglomerative approach to group data. This method is implemented in the HierarchicalClustering class which also follows the Strategy pattern by implementing the ClusterStrategy interface. It similarly accepts a configuration map through the fit(data, config) method; parameters required by the hierarchical algorithm (such as the distance metric) are exposed via the getParameters() method.

Supporting Classes and Flow:

- **ClusterNode:** Hierarchical clustering builds its cluster structure by creating individual ClusterNode objects for each Entry in the dataset. These nodes represent the leaves of a clustering tree and are recursively merged based on the minimum pairwise distance.
- **Distance Calculation:** Like K-means, this algorithm uses the EuclideanDistance class (through the Distance interface) to compute distances between nodes, ensuring that the merge decisions reflect the true similarity between data points.
- **HierarchicalClusteredData:** The merged structure is captured in a HierarchicalClusteredData object, which typically holds a reference to the root ClusterNode of the dendrogram. This structure enables visualization with components such as the DendrogramView.
- **Dynamic Configuration:** The implementation supports dynamic runtime configuration by retrieving settings via its getParameters() method. This offers flexibility in adjusting how clusters are merged or how the dendrogram is rendered.

3.3 View layer and introduction to UI

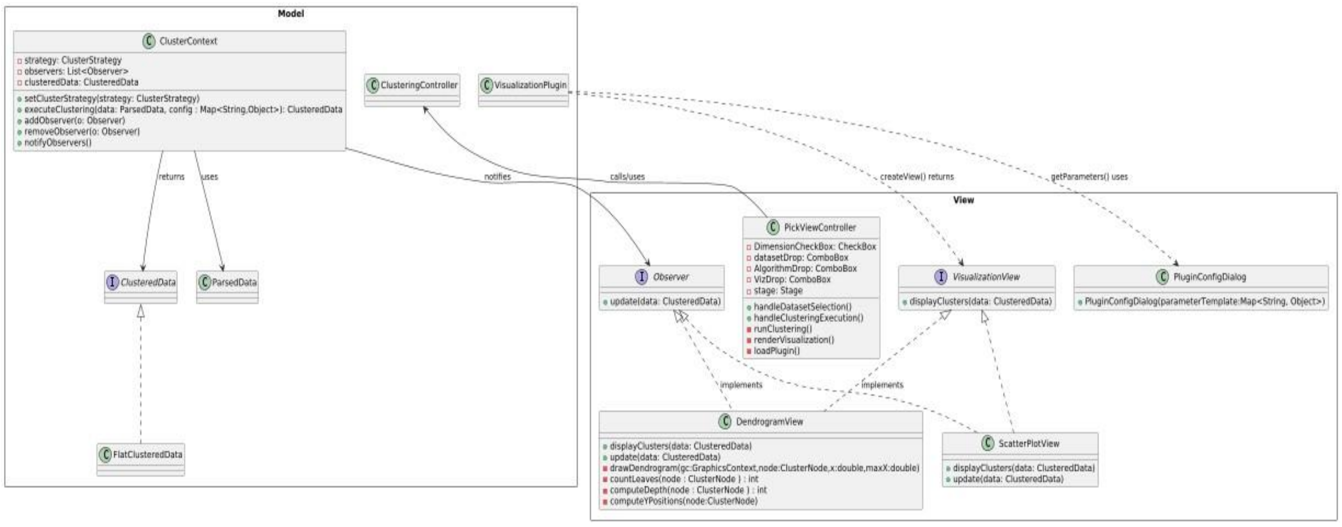

Model
ClusterContext
strategy: ClusterStrategy
observers: List<Observer>
clusteredData: ClusteredData
setClusterStrategy(strategy: ClusterStrategy)
executeClustering(data: ParsedData, config : Map<String,Object>): ClusteredData
addObserver(o: Observer)
removeObserver(o: Observer)
notifyObservers()
ClusteringController
VisualizationPlugin
returns
uses
notifies
calls/uses
createView() returns
getParameters() uses
ClusteredData
ParsedData
FlatClusteredData
View
PickViewController
DimensionCheckBox: CheckBox
datasetDrop: ComboBox
AlgorithmDrop: ComboBox
VizDrop: ComboBox
stage: Stage
handleDatasetSelection()
handleClusteringExecution()
runClustering()
renderVisualization()
loadPlugin()
Observer
update(data: ClusteredData)
VisualizationView
displayClusters(data: ClusteredData)
PluginConfigDialog
PluginConfigDialog(parameterTemplate:Map<String, Object>)
implements
implements
DendrogramView
displayClusters(data: ClusteredData)
update(data: ClusteredData)
drawDendrogram(gc:GraphicsContext,node:ClusterNode,x:double,maxX:double)
countLeaves(node : ClusterNode) : int
computeDepth(node : ClusterNode) : int
computeYPositions(node:ClusterNode)
ScatterPlotView
displayClusters(data: ClusteredData)
update(data: ClusteredData)

Figure 3. Shows the flow of information between model and view

The view layer is responsible for presenting the clustering results to the user in an intuitive and responsive manner. To achieve this, the system uses both the Strategy and Observer design patterns. The view layer comprises two concrete visualization classes: **ScatterPlotView** and **DendrogramView**. These classes implement the VisualizationView interface, which defines the method to display clusters, and the Observer interface, which provides an update method that gets triggered whenever clustering results are refreshed.

- **Strategy Pattern in the View:** Although the Strategy pattern is mainly used within the parsing and clustering layers, its principles are much useful in the view layer by allowing flexible and interchangeable visualization components. For example, both ScatterPlotView and DendrogramView adhere to a common interface, meaning they can be dynamically chosen based on user preference or data type without altering the overall application logic.

- **Observer Pattern for Dynamic Updates:** The view components are registered as observers within the ClusterContext. When the clustering process completes, ClusterContext notifies all subscribed observers by calling their update methods. This decoupling ensures that every time new clustering data is produced, the active visualization components (be it a scatter plot or a dendrogram) are automatically updated, maintaining consistency between the data and the displayed information.

Overall, by utilizing these design patterns, the view layer remains modular and adaptable. The separation via interfaces promotes cleaner code and enables easy integration of new visualization methods as plugins in future extensions. The use of JavaFX further enhances user interaction, ensuring that all visualizations are rendered in a responsive and user-friendly manner.

3.3.1 ScatterPlotView

ScatterPlotView is implemented as a JavaFX VBox containing a ScatterChart that visualizes Kmeans clustering results. When new clustering data is available, the update() method (from the Observer interface) calls displayClusters(), which clears previous data and iterates through each cluster in a FlatClusteredData instance. For each cluster, it creates a new series using two selected gene expression values (that we get from the PCA reduction) as x and y coordinates. The series are then added to the scatter chart to render the clusters in real time.

- **displayClusters(data: ClusteredData):**

- Clears existing chart data.

- Iterates over clusters (from FlatClusteredData), creating an XYChart.Series for each cluster using the first two gene expression values as x and y coordinates.
- Adds these series to the chart for display.
- **update(data: ClusteredData):** • Simply calls displayClusters(data) to ensure that whenever new clustering data is received, the view is refreshed

3.3.2 DendrogramView

DendrogramView extends JavaFX's ScrollPane to accommodate large hierarchical data displays. It contains a Canvas within a Pane that is used to draw dendrograms for hierarchical clustering. The displayClusters() method first validates that the input is HierarchicalClusteredData, then retrieves the root ClusterNode. It computes the tree's depth and the number of leaves to dynamically adjust the canvas dimensions. The computeYPositions() method recursively calculates vertical positions for each node, and drawDendrogram() uses GraphicsContext to draw connecting lines and labels, resulting in a scrollable dendrogram visualization.

- **computeYPositions(node: ClusterNode):**
- Recursively traverses the cluster tree, assigning y-coordinates to each node.
- For leaf nodes, it uses a fixed spacing, while internal nodes receive the average ycoordinate of their children.

- **drawDendrogram(gc: GraphicsContext, node: ClusterNode, x: double, maxX: double):**
- Draws horizontal and vertical lines using strokeLine to connect nodes, visually depicting the clustering hierarchy.
- For leaf nodes, displays labels (using fillText) with the sample's name, and then recurses on child nodes.

- **update(data: ClusteredData):**
- Calls displayClusters(data) to refresh the dendrogram whenever new clustering data is provided by the clustering layer.

3.4 Controller layer

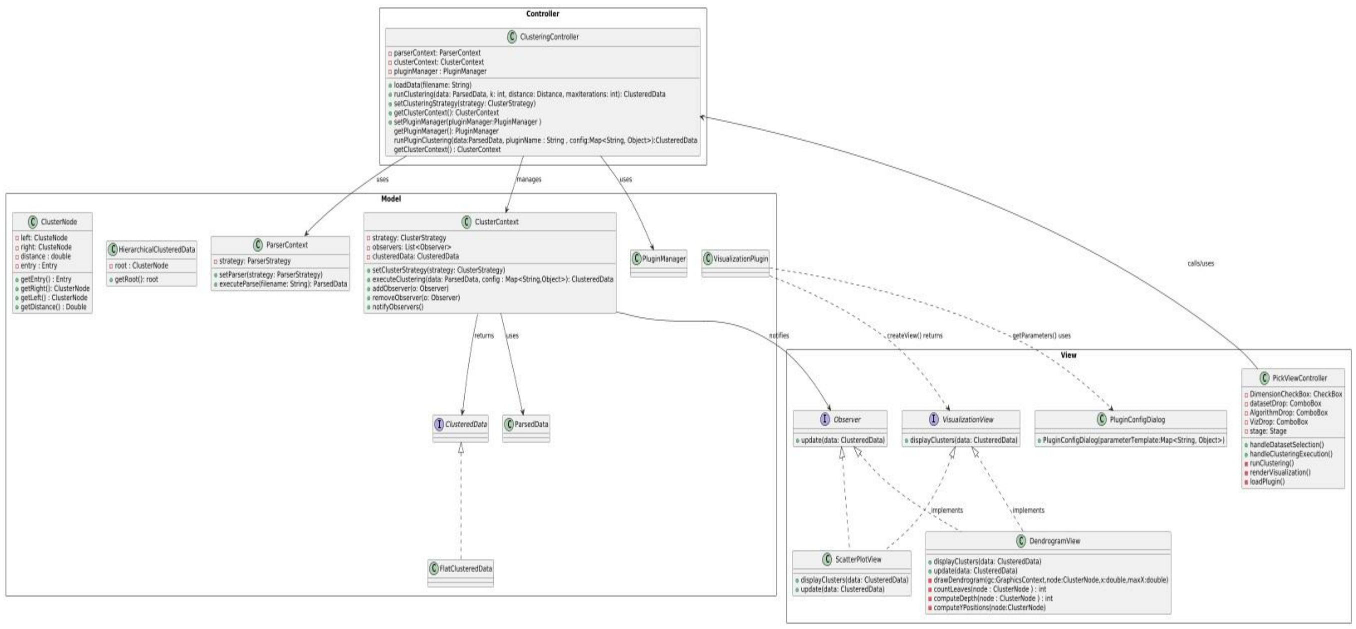

Controller
ClusteringController
parserContext: ParserContext
clusterContext: ClusterContext
pluginManager : PluginManager
loadData(filename: String)
runClustering(data: ParsedData, k: int, distance: Distance, maxIterations: int): ClusteredData
setClusteringStrategy(strategy: ClusterStrategy)
getClusterContext(): ClusterContext
setPluginManager(pluginManager:PluginManager)
getPluginManager(): PluginManager
runPluginClustering(data:ParsedData, pluginName : String , config:Map<String, Object>):ClusteredData
getClusterContext() : ClusterContext
uses
manages
uses
Model
ClusterNode
left: ClusteNode
right: ClusteNode
distance : double
entry : Entry
getEntry() : Entry
getRight(): ClusterNode
getLeft() : ClusterNode
getDistance() : Double
HierarchicalClusteredData
root : ClusterNode
getRoot(): root
ParserContext
strategy: ParserStrategy
setParser(strategy: ParserStrategy)
executeParse(filename: String): ParsedData
ClusterContext
strategy: ClusterStrategy
observers: List<Observer>
clusteredData: ClusteredData
setClusterStrategy(strategy: ClusterStrategy)
executeClustering(data: ParsedData, config : Map<String,Object>): ClusteredData
addObserver(o: Observer)
removeObserver(o: Observer)
notifyObservers()
PluginManager
VisualizationPlugin
returns
uses
ClusteredData
ParsedData
FlatClusteredData
notifies
createView() returns
getParameters() uses
calls/uses
View
Observer
update(data: ClusteredData)
VisualizationView
displayClusters(data: ClusteredData)
PluginConfigDialog
PluginConfigDialog(parameterTemplate:Map<String, Object>)
PickViewController
DimensionCheckBox: CheckBox
datasetDrop: ComboBox
AlgorithmDrop: ComboBox
VizDrop: ComboBox
stage: Stage
handleDatasetSelection()
handleClusteringExecution()
runClustering()
renderVisualization()
loadPlugin()
implements
implements
ScatterPlotView
displayClusters(data: ClusteredData)
update(data: ClusteredData)
DendrogramView
displayClusters(data: ClusteredData)
update(data: ClusteredData)
drawDendrogram(gc:GraphicsContext,node:ClusterNode,x:double,maxX:double)
countLeaves(node : ClusterNode) : int
computeDepth(node : ClusterNode) : int
computeYPositions(node:ClusterNode)

Figure 4. This shows the interaction between the controller and the View layer.

ClusteringController serves as the central hub in the Controller layer, binding the view and the model. It uses the Strategy design pattern by delegating parsing to the ParserContext and clustering to the ClusterContext. Additionally, it ensures dynamic plugin integration via the PluginManager, enabling the system to switch clustering algorithms or run plugin-based clustering without altering the rest of the code. Overall, ClusteringController abstracts data loading, clustering execution, and plugin handling, ensuring that the user interface can trigger these functions through simple method calls.

- **loadData(File filename):** Checks the file extension and sets the appropriate parser (e.g., SoftParser).
- Invokes ParserContext to execute parsing and returns a ParsedData object, ensuring a uniform data model for upstream processing.

- **runClustering(ParsedDatadata,Map<String,Object>config):**
- Delegates the clustering operation to the ClusterContext by invoking its executeClusterin gmethod.
- Uses the configuration map to parameterize the clustering algorithm, ensuring flexibility in algorithm execution.

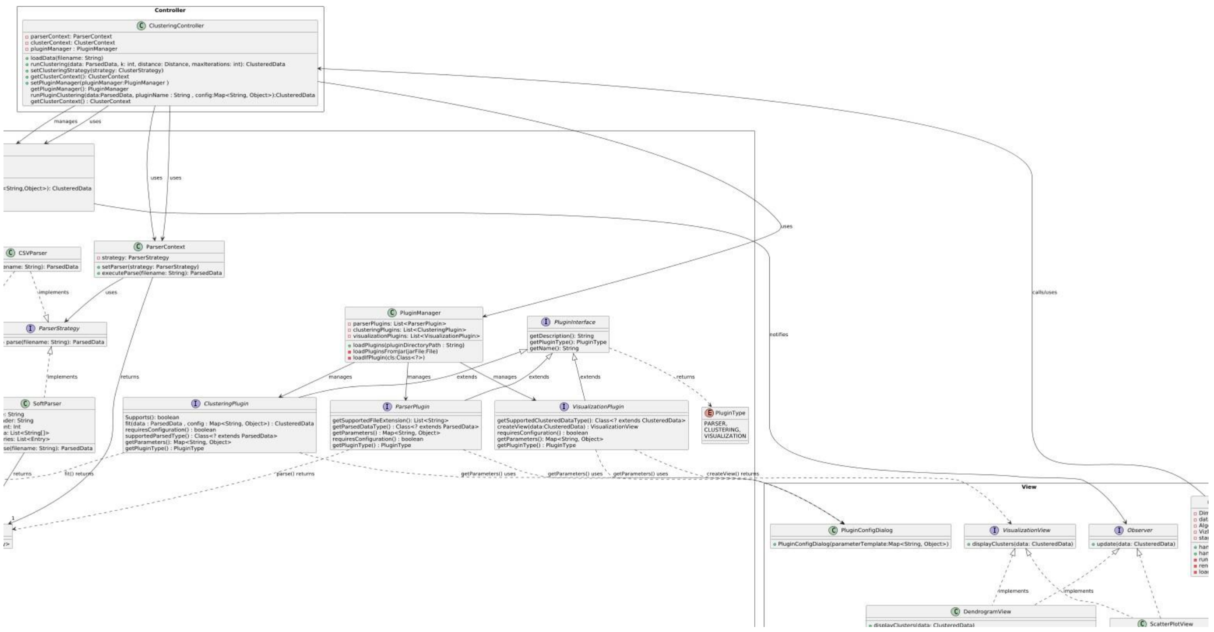
- **runPluginClustering(ParsedData data, String pluginName, Map<String, Object> config):**
•Uses the PluginManager to look up a ClusteringPlugin by name.
- Calls the plugin's fit method with the provided data and configuration, enabling dynamic extension of clustering capabilities.

- **setClusteringStrategy(ClusterStrategy strategy):**
- Sets the current clustering algorithm in the ClusterContext, adhering to the Strategy pattern for flexibility and modularity.

- **setPluginManager(PluginManager pluginManager) & getPluginManager():** • Provides access to the PluginManager, supporting the retrieval and management of plugins which can be dynamically integrated into the clustering process.

This design ensures that the controller efficiently coordinates between the UI and the underlying parsing and clustering layers, using established design patterns to maintain a robust, extendable architecture.

3.5 Plugin Architecture


Controller
ClusteringController
parserContext: ParserContext
clusterContext: ClusterContext
pluginManager : PluginManager
loadData(filename: String)
runClustering(data: ParsedData, k: int, distance: Distance, maxIterations: int): ClusteredData
setClusteringStrategy(strategy: ClusterStrategy)
getClusterContext(): ClusterContext
setPluginManager(pluginManager:PluginManager)
getPluginManager(): PluginManager
runPluginClustering(data:ParsedData, pluginName : String , config:Map<String, Object>):ClusteredData
getClusterContext() : ClusterContext
manages
uses
CSVParser
ParserContext
strategy: ParserStrategy
setParser(strategy: ParserStrategy)
executeParse(filename: String): ParsedData
implements
ParserStrategy
parse(filename: String): ParsedData
SoftParser
PluginManager
parserPlugins: List<ParserPlugin>
clusteringPlugins: List<ClusteringPlugin>
visualizationPlugins: List<VisualizationPlugin>
loadPlugins(pluginDirectoryPath : String)
loadPluginsFromJar(jarFile:File)
loadIfPlugin(cls:Class<?>)
PluginInterface
getDescription(): String
getPluginType(): PluginType
getName(): String
extends
returns
ClusteringPlugin
Supports(): boolean
fit(data : ParsedData , config : Map<String, Object>) : ClusteredData
requiresConfiguration() : boolean
supportedParsedType() : Class<? extends ParsedData>
getParameters(): Map<String, Object>
getPluginType() : PluginType
ParserPlugin
getSupportedFileExtension(): List<String>
getParsedDataType() : Class<? extends ParsedData>
getParameters() : Map<String, Object>
requiresConfiguration() : boolean
getPluginType() : PluginType
VisualizationPlugin
getSupportedClusteredDataType(): Class<? extends ClusteredData>
createView(data:ClusteredData) : VisualizationView
requiresConfiguration() : boolean
getParameters(): Map<String, Object>
getPluginType() : PluginType
PluginType
PARSER, CLUSTERING, VISUALIZATION
fit() returns
parse() returns
getParameters() uses
createView() returns
notifies
calls/uses
View
PluginConfigDialog
PluginConfigDialog(parameterTemplate:Map<String, Object>)
VisualizationView
displayClusters(data: ClusteredData)
Observer
update(data: ClusteredData)
DendrogramView
displayClusters(data: ClusteredData)
ScatterPlotView

The plugin architecture in this project is fundamental to achieving extensibility and modularity. By encapsulating functionalities such as parsing, clustering, and visualization as plugins, the system allows new components to be integrated easily without altering the core codebase. This approach reduces maintenance overhead and enables researchers to extend the software with minimal effort.

3.5.1 Purpose and Benefits:

- Enables dynamic extension at runtime.
- Supports multiple types of functionality (e.g., data parsing, clustering, visualization) via a common interface.
- Provides flexibility and encourages reusability by adhering to design patterns like Strategy, Adapter, Observer and Factory.
- Allows integration of new bioinformatics tools while maintaining a uniform framework, which is crucial for scalability (D., n.d.; Shannon, 2003).

3.5.2 Key Interfaces and Their Method **PluginInterface:**

- *Role:* Acts as the root interface for all plugin types, enforcing a basic contract that includes methods to retrieve the plugin's name, description, and type.
- *Methods:*
  - getName(): Returns the plugin name.
  - getDescription(): Provides a description of the plugin's purpose.
  - getPluginType(): Identifies the plugin category (PARSER, CLUSTERING, VISUALIZATION).

ParserPlugin:

- *Role:* Extends PluginInterface and ParserStrategy to handle parsing of biological data. It specifies methods for file extension support and returning a ParsedData object.

- *Key Methods:*
    - getSupportedFileExtension(): Returns a list of file types the plugin can parse. o

      getParsedDataType(): Indicates the specific type of ParsedData produced.
    - getParameters(): Provides default configuration parameters; useful for dynamic configuration if required.
    - requiresConfiguration(): Signals whether additional user configuration is needed prior to parsing.

ClusteringPlugin:

- *Role:* Similar to ParserPlugin but focused on clustering algorithms. It extends both PluginInterface and ClusterStrategy.
- *Key Methods:*
    - fit(data, config): Executes the clustering algorithm using a dynamic configuration map.
    - getParameters(): Returns default configuration settings.
    - requiresConfiguration(): Indicates if the algorithm requires additional setup before execution.
    - supports(data): Checks if the provided ParsedData is compatible with the clustering algorithm.

VisualizationPlugin:

- *Role:* Provides a framework for creating diverse visualization techniques. It extends PluginInterface.
- *Key Methods:*
    - getSupportedClusteredDataType(): Identifies the type of clustered data the plugin can visualize.
    - createView(data): Returns a JavaFX component (VisualizationView) that renders the clustered data.
    - getParameters(): Offers configuration parameters for the visualization component.
    - requiresConfiguration(): Indicates whether user interaction is needed before rendering the visualization.

PluginType:

- *Role:* A simple enumeration used to classify plugins into PARSER, CLUSTERING, or VISUALIZATION categories. This categorization is key to organizing plugins within the application.

PluginManager:

- *Role:* Manages the discovery and loading of plugins. It scans a predefined directory for JAR files, dynamically loads them, and then provides access to plugins via its public methods.
- *Functionality:*
  - Aggregates plugins of various types (ParserPlugin, ClusteringPlugin, VisualizationPlugin).
  - Provides methods (like loadPlugins()) to refresh and manage the available plugins, enhancing modularity and ease of extension.

3.5.3 How Plugins are Used

Plugins are integrated into the system using a uniform approach:

- The Controller layer (specifically, ClusteringController) accesses PluginManager to retrieve and execute clustering plugins.
- Each plugin's getParameters() method supplies a configuration template, which is then presented to the user via dialogs like PluginConfigDialog if configuration is required.
- Once configured, the plugin’s fit() (for clustering) or createView() (for visualization) method is called to process data and produce outputs.
- This dynamic loading and execution ensure that new functionalities can be added to the system without rewriting existing code, a principle supported by design patterns outlined in software engineering literature (Gamma et al., 1994).

3.5.4 Extending the Plug-in

Developers can extend the system easily by creating new plugins that adhere to the defined interfaces. The plugin architecture is built around three main interfaces—ParserPlugin, ClusteringPlugin, and VisualizationPlugin—which extend PluginInterface. To add a new functionality, a developer must implement the corresponding interface, ensuring that the plugin returns the appropriate type (such as ParsedData, ClusteredData, or VisualizationView) and provides configuration details through the getParameters() method.

- **For a New Parser Plugin:**
  - Implement the ParserPlugin interface.
  - Provide a list of supported file extensions via getSupportedFileExtension().
  - Implement the parse(filename: String) method to return a ParsedData object (commonly a specialized form like GeneExpressionParsedData).
  - Optionally override requiresConfiguration() and getParameters() to support dynamic configuration.
- **For a New Clustering Plugin:**
  - Implement the ClusteringPlugin interface, which extends ClusterStrategy.
  - Write the fit(data: ParsedData, config: Map<String, Object>) method so that it returns a ClusteredData object according to the clustering algorithm.
  - Override supports(data: ParsedData) to check compatibility, and provide default configuration via getParameters().

- Ensure to declare whether additional configuration is needed using requiresConfiguration().

- **For a New Visualization Plugin:**
  - Implement the VisualizationPlugin interface.
  - Define getSupportedClusteredDataType() to specify which form of ClusteredData the plugin accepts.
  - Implement createView(data: ClusteredData) to return a JavaFX component (a VisualizationView) that visually represents clustering results.
  - Optionally, override requiresConfiguration() and getParameters() if the visualizer requires user input before rendering.

By following these guidelines, new plugins integrate seamlessly into the existing system via the PluginManager, which dynamically discovers and loads plugins at runtime. This modular design, combined with clear interface contracts, ensures that developers can add new functionalities—whether parsing new data types, adding novel clustering algorithms, or creating visualization techniques—without modifying the core software (Erich Gamma, 1994)

3.6 Critical Analysis and Discussion

In developing the extensible bioinformatics clustering package, several successes and challenges emerged throughout the project lifecycle. This section critically examines the progress, the effectiveness of various design choices, and areas where further refinements could enhance the system.

- **Design and Architecture:**
  - The use of the Strategy pattern for both parsing and clustering has enabled flexible integration of multiple algorithms. This architectural decision provides clear modularity, allowing new parsers or clustering algorithms to be incorporated with minimal changes to the existing codebase.
  - The implementation of a plugin-based approach has met its goal by allowing dynamic extension. However, integrating dynamic configuration dialogs introduced complexity in ensuring that each plugin adheres to a consistent interface for parameters.
  - The MVC separation has effectively decoupled user interface logic from business logic; however, ensuring thorough communication between the Controller and the dynamically loaded plugins was a technical challenge that required careful design of adapter interfaces.

- **Implementation Efficiency and Code Quality:**
  - Overall, the code exhibits good modularity and readability, supported by a consistent use of design patterns such as Strategy and Observer.
  - The extensive use of interfaces (e.g., ParserStrategy, ClusterStrategy) has promoted polymorphism, yet the actual implementations—such as those in SoftParser and CSVParser—could benefit from further optimization in handling large data volumes.

   - Error handling has been implemented at key points (e.g., validating file formats, using JavaFX Alerts), but additional unit and integration testing would further solidify the code's robustness.

- **Testing and Integration:**
   - The integration of unit tests and manual testing during development has shown critical areas for refinement, particularly in parsing complex datasets and ensuring that clustering outputs align with expectations.
   - The dynamic update mechanisms (via the Observer pattern) have improved system responsiveness, though problems in UI updates during edge cases have been observed and warrant further investigation.

- **Usability and Extensibility:**
   - The JavaFX GUI offers a user-friendly interface that reduces the learning curve for bioinformatics tools, and dynamic plugin loading empowers end users to extend functionality as needed.
   - Future work could enhance usability by refining input validation in configuration dialogs and incorporating more advanced visualization options.
   - The system's extensibility is a strong point; however, maintaining this flexibility while ensuring consistent performance will require ongoing evaluation as new plugins or data types are integrated.

In summary, the project shows strengths in modular design and extensibility. While the implemented solution addresses the primary requirements effectively, further improvements in error handling, performance optimization, and comprehensive testing could elevate the overall robustness and user experience. This reflection informs the planned future enhancements and provides a solid foundation for subsequent development.

Chapter 4: Software Engineering

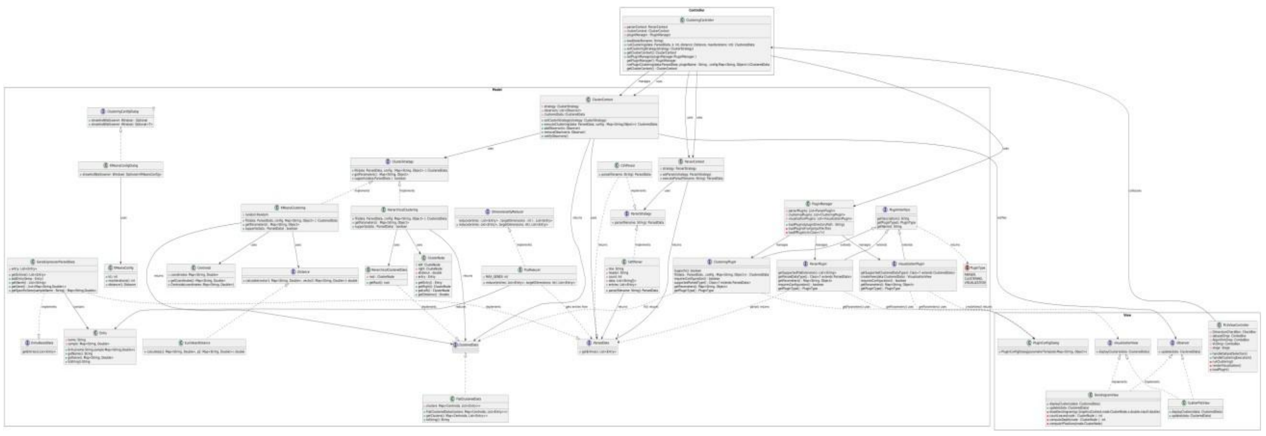


Figure 5. The entire application.

4.1 Requirement Elicitation (Or Planning)

The project requirements were planned through an iterative planning process, using both theoretical insights and practical considerations. Initially, I read thorugh an extensive amount of literature, studying key works such as "Design Patterns: Elements of Reusable Object-Oriented Software" (Gamma et al., 1994) and "Building Bioinformatics Solutions" (Bessant, 2009) to understand the challenges and best practices in developing modular and extensible bioinformatics tools. This theoretical foundation was complemented by examining contemporary bioinformatics platforms like Cytoscape (Shannon, 2003) and Galaxy (Goecks, 2010), which highlighted the need for user-friendly interfaces and flexible architectures.

To translate these findings into practical requirements, I had discussions with my supervisor and had brainstorming sessions to understand and plan the essential functional needs—such as the ability to parse multiple data formats (e.g., SOFT and CSV), dynamically select and configure clustering algorithms (including K-means and hierarchical clustering), and provide intuitive visual outputs through JavaFX visualizations, most importantly the need to keep things loosely coupled, design patterns and extensibility in mind. Additionally, feedback from early prototypes, including my Figma designs, helped refine non-functional requirements like usability, performance, and extensibility.

An agile, iterative approach to requirement review was used, where early test cases and proof-of-concept implementations helped with subsequent refinement. This process ensured that the software's design would support dynamic plugin loading, error handling, and easy future extensions. The combination of structured literature review, comparative analysis of existing tools, and iterative prototyping provided a complete set of requirements that guided the development and eventual integration of the system's various components (Bird et al., 2009; Chacon & Straub, 2014).

4.2 Software Design

4.2.1 Sequence diagrams

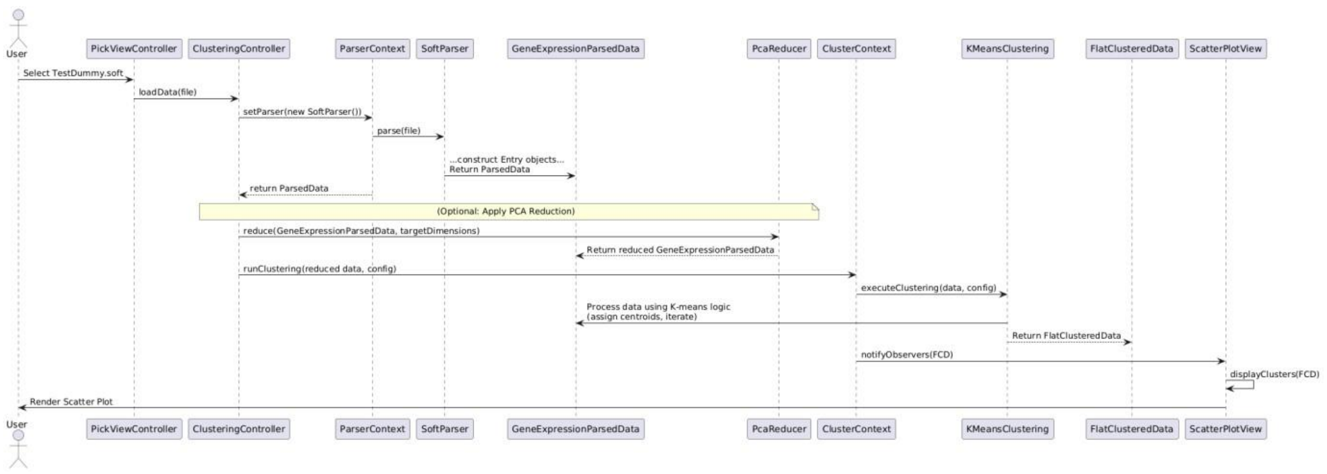


Figure 6. This Sequence Diagram shows the detailed interactions when a user loads a SOFT file, applies PCA-based dimensionality reduction, executes K-Means clustering, and updates the ScatterPlot visualization. The flow is as follows:

- The **User** selects the file (TestDummy.soft).
- **PickViewController** calls **ClusteringController.loadData()**, which in turn uses **ParserContext** and **SoftParser** to return a **GeneExpressionParsedData** object.
- If PCA reduction is enabled, the PCA module (PcaReducer) processes the data.
- **ClusteringController.runClustering()** is then called with clustering parameters (via KMeansConfigDialog) and the processed data.
- The clustering result is returned as a **FlatClusteredData** object, and **ClusterContext** notifies its observers.
- **ScatterPlotView** (an observer) receives the update and calls **displayClusters()** to render the results.

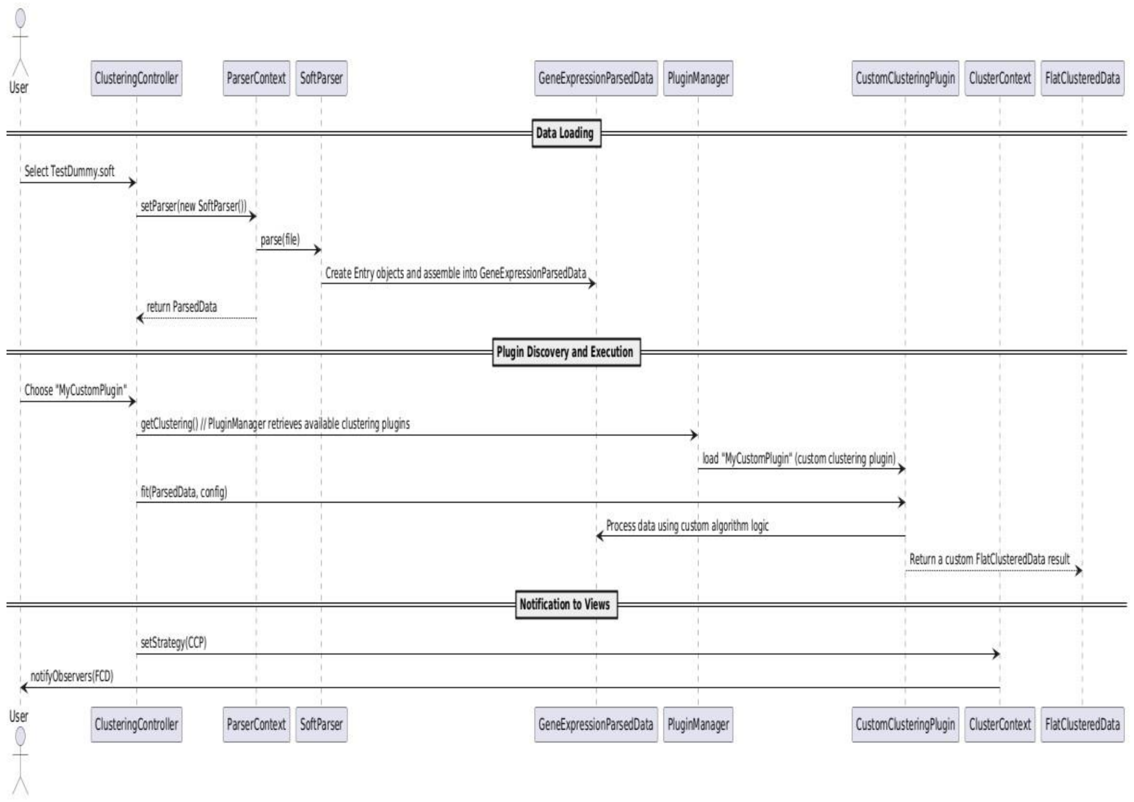

User
ClusteringController
ParserContext
SoftParser
GeneExpressionParsedData
PluginManager
CustomClusteringPlugin
ClusterContext
FlatClusteredData
Data Loading
Select TestDummy.soft
setParser(new SoftParser())
parse(file)
Create Entry objects and assemble into GeneExpressionParsedData
return ParsedData
Plugin Discovery and Execution
Choose "MyCustomPlugin"
getClustering() // PluginManager retrieves available clustering plugins
load "MyCustomPlugin" (custom clustering plugin)
fit(ParsedData, config)
Process data using custom algorithm logic
Return a custom FlatClusteredData result
Notification to Views
setStrategy(CCP)
notifyObservers(FCD)

Figure 7. Illustrates how a developer can extend the system by writing their own clustering software for .soft files. This diagram shows how a custom clustering plugin (named "MyCustomPlugin") that implements the ClusteringPlugin interface is discovered, loaded, and executed within the system.

4.3 Testing

To validate the functionality, performance, and robustness of the system, multiple testing strategies have been employed:

- **Unit Testing for Model and Controller Layers:**
  Unit Tests were written to rigorously verify the functionality of key components such as the parsers (SoftParser, CSVParser), clustering algorithms (K-Means, Hierarchical Clustering), and plugin management. These tests ensure that:

- Each parser correctly converts raw files into a consistent ParsedData structure. o
  Clustering algorithms correctly process data and produce ClusteredData outputs. o The PluginManager successfully loads and manages plugins.
- Context classes can change algorithms without the need for hardcoded logic o
  Logic of all the in-built algorithms are sound and perform within spec o
  Views update and clusters are meaningful o Interfaces and design patterns work like expected

- **Integration Testing:**
  Integration tests have been conducted to confirm that the interaction between the ParserContext, ClusterContext, and ClusteringController works as intended. The tests simulate end-to-end scenarios—starting from file loading, through clustering execution, to the notification of observers—ensuring that the system components interact correctly under various configurations.

- **Manual Testing for the View Layer** : Because the visual components (ScatterPlotView and DendrogramView) involve dynamic graphical updates, manual testing was extensively employed:

  - The JavaFX UI was tested in various scenarios to ensure that elements correctly respond to user actions (e.g., file selection, plugin refresh, and configuration dialogs).

  - Tools like TestFX were considered to further automate UI testing; however, due to the dynamic and graphical nature of the interface, manual validation played a key role in confirming usability.

- **Performance and Robustness Evaluation** : Special attention was given to stress-testing parsing and clustering routines with large bioinformatics datasets to ensure the system scales and remains stable under load. Edge cases such as invalid file formats and configuration errors were also systematically tested, ensuring robust error handling.

- **Security Considerations**: Basic security aspects were tested by verifying that the system only processes allowed file formats and handles exceptions gracefully to prevent potential crashes or data corruption.

These testing methods have collectively contributed to a reliable and maintainable system. If needed, I can provide additional test cases and scripts from the Git repository that further demonstrate the thoroughness of the testing process (Chacon & Straub, 2014)

4.4 Code Quality and Version Control

To maintain high code quality, I worked with best practices in software engineering by ensuring that the code is clean, modular, and well-documented. The project extensively uses design patterns—specifically the Strategy pattern (for interchangeable parsers and clustering algorithms) Adapter and the Observer pattern (for updating visualization components)—which promote loose coupling and high cohesion. Javadoc comments and inline documentation have been provided throughout the codebase, making easier maintenance and future extension. Regular code reviews and the use of tools like Checkstyle have been instrumental in keeping the code consistent and readable.

For version control, I used Git due to its distributed architecture, which allows each developer to work with a full local repository. Git's useful branching and merging capabilities enable safe experimentation and collaborative development, while platforms like GitHub/GitLab support continuous integration and code review processes. This workflow not only ensures a robust history of code changes but also contributes to overall project stability and accountability (Chacon & Straub, 2014; Bird et al., 2009).

Chapter 5: Professional issues

Professional Issues

The development of this extensible clustering platform for bioinformatics involved technical challenges and several professional issues that have had implications throughout the project lifecycle. In today's digital age, professional and ethical conduct in computing is paramount, as highlighted by professional bodies such as the British Computer Society (BCS) and the Association for Computing Machinery (ACM) (Bynum, 2003). This section discusses various professional issues encountered during the project, including proper citation, licensing, accessibility, and ethical use of technology, and reflects on how these considerations have influenced the design and implementation process.

Citation and Academic Integrity

Correct citation is a fundamental professional issue in academic work. Throughout this project, I have relied on a wide range of literature—from seminal texts like "Design Patterns: Elements of Reusable Object-Oriented Software" (Gamma et al., 1994) to contemporary articles on bioinformatics platforms such as Cytoscape (Shannon, 2003) and Galaxy (Goecks, 2010). Proper citation not only gives credit where it is due but also reinforces the credibility of the research. During the development process, I ensured that all external code snippets, libraries, and research findings were appropriately referenced in my work. Failing to do so would undermine the academic integrity of the project and violate ethical norms within the computing community. As academic professionals, it is our duty to maintain the highest standards of scholarly honesty.

Licensing and Open-Source Software

Licensing is another critical aspect that has been carefully considered throughout this project. The choice of using Java and JavaFX, as well as various open-source libraries (such as PlantUML for diagramming and Git for version control), showed a commitment to using widely accepted, community-driven tools that often come with permissive licenses. These licenses (e.g., Apache, MIT) ensure that the software can be extended or modified without restrictive conditions. However, we must be vigilant when combining different libraries to ensure compatibility of licenses and to avoid any accidental conflicts that could inhibit future development or distribution. In this project, I have taken care to comply with the licensing requirements of all third-party tools used, and I have documented these appropriately within the project repository and report. This not only protects the project legally but also aligns with ethical standards that safeguard intellectual property while promoting open-source collaboration.

Accessibility and Usability

One of the main motivations behind this project was to reduce the barriers that many bioinformatics tools impose on end users. The goal was to create a system with a low learning curve, one that does not require deep technical knowledge to operate. The use of JavaFX for the user interface demonstrates this guarantee to accessibility. JavaFX provides a modern, responsive interface that makes it easier for researchers—regardless of their programming background—to select datasets, configure clustering algorithms, and visualize results interactively. The design of the UI was informed by principles of usability and accessibility, ensuring that the system is intuitive and that its components (such as dynamic configuration dialogs) provide clear feedback. These design decisions are critical because, as noted in professional guidelines, accessible software not only improves productivity but also reflects a commitment to ethical design by ensuring that technological advancements benefit all users.

Ethical Use and Societal Impact

Professionalism in computing extends beyond technical skills to consider the societal and ethical implications of software development. In bioinformatics, ethical issues take on additional layers of complexity due to the sensitive nature of biological data. This project has been designed with a focus on ensuring that user data is handled securely and responsibly, with robust error handling and clear user feedback mechanisms that minimize misuse or misinterpretation of results. The use of a modular, plugin-based architecture enables transparency, as each plugin is self-contained and its functionality clearly defined. This allows users and developers to audit the code more easily and to trust that the algorithms and visualizations provided are both reliable and ethically designed. As Terrell Ward Bynum (2003) suggests, recognizing the societal impact of computing is a critical part of being a responsible computing professional.

Project Management and Version Control

Effective project management and version control are essential professional practices that have played a crucial role in the development of this project. Git was chosen as the version control system due to its distributed nature, powerful branching, and merging capabilities. Using Git, I was able to maintain a complete history of changes, experiment with different features safely through branches, and ensure that I could rollback to stable versions when necessary. This adherence to best practices in version control not only allows collaboration but also ensures that the project's evolution is transparent and well-documented. Consistent use of Git has enabled regular backups and a structured approach to the code's iterative development, which is a hallmark of professional software engineering.

Time Management and Resource Allocation

Another professional issue addressed during the project was effective time management and resource allocation. Developing complex bioinformatics software while balancing academic commitments requires strong time management strategies and a structured plan. Throughout this project, I maintained a detailed work diary and employed agile methodologies to iteratively develop, test, and refine various components. This approach ensured progress and also provided a clear record of how requirements evolved over time, which is critical for both self-reflection and professional accountability. In retrospect, while the project faced technical challenges—including parsing large, complex SOFT files and integrating dynamic plugin configurations—the iterative development process allowed for timely identification and resolution of issues, thereby ensuring that the project stayed on track.

Conclusion of Professional Issues

In summary, the professional issues encountered during this project—ranging from ethical citation and licensing practices, ensuring accessibility and usability, to robust project management and version control—have significantly shaped the development process. By addressing these issues proactively, the project not only meets academic and industrial standards for professional software development but also ensures that the final product is both reliable and ethically sound. These considerations underscore the importance of professionalism in computing, as endorsed by bodies like the BCS and ACM, and serve as a testament to the project's commitment to excellence in both technical and ethical dimensions.

Chapter 6: Self-evaluation & Next Steps

This project has achieved significant milestones in developing a modular and extensible clustering environment for bioinformatics. With the integration of design patterns, such as Strategy, Adapter, Observer, and a dynamic plugin architecture, the project has demonstrated its ability to handle diverse biological datasets and clustering techniques.

Achievements:

- **Modular Architecture:**
    - Successfully implemented a Model-View-Controller structure that separates core data processing from user interface concerns, making the system flexible and maintainable.
    - The plugin framework has enabled the integration of multiple parser and clustering strategies without modifying the core system.
- **Parsing and Clustering:**
    - Developedparsing algorithms (SoftParser and CSVParser) that convert raw biological data into consistent internal representations.
    - Implemented two clustering algorithms—K-means and Hierarchical clustering—each configurable via dynamic dialogs.
    - The use of adapter interfaces (e.g., EntryBasedData) has enabled polymorphism, making sure that new data types can be integrated easily.
- **User-Friendly Interface:**
    - A JavaFX-based GUI significantly lowers the entry barrier, allowing users to select datasets, configure algorithms, and visualize clustering outputs through ScatterPlotView and DendrogramView.
    - The dynamic configuration dialogs and real-time update mechanisms (via the Observer pattern) improve user interaction and overall system responsiveness.
- **Testing and Stability:**
    - Thorough unit and integration tests have been used to validate individual components such as parsers, clustering algorithms, and plugin loading, ensuring system stability under various scenarios.
    - Manual testing of the UI confirms that the software meets its usability and performance requirements.

Challenges:

- **Complex Data Parsing:**
    - Parsing SOFT files, given their hierarchical and voluminous nature, posed challenges in terms of performance and memory management.
    - Ensuring consistency in data representation across different parsers required rigorous design and iterative testing.

- **Plugin Configuration and Integration:**
    - Designing a unified parameter configuration mechanism that helps diverse plugin requirements introduced complexity.
    - Balancing extensibility with error handling, especially in dynamic environments, proved to be challenging.
- **User Interface Refinements:**
    - While the JavaFX GUI is effective, integrating real-time updates and maintaining responsiveness under complex visualization scenarios asked the need for further optimization.
    - Ensuring that the interface remains accessible to non-technical users required additional design iterations.

Next Steps:

- **Enhanced Testing Framework:**
    - Develop more automated UI tests (using tools like TestFX) and complete integration tests to ensure full coverage of edge cases and user scenarios.
    - Continue to refine performance benchmarks, particularly for large datasets and complex clustering configurations.
- **Expanding Plugin Capabilities:**
    - Extend the plugin framework to support additional clustering algorithms and visualization techniques.
    - Consider incorporating a factory pattern or additional adapter interfaces to further streamline the integration of new data types and processing modules.
- **User Interface and Usability Improvements:**
    - Further refine the JavaFX interface to enhance responsiveness and accessibility; optimize dynamic dialog behaviors and error feedback.
    - Explore potential enhancements in visualization options—such as multidimensional scaling (MDS) or advanced dendrogram interactivity.
- **Documentation and Deployment:**
    - Update the user and installation manuals with detailed step-by-step instructions, including screenshots and video demonstrations.
    - Ensure all elements of the project (code, documentation, test suites) are wellintegrated into the Git repository, reflecting best practices in version control and continuous integration.
- **Reflection and Professional Development:**
    - Reflect on the lessons learned regarding time management, design trade-offs, and integration challenges.

- Identify areas for personal growth in adopting advanced software engineering practices that will prepare you for future professional challenges.

Overall, the project has successfully addressed its primary objectives, displaying robust design and extensibility. While challenges remain—particularly in optimizing parsing performance and further refining the UI—these issues provide valuable opportunities for future improvements and continued professional growth. This reflection, alongside detailed documentation and a clear roadmap for next steps, show the project's commitment to excellence in both technical execution and professional practice.

Chapter 7: Bibliography

(n.d.). Retrieved from TIOBE: https://www.tiobe.com/tiobe-index/java/

Ali Dehghani, E. M. (2024, January 8). *The K-Means Clustering Algorithm in Java*. Retrieved from baeldung: https://www.baeldung.com/java-k-means-clustering-algorithm

Anna Niarakis, D. W. (2022). Addressing barriers in comprehensiveness, accessibility, reusability, interoperability and reproducibility of computational models in systems biology. *Briefings in Bioinformatics*, Volume 23, Issue 4.

Bessant, C. S. (2009). *Building bioinformatics solutions : with Perl, R and MySQL.* Oxford University Press.

Bloch, J. (2017). *Efficitive Java.* Addison-Wesley Professional.

CRICK, F. (1970). Central Dogma of Molecular Biology. *Nature , 227*, 561–563 .

Eric Clayberg, D. R. (2008). *Eclipse Plug-ins 3rd Ed.* Addison-Wesley Professional.

Erich Gamma, R. H. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional.

Gentleman, R. C. (2008). Bioconductor: open software development for computational biology and bioinformatics. *Genome Biol 5*.

Goecks, J. N. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, 1-13.

Jeff Gauthier, A. T. (2019). A brief history of bioinformatics. *Briefings in Bioinformatics, 20*(6), 1981–1996.

Lesk, A. M. (2019). *Introduction to bioinformatics.* Oxford university press.

Reich, M. L. (2006). GenePattern 2.0. *Nat Genet*, 500-501.

Shannon, P. M. (2003). Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 2498-2504.

Spjuth, O. H. (2007). Bioclipse: an open source workbench for chemo- and bioinformatics. *BMC Bioinformatics*, 59.

Steiper, M. E. (2005). *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins.* New York: John Wiley and Sons.

Chapter 8: Appendices

8.1 Project Diary

2-3th October

Set up git repo, started structing my project plan, did intial reading in bioinfomratics.

4th October

Read about protein sequencing, DNA synthesis which is how later on CS and biologists came together to create this field of bioinformatics, since this created a lot of data which had to be analysed. This helped me start my abstract of the project paper.

5th October

Reading a Research paper that has helped me understand the evolution of protein to DNA analysis, which has helped me refine my abstract until now. Plan is to finish it and majority of my abstract by tomorrow.

7th October

On my way to finish the research paper i was reading on the 5th, went on a tangent of reading 2 more research papers which were informative but not too useful for abstract, hoping to finish abstract tomorrow and start working on timeline and risk mitigations.

9th October

Finally finished my abstract and got my citations in, going to start working on the timeline and risk mitigations.

11th Ocotber

Finished my plan and submitted

28th October

Reading through SE books to plan out my code - Hope to have a rough outline by Wednesday to discuss with supervisor on Thursday

11th November

Looked at different biological sets to use and their respective clustering that can be used - going to use gse10072 for the first biological dataset - and implement k means clustering on it.

12th

Tried to find datasets that i could use - but ended up starting from the ground up and reading a bit of lit before i start my project, there are a lot of things that i dont understand that i hope i can once i am through with the lit - plug in system, datasets for bioinformatics, software engineering principles.

I started reading building bioinformatics solution(by conrad bessant et al) to understand how the data is stored and their relevant metadata, this book teaches us how a bioinformatics software is created from scratch - hopefully when i am through with the relevant chappters i can have more confidence.

15th-18th

Reading on lit, the design pattern help understand what i could implement and working on the parser for a soft file since its one of the harder files to parse, this will give me a good grasp for future file parsing.

19th november

Worked on the SoftParser class so it can parse SOFT data files - in progress

2nd decemeber

Finished SoftParser class, now it can successfully parse a SOFT file, was working on a bug 21-24 which caused the parser to take in unnecessary lines.

3rd december

Worked on making suplementary classes for my K means clustering, we can turn our parser output into it's respective classes so it can be encapsulated well, this will help for our code to be cleaner and more useable.

Worked on eucledian distance - i am using the eucledian because the implementation is pretty straightforward.

Worked on centroid representation for our k means - so our clusters can be in the same dimension and converge faster

4th december

Our softparser was giving out a very complicated data structure which didn't align well with good perormance and optamization, started working on refactoring that.

Worked on centroid generation, and added SE elements into code for better SE practise(Strategy)

5th december

Worked on K means more - for random centroid generation

6th-8th decemebr

Finished working on K means - assign, average and the other methods

9th-11th

refactoring SoftPArser for a more apropriate data structure.

==break===

jan 18-30: project was revisited to improve redability, there were a lot of intilisation mistakes - which were then taken care of

jan 31 - feb 6: Design of the JavaFx interface with prototyping, SOLID principles were visited to see if data persistence was possible feb 7 - feb 13: We wont be having data persistence since its not paramount, new UML ddesiging takes place, i have to look into introducting design patterns since the codebase needs to be modular.

feb 13- feb 22 : New UML with strategy, and observer were created and had a meeting with supervisor discussing the same.

feb 26- feb 28: old classes were brushed up, new unit tests for everything and addition of new classes(cluserting) to meet spec.

march 4 - march 11: Ran into javaFx compatability issue, turns out my JavaFx files weren't compatible with Scenebuilder, the plugin used by eclipse wan't helped had to do a clean rebuild of everything, and redownload all the needed files, during this time i was working on the UI on a different package for this project but forgot to keep it tracked

march 13- march 20: Extensive changes of UML and discussion with supervisor about the same, got the interface checked and got feedback, working on the same.

march 24th- April 1: Working on Code, implementing design patterns for the needed Classes(clustering - view -parsings), a lot of refactoring as classes evolve.

April 1 - April 11 : Started working on Final report submission alongside the code, code is slowing being generalsied with a lot of focus on plug-ins, design patterns

April 11 - April: Final touches on Plug-in a lot of manual testing for JavaFx intergration. UML upgradation and first major release of software and lock down of the UML for the same, final report comes to an end and submission.

8.2 UML figure code

8.2.1 Figure 1

```
@startuml

' Force layout with positioning scale

0.5


rectangle "Model" as M {
  ' Layer 1: Entry & ParsedData types
class Entry {
    -name: String
    -sample: Map<String, Double>
    +Entry(name:String,sample:Map<String,Double>)
    +getName(): String
    +getGene(): Map<String, Double>
    +toString():String
  }

  interface ParsedData {
  }

class PcaReducer {
    -MAX_GENES: int
    +reduce(entries: List<Entry>, targetDimensions: int): List<Entry>
  }
  class GeneExpressionParsedData {
    -entry: List<Entry>
    +getEntries():              List<Entry>
+addEntry(temp : Entry)
    +getName() : List<String>
    +getGene() : List<Map<String,Double>>
    +getSpecificGene(sampleName : String) : Map<String,Double>
  }

  '   Layer   2:   Parsing
interface ParserStrategy {
    +parse(filename: String): ParsedData
```

```
   }
interface EntryBasedData{  getEntries():List<Entry>
}


   class SoftParser {         -
line: String
       -header: String
       -count: Int
       -data: List<String[]>
       -entries: List<Entry>
       +parse(filename: String): ParsedData
   }
class CSVParser {
+parse(filename: String): ParsedData
} interface
EntryBasedData{
getEntries():List<Entry>
}
interface  DimensionalityReducer  {   reduce(entries  :  List<Entry>  ,
targetDimensions : int ) : List<Entry>
}     class ParserContext
{
       -strategy: ParserStrategy
       +setParser(strategy: ParserStrategy)
       +executeParse(filename: String): ParsedData
   }



   ' Relationships
```

```
    ParserStrategy <|.. SoftParser : implements
    ParserContext --> ParserStrategy : uses
    ParserContext --> "1" ParsedData : returns
    SoftParser --> ParsedData : returns
    GeneExpressionParsedData --> Entry : contains
GeneExpressionParsedData ..|> EntryBasedData : "implements"
CSVParser ..|> ParserStrategy : implements
CSVParser ..|> ParsedData : returns
EntryBasedData ..|> ParsedData : "adapts"
    DimensionalityReducer <|.. PcaReducer : implements
    PcaReducer --> Entry : reduces
    PcaReducer ..> ParsedData : returns


}
@enduml
```

8.2.2 Figure 2

```
@startuml
' Force layout with positioning scale
0.5



rectangle "Model" as M {
    ' Layer 1: Entry & ParsedData types
' Layer 3: Clustering Core class
PluginConfigDialog{
+PluginConfigDialog(parameterTemplate:Map<String, Object>)
}
```

```
    interface ClusterStrategy {
        +fit(data: ParsedData, config : Map<String, Object> ): ClusteredData
        +getParameters() : Map<String, Object>
     +supports(data:ParsedData ) : boolean
}                                          interface
ClusteringConfigDialog<T>{
+showAndWait(owner: Window) : Optional
}

class ClusterNode{ -left:
ClusteNode
-right: ClusteNode
-distance : double
-entry : Entry

+getEntry() : Entry
+getRight(): ClusterNode
+getLeft() : ClusterNode
+getDistance() : Double
}

class HierarchicalClusteredData{
-root : ClusterNode
+getRoot(): root

}    class KMeansClustering
{
        -random:Random
        +fit(data: ParsedData, config: Map<String, Object> ): ClusteredData
+getParameters() : Map<String, Object>

+supports(data : ParsedData) : boolean


}                                class

HierarchicalClustering{

     +fit(data: ParsedData, config: Map<String, Object> ): ClusteredData

+getParameters() : Map<String, Object>

+supports(data : ParsedData) : boolean


}    class ClusterContext

{

     -strategy: ClusterStrategy

     -observers: List<Observer>

     -clusteredData: ClusteredData

     +setClusterStrategy(strategy: ClusterStrategy)

     +executeClustering(data: ParsedData, config : Map<String,Object>): ClusteredData

     +addObserver(o: Observer)

     +removeObserver(o: Observer)

     +notifyObservers()

  }


  interface     ClusteredData     {}

class FlatClusteredData {

     -clusters: Map<Centroids, List<Entry>>

     +FlatClusteredData(clusters: Map<Centroids, List<Entry>>)

     +getClusters(): Map<Centroids, List<Entry>>

     +toString(): String

  }
```

```
class Centroids {
    -coordinates: Map<String, Double>
    +getCoordinates(): Map<String, Double>
    +Centroids(coordinates: Map<String, Double>)
}

' Layer 4: Distance
interface Distance {
    +calculate(vector1: Map<String, Double>, vector2: Map<String, Double>): double
}

class EuclideanDistance {
    +calculate(p1: Map<String, Double>, p2: Map<String, Double>): double
}



' Layer 6: Configuration
class KMeansConfig {
    +k(): int
    +maxIterations(): int
    +distance(): Distance
}

interface ClusteringConfigDialog<T> {
    +showAndWait(owner: Window): Optional<T>
}

class KMeansConfigDialog {
    +showAndWait(owner: Window): Optional<KMeansConfig>
}


' Relationships


ClusterStrategy <|.. KMeansClustering : implements
ClusterContext --> ClusterStrategy : uses
KMeansClustering --> Centroids : uses
KMeansClustering --> Distance : uses
Distance <|.. EuclideanDistance
KMeansClustering --> ClusteredData : returns
HierarchicalClustering --> ClusterNode : uses
HierarchicalClustering --> HierarchicalClusteredData: uses
HierarchicalClusteredData ..|> ClusteredData : implements
HierarchicalClustering --> ClusteredData : returns
ClusterStrategy <|.. HierarchicalClustering : implements
ClusterContext --> ParsedData : uses
ClusterContext --> ClusteredData : returns
ClusterContext --> Observer : notifies
ClusteredData <|.. FlatClusteredData
' All plugin interfaces extend PluginInterface
ClusteringConfigDialog <|.. KMeansConfigDialog
KMeansConfigDialog ---> KMeansConfig : uses
KMeansClustering --> PluginConfigDialog : uses
HierarchicalClustering  --> PluginConfigDialog : uses
}
@enduml
```

8.2.3 Figure 3

```
@startuml

' Force layout with positioning

scale 0.5 rectangle "View" as

V {     interface Observer {

        +update(data: ClusteredData)

    }


    interface VisualizationView {

        +displayClusters(data: ClusteredData)

    }


    class ScatterPlotView {

        +displayClusters(data: ClusteredData)

        +update(data: ClusteredData)

    }

class DendrogramView {

        +displayClusters(data: ClusteredData)        +update(data: ClusteredData)

-drawDendrogram(gc:GraphicsContext,node:ClusterNode,x:double,maxX:double)

-countLeaves(node : ClusterNode ) : int

-computeDepth(node : ClusterNode ) : int

-computeYPositions(node:ClusterNode)

    }

class PluginConfigDialog{

+PluginConfigDialog(parameterTemplate:Map<String, Object>)

}

    VisualizationView <|.. ScatterPlotView

    Observer <|.. ScatterPlotView
    class PickViewController {
- DimensionCheckBox: CheckBox
- datasetDrop: ComboBox
- AlgorithmDrop: ComboBox
- VizDrop: ComboBox
- stage: Stage
        + handleDatasetSelection()
        + handleClusteringExecution()
-runClustering()
-renderVisualization()
-loadPlugin()
    }
}



rectangle "Model" as M {



    class ClusterContext {
        -strategy: ClusterStrategy
        -observers: List<Observer>
        -clusteredData: ClusteredData
        +setClusterStrategy(strategy: ClusterStrategy)
        +executeClustering(data: ParsedData, config : Map<String,Object>): ClusteredData
        +addObserver(o: Observer)
        +removeObserver(o: Observer)
        +notifyObservers()
    }
```

```
   interface ClusteredData {}




   ' Relationships

   ClusterContext --> ParsedData : uses

   ClusterContext --> ClusteredData : returns

   ClusterContext --> Observer : notifies

   ClusteredData <|.. FlatClusteredData

 VisualizationView <|.. DendrogramView : implements

 Observer <|.. DendrogramView : implements


 VisualizationPlugin ..> PluginConfigDialog : "getParameters() uses"



 VisualizationPlugin ..> VisualizationView : createView() returns



 PickViewController --> ClusteringController : "calls/uses"


 }

 @enduml
```

8.2.4 Figure 5 (entire Application)

```
 @startuml

 ' Force layout with positioning

 scale 0.5 rectangle "View" as

 V {     interface Observer {

       +update(data: ClusteredData)
```

```
    }


    interface VisualizationView {
        +displayClusters(data: ClusteredData)
    }


    class ScatterPlotView {
        +displayClusters(data: ClusteredData)
        +update(data: ClusteredData)
    }
class DendrogramView {
        +displayClusters(data: ClusteredData)
        +update(data: ClusteredData)
-drawDendrogram(gc:GraphicsContext,node:ClusterNode,x:double,maxX:double)
-countLeaves(node : ClusterNode ) : int
-computeDepth(node : ClusterNode ) : int
-computeYPositions(node:ClusterNode)
    }
class PluginConfigDialog{
+PluginConfigDialog(parameterTemplate:Map<String, Object>)
}
    VisualizationView <|.. ScatterPlotView
    Observer <|.. ScatterPlotView


    class PickViewController {
- DimensionCheckBox: CheckBox
- datasetDrop: ComboBox
- AlgorithmDrop: ComboBox
- VizDrop: ComboBox
- stage: Stage
        + handleDatasetSelection()
        + handleClusteringExecution()
-runClustering()
-renderVisualization()
-loadPlugin()
    }
}


rectangle   "Controller"   as   C   {
class ClusteringController {
        -parserContext: ParserContext
        -clusterContext: ClusterContext
-pluginManager : PluginManager
        +loadData(filename: String)
        +runClustering(data: ParsedData, k: int, distance: Distance, maxIterations: int): ClusteredData
+setClusteringStrategy(strategy: ClusterStrategy)
        +getClusterContext(): ClusterContext
+setPluginManager(pluginManager:PluginManager ) getPluginManager():
PluginManager
runPluginClustering(data:ParsedData,   pluginName      :           String    ,
           config:Map<String, Object>):ClusteredData  getClusterContext() : ClusterContext
    }
}


rectangle "Model" as M {
    '  Layer  1:  Entry  &  ParsedData  types
class Entry {
        -name: String
        -sample: Map<String, Double>
        +Entry(name:String,sample:Map<String,Double>)
        +getName(): String
        +getGene(): Map<String, Double>
        +toString():String
    }

    interface ParsedData {
        +getEntries(): List<Entry>
    }

    class GeneExpressionParsedData {
        -entry: List<Entry>
        +getEntries(): List<Entry>
        +addEntry(temp : Entry)
        +getName() : List<String>
        +getGene() : List<Map<String,Double>>
        +getSpecificGene(sampleName : String) : Map<String,Double>
    }

    '    Layer    2:    Parsing
  interface ParserStrategy {
        +parse(filename: String): ParsedData
    }

    class SoftParser {        -
  line: String
        -header: String
        -count: Int
        -data: List<String[]>
        -entries: List<Entry>

        +parse(filename: String): ParsedData

    }

  class CSVParser {

  +parse(filename: String): ParsedData

  } interface

  EntryBasedData{

  getEntries():List<Entry>

  } interface DimensionalityReducer {  reduce(entries : List<Entry> ,

  targetDimensions : int ) : List<Entry>

  }     class ParserContext

  {

        -strategy: ParserStrategy

        +setParser(strategy: ParserStrategy)

        +executeParse(filename: String): ParsedData

    }


    '    Layer    3:    Clustering    Core

  interface ClusterStrategy {

        +fit(data: ParsedData, config : Map<String, Object> ): ClusteredData

        +getParameters() : Map<String, Object>

     +supports(data:ParsedData ) : boolean

  }                                         interface

  ClusteringConfigDialog<T>{

  +showAndWait(owner: Window) : Optional

  }


  class ClusterNode{ -left:

  ClusteNode
```

```
-right: ClusteNode
-distance : double
-entry : Entry

+getEntry() : Entry
+getRight(): ClusterNode
+getLeft() : ClusterNode
+getDistance() : Double
}

class HierarchicalClusteredData{
-root : ClusterNode
+getRoot(): root

}     class KMeansClustering
{
        -random:Random
        +fit(data: ParsedData, config: Map<String, Object> ): ClusteredData
+getParameters() : Map<String, Object>
+supports(data : ParsedData) : boolean

}                                   class
HierarchicalClustering{
        +fit(data: ParsedData, config: Map<String, Object> ): ClusteredData
+getParameters() : Map<String, Object>
+supports(data : ParsedData) : boolean

}     class ClusterContext
{
        -strategy: ClusterStrategy
        -observers: List<Observer>
        -clusteredData: ClusteredData
        +setClusterStrategy(strategy: ClusterStrategy)
        +executeClustering(data: ParsedData, config : Map<String,Object>): ClusteredData
        +addObserver(o: Observer)
        +removeObserver(o: Observer)
        +notifyObservers()
    }
interface PluginInterface{
getDescription(): String getPluginType():
PluginType  getName(): String
} interface ParserPlugin{
getSupportedFileExtension(): List<String>
getParsedDataType() : Class<? extends ParsedData>
getParameters() : Map<String, Object>
requiresConfiguration() : boolean  getPluginType() :
PluginType
} interface ClusteringPlugin{ Supports(): boolean fit(data :
ParsedData , config : Map<String, Object>) : ClusteredData
requiresConfiguration() : boolean  supportedParsedType() : Class<?
extends ParsedData> getParameters(): Map<String, Object>
getPluginType() : PluginType
} interface VisualizationPlugin{ getSupportedClusteredDataType():
Class<? extends ClusteredData>  createView(data:ClusteredData) :
VisualizationView  requiresConfiguration() : boolean
getParameters(): Map<String, Object> getPluginType() :
PluginType
}
```

```
enum PluginType{
            PARSER,
   CLUSTERING,
   VISUALIZATION
} class
PluginManager{ -
parserPlugins:
List<ParserPlugin>

- clusteringPlugins: List<ClusteringPlugin>
- visualizationPlugins: List<VisualizationPlugin>
+loadPlugins(pluginDirectoryPath : String)
-loadPluginsFromJar(jarFile:File)
-loadIfPlugin(cls:Class<?>)
}



   interface     ClusteredData     {}
class FlatClusteredData {
      -clusters: Map<Centroids, List<Entry>>
      +FlatClusteredData(clusters: Map<Centroids, List<Entry>>)
      +getClusters(): Map<Centroids, List<Entry>>
      +toString(): String
   }

   class Centroids {
      -coordinates: Map<String, Double>
      +getCoordinates(): Map<String, Double>
      +Centroids(coordinates: Map<String, Double>)
   }
```

```
  '    Layer    4:    Distance
interface Distance {
    +calculate(vector1: Map<String, Double>, vector2: Map<String, Double>): double
  }

  class EuclideanDistance {
    +calculate(p1: Map<String, Double>, p2: Map<String, Double>): double
  }

  '   Layer    5:    Dimensionality    Reduction
interface DimensionalityReducer {
    +reduce(entries: List<Entry>, targetDimensions: int): List<Entry>
  }

  class PcaReducer {
    -MAX_GENES: int
    +reduce(entries: List<Entry>, targetDimensions: int): List<Entry>
  }

  '   Layer    6:   Configuration
class KMeansConfig {
    +k(): int
    +maxIterations(): int
    +distance(): Distance
  }

  interface ClusteringConfigDialog<T> {
    +showAndWait(owner: Window): Optional<T>
  }


  class KMeansConfigDialog {
    +showAndWait(owner: Window): Optional<KMeansConfig>
  }


  ' Relationships
  ParserStrategy <|.. SoftParser : implements
  ParserContext --> ParserStrategy : uses
  ParserContext --> "1" ParsedData : returns
  SoftParser --> ParsedData : returns
  GeneExpressionParsedData ..|> ParsedData : implements
  GeneExpressionParsedData --> Entry : contains
GeneExpressionParsedData ..|> EntryBasedData : "implements"
CSVParser ..|> ParserStrategy : implements
CSVParser ..|> ParsedData : returns
VisualizationView <|.. DendrogramView : implements
Observer <|.. DendrogramView : implements


  ClusterStrategy <|.. KMeansClustering : implements
  ClusterContext --> ClusterStrategy : uses
  KMeansClustering --> Centroids : uses
  KMeansClustering --> Distance : uses
  Distance <|.. EuclideanDistance
  KMeansClustering --> ClusteredData : returns
HierarchicalClustering --> ClusterNode : uses
HierarchicalClustering --> HierarchicalClusteredData: uses
HierarchicalClusteredData ..|> ClusteredData : implements
HierarchicalClustering --> ClusteredData : returns
```

```
ClusterStrategy <|.. HierarchicalClustering : implements
    ClusterContext --> ParsedData : uses
    ClusterContext --> ClusteredData : returns
    ClusterContext --> Observer : notifies
    ClusteredData <|.. FlatClusteredData
' All plugin interfaces extend PluginInterface
PluginInterface <|-- ParserPlugin : extends
PluginInterface <|-- ClusteringPlugin : extends
PluginInterface <|-- VisualizationPlugin : extends
ParserPlugin ..> PluginConfigDialog : "getParameters() uses"
ClusteringPlugin ..> PluginConfigDialog : "getParameters() uses"
VisualizationPlugin ..> PluginConfigDialog : "getParameters() uses"

PluginInterface ..> PluginType : returns

PluginManager --> ParserPlugin : manages
PluginManager --> ClusteringPlugin : manages
PluginManager --> VisualizationPlugin : manages
ParserPlugin ..> ParsedData : parse() returns
ClusteringPlugin ..> ClusteredData : fit() returns
VisualizationPlugin ..> VisualizationView : createView() returns
ClusteringController --> ParserContext : uses
ClusteringController --> ClusterContext : manages
ClusteringController --> PluginManager : uses

PickViewController --> ClusteringController : "calls/uses"

    DimensionalityReducer <|.. PcaReducer : implements
    PcaReducer --> Entry : reduces
    PcaReducer ..> ParsedData : gets entries from


    ClusteringConfigDialog <|.. KMeansConfigDialog
    KMeansConfigDialog ---> KMeansConfig : uses
    ClusteringController --> ParserContext : uses
    ClusteringController --> ClusterContext : uses
}
@enduml
```

8.2.5 Figure 6

```
@startuml
actor User participant "PickViewController" as
PVC participant "ClusteringController" as CC
participant "ParserContext" as PC participant
"SoftParser" as SP participant
"GeneExpressionParsedData" as GED participant
"PcaReducer" as PCA participant
"ClusterContext" as CCtx participant
"KMeansClustering" as KM participant
"FlatClusteredData" as FCD participant
"ScatterPlotView" as SPV


User -> PVC: Select TestDummy.soft
PVC -> CC: loadData(file)
CC -> PC: setParser(new SoftParser())
PC -> SP: parse(file)
SP -> GED: \n...construct Entry objects...\nReturn ParsedData
PC --> CC: return ParsedData note over CC, PCA: (Optional:
Apply PCA Reduction)
CC -> PCA: reduce(GeneExpressionParsedData, targetDimensions)
```

PCA --> GED: Return reduced GeneExpressionParsedData

CC -> CCtx: runClustering(reduced data, config)

CCtx -> KM: executeClustering(data, config)

KM -> GED: Process data using K-means logic\n(assign centroids, iterate)

KM --> FCD: Return FlatClusteredData

CCtx -> SPV: notifyObservers(FCD)

SPV -> SPV: displayClusters(FCD)

SPV -> User: Render Scatter Plot

@enduml

8.2.6 Figure 7 @startuml

actor "User" as U participant

"ClusteringController" as CC participant

"ParserContext" as PC participant "SoftParser" as

SP participant "GeneExpressionParsedData" as

GED participant "PluginManager" as PM

participant "CustomClusteringPlugin" as CCP

participant "ClusterContext" as CCtx participant

"FlatClusteredData" as FCD

== Data Loading ==

U -> CC: Select TestDummy.soft

CC -> PC: setParser(new SoftParser())

PC -> SP: parse(file)

SP -> GED: Create Entry objects and assemble into GeneExpressionParsedData

PC --> CC: return ParsedData

== Plugin Discovery and Execution ==

U -> CC: Choose "MyCustomPlugin"

```
CC -> PM: getClustering() // PluginManager retrieves available clustering plugins

PM -> CCP: load "MyCustomPlugin" (custom clustering plugin)

CC -> CCP: fit(ParsedData, config)

CCP -> GED: Process data using custom algorithm logic

CCP --> FCD: Return a custom FlatClusteredData result


== Notification to Views ==

CC -> CCtx: setStrategy(CCP)

CCtx -> U: notifyObservers(FCD)

@enduml
```

8.3 Structure of submission

We have a main package(src/main/java)where the Model, view, controller, plugin, test, plugin, packages live, the model has the entire core logic as outlined in the UML, View has all the view classes and PickView, Controller – houses thin controller but the main one is the ClusteringController which handles all the logic, Plugin package has the main plug-in architecture and the supporting classes, testPlugin was just a package to test our plugins.

Src/main/resources – has all our scenes and views you need JavaFx 21 and above to run all of this – I am using java 21.0.6 version LTS, there are our dataset files, UML and other files used during testing.

Src/test/java – has all of our test classes./

And then there are general files like diary.md pom.xml and README.md at the root level.

VIDEO LINK

https://drive.google.com/file/d/1e2iMt29e3L1I3DEYZgvo2I0s61YGaDVw/view?usp=drive_link