A visualization system for the study of parallelly evolving time-series

Pantelidis Nikolaos

Diploma Thesis

Supervisor: Prof. Panos Vassiliadis

Ioannina, February 2021

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Acknowledgements

I would like to thank Prof. Panos Vassiliadis for his guidance and his patience to me and I would also like to thank my family and my friends for their support.

25/02/2021

Pantelidis Nikolaos

Abstract

The goal of this Diploma Thesis is the development of a parallelly evolving time-series visualization tool. This kind of time-series consist of peer entities and they evolve in a common timeframe. Our system is a generalized recreation of the [Giac15] and it uses its Parallel Lives Diagram to visually demonstrate how a group of entities co-evolve in parallel, over time. To generalize the old system, we introduce a new format that we call intermediate representation, which is employed to convert the different kind of parallelly time-series formats in a uniform representation. This uniform internal representation is then used to visualize the evolution of the group of entities as a 2D matrix, with each entity corresponding to a row, each time-point corresponding to a column and the color intensity of each cell to demonstrate the volume of activity for the combination of entity & time-point. To handle the volume of entities and time-points we can also cluster them into groups and phases, respectively, via an agglomerative clustering mechanism that we have built. The tool allows zooming in and out, reporting of details, exporting the intermediate representation as a file and the visual representation as a png file and importing/exporting the clustered data. Our system is designed in such way that it is easy to be expanded to support more kinds of formats.

Keywords: time-series, visualization, agglomerative clustering, Java, JavaFX

Περίληψη

Ο στόχος αυτής της Διπλωματικής εργασίας είναι η ανάπτυξη ενός εργαλείου οπτικής αναπαράστασης παράλληλα εξελισσόμενων χρονοσειρών. Οι χρονοσειρές αυτού του τύπου αποτελούνται από ομότιμες οντότητες, οι οποίες εξελίσσονται σε ένα κοινό σύστημα χρόνου. Το σύστημα μας είναι μία γενικευμένη υλοποίηση της εργασίας [Giac15] και χρησιμοποιεί το Διάγραμμα Παράλληλων Ζωών για να αναπαραστήσει οπτικά πως μία ομάδα οντοτήτων συν-εξελίσσεται παράλληλα στο πέρασμα του χρόνου. Για να γενικεύσουμε το παλιό σύστημα, παρουσιάζουμε μία νέα αναπαράσταση δεδομένων την ενδιάμεση αναπαράσταση, η οποία χρησιμοποιείται για να μετατρέψουμε τους διάφορους τύπους παράλληλα εξελισσόμενων χρονοσειρών σε μία πιο γενική αναπαράσταση. Αυτή η αναπαράσταση χρησιμοποιείται για να αναπαραστήσουμε οπτικά την εξέλιξη μίας ομάδας οντοτήτων ως έναν 2Δ πίνακα, όπου κάθε οντότητα αντιστοιχεί σε μία γραμμή, κάθε χρονική στιγμή αντιστοιχεί σε μία στήλη και η εντάσεις των χρωμάτων κάθε κελιού αναπαριστούν το μέγεθος της δραστηριότητας για τον συνδυασμό οντότητας και χρονικής στιγμής. Για να μπορέσουμε διαχειριστούμε το μέγεθος των οντοτήτων και χρονικών στιγμών μπορούμε να τις ομαδοποιήσουμε σε ομάδες και φάσεις αντίστοιχα, μέσω ενός μηχανισμού ιεραρχικής ομαδοποίησης (μέθοδος Agglomerative clustering) που υλοποιήσαμε. Το εργαλείο επιτρέπει την μεγέθυνση/ σμίκρυνση του διαγράμματος, την εμφάνιση λεπτομερειών, την αποθήκευση την ενδιάμεσης αναπαράστασης σε αρχείο, την αποθήκευση του διαγράμματος σε ένα .png αρχείο, την εισαγωγή/εξαγωγή των ομαδοποιημένων δεδομένων. Τέλος, το σύστημα μας υλοποιήθηκε με τρόπο που καθιστά εύκολη την επέκταση του για την υποστήριξη περισσότερων τύπων παράλληλα εξελισσόμενων χρονοσειρών.

Λέξεις κλειδιά: χρονοσειρές, οπτική αναπαράσταση, ιεραρχική ομαδοποίηση, Java, JavaFX

Index

Figures Index

Chapter 1. Introduction

1.1 Thesis subject

We live in a digital era and everyday huge amounts of data are produced. These data come in a variety of forms. In this thesis, we are going to focus on a specific type of data, the **parallelly evolving time-series**.

Parallelly evolving time-series are multiple time-series that follow the same rules. They use the same timeframe, they consist of peer entities and their measurements are of the same quantity/metric. Data like these are stocks, software components version control data, history of country indicators (e.g. babies per woman each year) etc.

These data are of great importance because they describe the history of their entities. This history can prove to be very useful. It could help us identify the good and the bad features of the entities and this way we could avoid the same mistakes and even predict the future evolution of this entities.

In 2015, in the University of Ioannina, Giachos Theophanes introduced Parallel Lives Diagram(PLD) as part of his MSc Thesis [Giac15]. A PLD is a strong parallelly evolving time-series visualization tool, it uses a 2D matrix to demonstrate the data. The rows correspond to the entities, the columns correspond to the time-points, and the color intensity of the cells correspond to the volume of activity for the combination of entity and time-point. However, the tool of [Giac15] supports only relational schema evolution data.

Our contributions in this Diploma Thesis are the following:

- The introduction of a generalized parallel time-series format, which we call intermediate representation. Using an intermediate representation for a set of lives of entities that parallelly evolve, allows us to be able to incorporate *any* such dataset (e.g. stocks, evolving relational tables, population data etc.) via the appropriate conversion.

- The development of a system that implements the Parallel Lives Diagram without its one-type restrictions. This system comes with a variety of other features that will be analyzed in the following chapters.
- We also studied the system's performance when converting relational schema evolution data to the intermediate language.

1.2 Thesis structure

In the following chapters we are going to describe the Diploma Thesis in detail.

The second chapter will contain the thesis goal, the summaries of the related work that we studied in the field of time-series visualization, and the system requirements as use cases.

In the third chapter we are going to define the problem and the solution we introduce, by presenting the system design and architecture. This chapter also contains the test cases we used to ensure the correct function of our system, the tools that we used for the system developing, a user guide and some tips to expand our system.

In the fourth chapter, we introduce the experimental methodology that we used to compute the performance of our system and we present and comment our experimental results.

Finally, the fifth chapter will contain a summary of this thesis, our conclusions and a section about the future work that can be made to improve and expand our system.

Chapter 2. Subject description

2.1 Thesis goal

The goal of this Diploma Thesis is to provide a system that will facilitate the simultaneous visual representation of the lives of peer entities, that co-evolve at the same time-frame.

We call this kind of datasets **parallelly evolving time-series** and they consist of three features. There must be a **common time system** for the whole dataset (think of it as the columns), a group of peer-entities that contains the **same kind of entities**, and the measurements of the entities which must be of the **same quantity/metric**.

This kind of data can be stocks, measurements of the same quantity/metric for different entities (e.g. population in different countries of the world from 1900 to 2020), software components version control data like classes, tables of relational schemas etc.

The functionality of the system will include **data conversion** to our format which is called **intermediate representation**. This format is basically our version of parallelly evolving time-series. Its purpose is to make other more complicated data formats (in our case relational schema histories) compatible with our system.

The system also offers the ability to create a **summary of the data** using **clustering methods**. In many cases the data contain too many entities (rows) or too many time-beats that cannot fit in a screen. By using clustering, we can reduce them to the desired size without losing too much information.

Parallel Lives Diagram (PLD) is another feature of the system. It is a custom chart that visualizes parallel time-series in **2-dimensions**. It can be zoomed in/out and exported as PNG file.

Finally, the rows of the data can be sorted by any of our supported **sorting** types (birth date, duration, activity).

2.2 Related work

In this section we describe Plutarch's Parallel Lives of Giachos [Giac15] which is the tool that we want to rebuild from scratch and make its functionality compatible with more kinds of time series. Furthermore, we describe three interesting visualization techniques.

CloudLines of Miloˇs Krstaji´c, Enrico Bertini και Daniel A. Keim [KrBK11], TimeNotes of James Walker, Rita Borgo and Mark W. Jones [WaBJ16] and finally the CodeTimeLines of Adrian Kuhn and Mirko Stocker [KuSt12].

2.2.1 Plutarch's Parallel Lives

Giachos in his MSc Thesis studies the schema evolution in databases[Giac15]. This study is particularly interesting because the changes in tables' schemata can affect the applications that are based on a database. The visualization of the data seems to be a difficult task because these data consist of numerous changes per version of the database.

Giachos in his work introduces two algorithms that help with the summarized representation of a database's history. The first algorithm is a properly modified, to group time moments, agglomerative clustering implementation. It groups the time moments to phases based on their similarity. The second algorithm is also a modified agglomerative clustering method, but it is used to create clusters of schemata(entities) based on their similarity. By creating the summary of the data Giachos is able to fit the large data in a screen without sacrificing significant amount of information.

These two algorithms are implemented in the "Plutarch's Parallel Lives Tool" which we will try to recreate in a more generalized version in this Diploma Thesis.

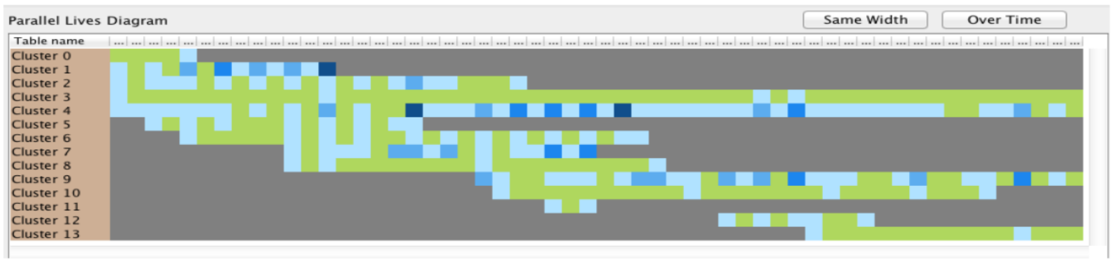


Figure 1: [Giac15] Parallel Lives Diagram

2.2.2 CloudLines

In this work [KrBK11] the authors try to solve some visualization problems about non-equidistant time-series data. This kind of time-series are time-series of a specific news topic. The data are generated from many sources in different periods of time and the objective is to identify when the data are important.

The authors designed this method to handle the following tasks:

- Time-series visualization by granting more screen space to the more recent data so they can be displayed in more detail.
- Important events detection.
- Keep the entire data information and give access in atomic level.

The authors use a method of logarithmic compression to save space from the older data, so they have more available space to display the recent data in more detail.

The important data are detected in dense areas, where they cause overplotting. The authors handle this problem with two functions:

- Importance function
- Cut-off function

By mapping the importance function to the transparency factor and the cut-off function to the dot size they succeeded to create a smoother visualization without the overplotting.

Finally, the authors implemented a magnification lens to provide the user with more details in the dense areas by giving him access to atomic level information of the data.

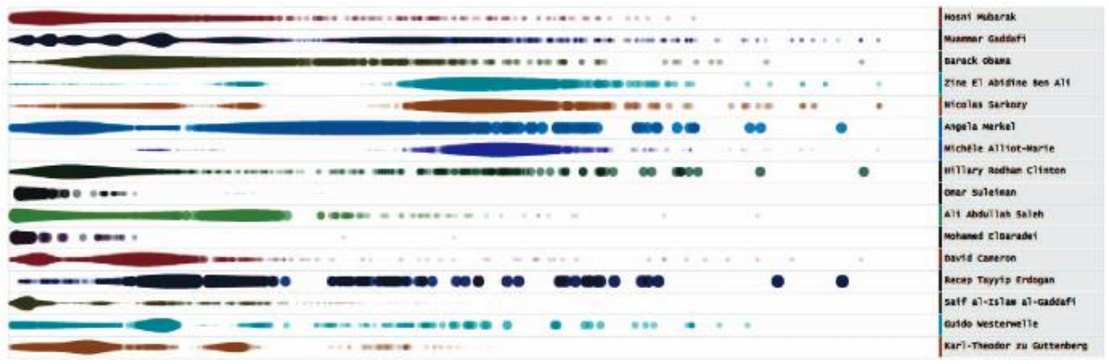


Figure 2: [KrBK11] CloudLines

2.2.3 TimeNotes

TimeNotes [WaBJ16] is a visualization tool that gives the user the ability to choose a specific area of the data to get a more detailed display.

This display is hierarchical in the form of a tree. The tree’s root represents the initial data and the node’s children are the detailed displays of the selected areas. Children nodes are connected to their parents through edges in the form of stripes. In this way they manage smooth transition from the parent nodes to the more detailed children nodes.

The nodes can be moved and resized by the user.

To free up space in the display area, we can also minimize the nodes and the sub-trees to small rectangles that are named bookmarks. User can restore the minimized nodes.

Another feature of the tool is the ability to place many nodes to the same window with overlapping to offer a more direct data comparison.

Finally, the user can create and format notes for every node.

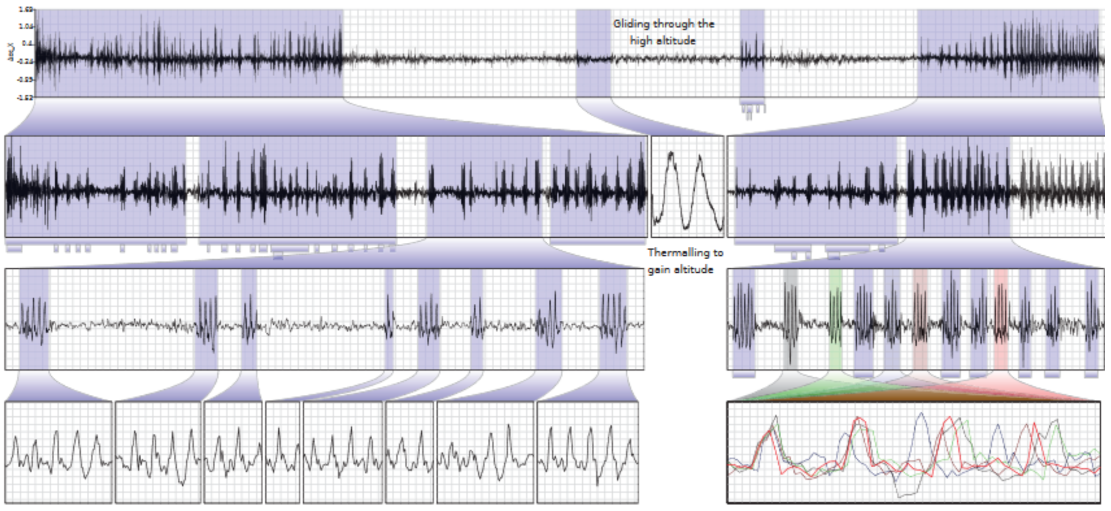


Figure 3: [WaBJ16] TimeNotes

2.2.4 CodeTimeLine

The authors try to solve a problem that comes up when working with software systems, specifically the visual display of the system history to the developers.

The solution that the authors of [KuSt12] suggest, is a pinboard that contains two visualization tools the collaboration view and the sourcecloud flow view.

The collaboration view is visualization technique of software source control data. The columns display all the programmers that have contributed to the software, the rows display the different versions of the software and the dots display the commits.

The sourcecloud flow view use the source code and other changes that occurred in every version. In each version a cloud is formed by the words that were either deleted (red) or added (blue).

The authors add one more level of interactivity in these two applications by giving the user the ability to add notes, photos, email etc. on the events to motivate the users (programmers) to connect their memories or the events with software that they contributed on.

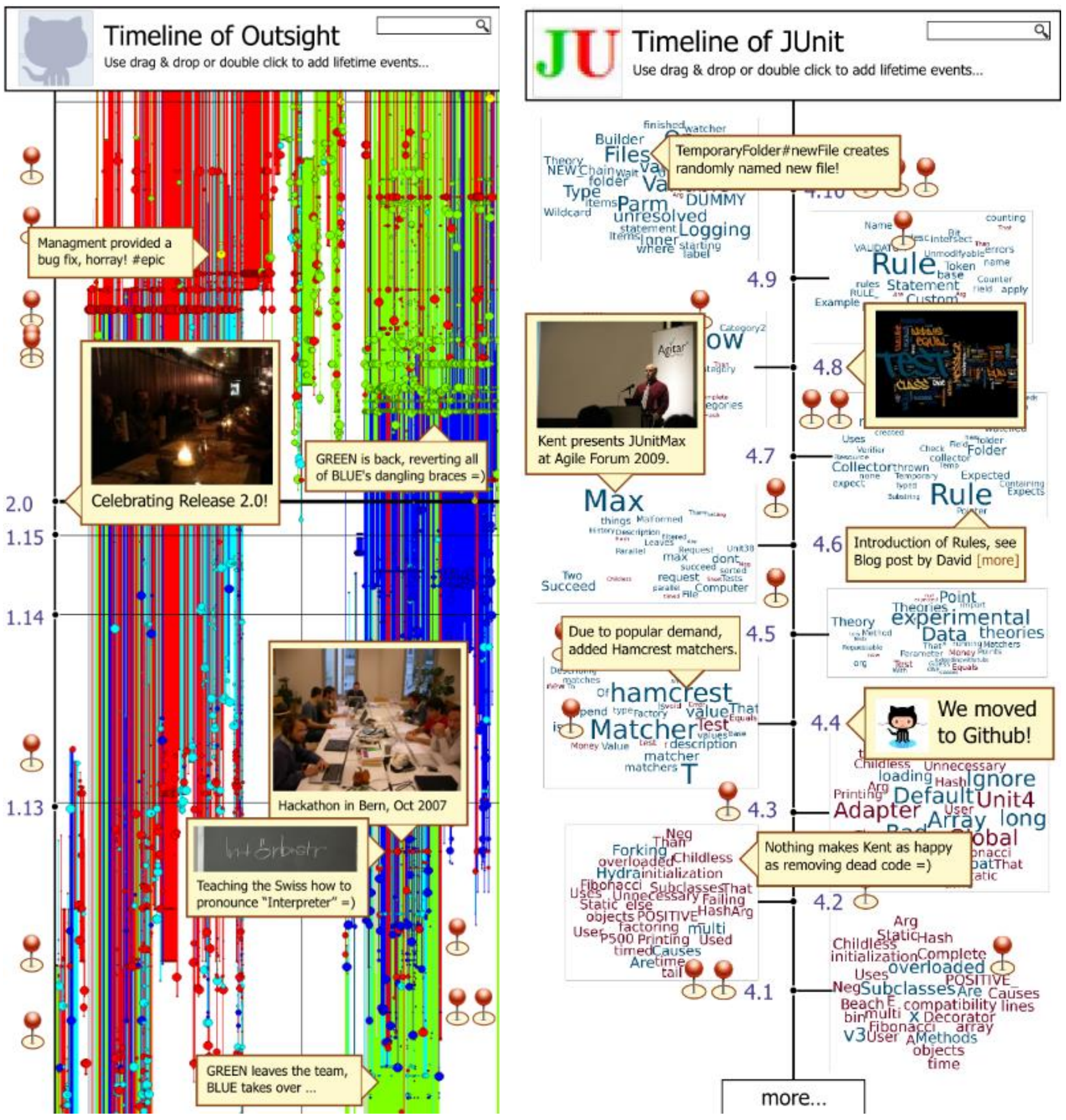

Timeline of Outsight
Use drag & drop or double click to add lifetime events...
Managment provided a bug fix, horray! #epic
Celebrating Release 2.0!
GREEN is back, reverting all of BLUE's dangling braces =)
Hackathon in Bern, Oct 2007
Teaching the Swiss how to pronounce "Interpreter" =)
GREEN leaves the team, BLUE takes over ...
2.0
1.15
1.14
1.13
Timeline of JUnit
Use drag & drop or double click to add lifetime events...
TemporaryFolder#newFile creates randomly named new file!
Kent presents JUnitMax at Agile Forum 2009.
Introduction of Rules, see Blog post by David [more]
Due to popular demand, added Hamcrest matchers.
We moved to Github!
Nothing makes Kent as happy as removing dead code =)
4.9
4.8
4.7
4.6
4.5
4.4
4.3
4.2
4.1
more...

Figure 4: [KuSt12] CodeTimeLine, (Left) Collaboration view, (Right) sourcecloud flow view

2.3 Requirements analysis

This tool will provide the following set of features for the parallel time-series study:

1. Load .tsv/csv datasets of any numerical metric, evolving over time, for a group of peer-entities, to the intermediate representation.
2. Load schema evolution (Hecate) datasets to the intermediate representation.
3. Export the intermediate representation to a .tsv file.
4. Summarize data using clustering methods in both rows(entities) and columns(time beats).
5. Visualize data using parallel lives diagram(pld).

6. Export the diagram as pictures (png).
7. Zoom in and out the diagram.
8. Get the details of a specific phase.
9. Get the details of a specific entity group.
10. Sort rows by birth date or life duration or activity (ascending or descending).
11. Import a data snapshot from a specific folder (tem.tsv, gpm.tsv).
12. Export a snapshot of our loaded data (intermediate and summarized representations) to a specific folder (tem.tsv, gpm.tsv)

Use case: Load tsv/csv dataset to intermediate language
ID: UC1
Actors(id): Analyst(A1)
Preconditions: 1. Datasets are in a supported format (csv, tsv)
Flow of events: 1. The use case starts when the Analyst imports the path of the dataset in the system. 2. The system detects the dataset's type. 3. If the type is supported and equals either csv or tsv: 3.1. The system initializes the suitable loader. 3.2. The system parses the dataset 3.3. The system loads the file in the intermediate representation. 3.4. The system returns the intermediate representation. 4. If the type is not supported: 4.1. System prints an error message, and the use case ends.
Postconditions: The dataset is loaded on the system.

Figure 5: UC1 Load tsv/csv dataset to intermediate representation

Use case: Load schema evolution dataset to intermediate language

ID: UC2

Actors(id): Analyst (A1)

Preconditions:

1. The given dataset folder contains a folder named “results” that contains the following files: SchemaHeartbeat.tsv, tables_DetailedStats.tsv, transitions.csv

Flow of events:

1. The use case starts when the Analyst imports the path of the dataset in the system.
2. The system detects the dataset’s type.
3. If the type is supported and equals schema_evo:
    - 3.1. The system initializes the custom SchemaEvoLoader.
    - 3.2. The system parses the dataset’s files.
    - 3.3. The system aggregates the transactions to measurements.
    - 3.4. The system loads the measurements in the intermediate representation.
    - 3.5. The system returns the intermediate language.
    - 3.6. The system initializes the UC3 with output path project_folder/figures/PPL.
4. If the type is not supported:
    - 4.1. System prints an error message, and the use case ends.

Postconditions: The dataset is loaded on the system.

Figure 6: UC2 Load schema evolution dataset to intermediate representation

Use case: Export intermediate representation to a tsv file
ID: UC3
Actors(id): Analyst(A1)
Preconditions: 1. The system is loaded with a dataset.
Flow of events: 1. The use case starts when the Analyst imports the path of the output file in the system. 2. The system generates a string from the loaded data. 3. The system writes the generated string to given file path.
Postconditions: The intermediate representation is stored in a .tsv file.

Figure 7: UC3 Export intermediate representation to a tsv file

Use case: Summarize data
ID: UC4
Actors(id): Analyst(A1)
Preconditions: 1. The system is loaded with a dataset. 2. The given clustering profile has valid values.
Flow of events: 1. The use case starts when the Analyst creates a clustering profile. 2. The A1 imports the clustering profile to the system. 3. The system initializes the selected PhaseExtractor. 4. The system cluster time-beats and generates phases. 5. The system initializes the selected EnityGroupExtractor. 6. The system cluster entities and generates entity groups. 7. The system converts the TimeEntityMeasurements (intermediate representation) to GroupPhaseMeasurements (summarized representation)
Postconditions: A new summarized representation of the data is stored in the system.

Figure 8: UC4 Summarize data

Use case: Visualize data using PLD
ID: UC5
Actors(id): Analyst(A1)
Preconditions: 1. The system is loaded with a summarized representation of a dataset.
Flow of events: 1. The use case starts when the Analyst clicks the “show pld” button. 2. The system loads data on the parallel lives diagram. 3. The system shows the diagram.
Postconditions: The system shows the data using a pld.

Figure 9: UC5 Visualize data using PLD

Use case: Save PLD as a png
ID: UC6
Actors(id): Analyst(A1)
Preconditions: 1. The system is showing the data using a pld.
Flow of events: 1. The use case starts when the Analyst clicks the “save screenshot” button. 2. The system asks the Analyst to enter the output path. 3. The Analyst enters an output path. 4. The system saves the diagram as a .png file in the output path.
Postconditions: The diagram is saved as a png file in the output path.

Figure 10: UC6 Save PLD as a png

Use case: Zoom in and out the PLD
ID: UC7
Actors(id): Analyst(A1)
Preconditions: 1. The system is showing the data using a pld.
Flow of events: 1. The use case starts when the Analyst places the mouse in the pld area and scrolls up or down. 2. If the Analyst scrolls down. 2.1. The system zooms out the diagram. 3. If Analyst scrolls up. 3.1. The system zooms in the diagram.
Postconditions: The diagram is zoomed in or out.

Figure 11: UC7 Zoom in and out the PLD

Use case: Get the details of a specific phase
ID: UC8
Actors(id): Analyst(A1)
Preconditions: 1. The system is showing the data using a pld.
Flow of events: 1. The use case starts when the Analyst right clicks on a phase cell and selects "show details". 2. The system retrieves the details of the selected phase. 3. The system shows a panel with the phase details.
Postconditions: A pane with the details of the selected phase is created.

Figure 12: UC8 Get the details of a specific phase

Use case: Get the details of a specific entity group
ID: UC9
Actors(id): Analyst(A1)
Preconditions: 1. The system is showing the data using a pld.
Flow of events: 1. The use case starts when the Analyst clicks on an entity group cell. 2. The system retrieves the details of the selected entity group. 3. The system shows a panel with the entity group details.
Postconditions: A pane with the details of the selected entity group is created.

Figure 13: UC9 Get the details of a specific entity group

Use case: Sort summarized data using one of the available sorting types
ID: UC10
Actors(id): Analyst(A1)
Preconditions: 1. The system is loaded with a summarized representation of a dataset.
Flow of events: 1. The use case starts when the Analyst selects one of the available sorting options. 2. The system sorts the data by the selected sorting option. 3. If the data are already displayed in the PLD: 3.1. The displayed data are sorted automatically.
Postconditions: The system has a sorted summarized representation available for use.

Figure 14: UC10 Sort summarized data using one of the available sorting types

Use case: Import data snapshot from files (tem & gpm)

ID: UC11

Actors(id): Analyst(A1)

Preconditions:

1. The folder that contains the snapshot contains the files (tem.tsv & gpm.tsv)

Flow of events:

1. The use case starts when the Analyst enters the path of the folder that contains the snapshot.
2. The system executes UC1 for the tem.tsv file.
3. The system parses the gpm.tsv.
4. The system combines the intermediate representation with the parse data from the gpm.tsv to generate the phases and the entity groups.
5. The system converts the load TimeEntityMeasurements to GroupPhaseMeasurements using the phases and the entity groups.

Postconditions: The system has loaded a summarized version of the data.

Figure 15: UC11 Import data snapshot from files (tem & gpm)

Use case: Export data snapshot to files (tem & gpm)

ID: UC12

Actors(id): Analyst(A1)

Preconditions:

1. The system is loaded with a summarized representation of a dataset.
2. The folder path is valid

Flow of events:

1. The use case starts when the Analyst enters the path of an output folder.
2. The system executes UC3 with path the file (tem.tsv) in the given folder path.
3. The system writes another file (gpm.tsv) in the given folder that contains the indexes of the phase and entity components.

Postconditions: The given folder contains a file tem.tsv (intermediate representation) and a file gpm.tsv (with the necessary data to reconstruct summarized representation)

Figure 16: UC12 Export data snapshot to files (tem & gpm)

Chapter 3. Design & Implementation

3.1 Problem definition and solution

The first problem that we faced when we started designing our system was finding the common denominator between the different parallel time-series formats. On the one hand we have the schema evolution datasets which are relatively complicated data and on the other hand we have some GapMinder datasets which have a simple format.

So, we came up with a generalized format which we call intermediate representation format. This format consists of four key classes.

1. **Beats:** A common timeframe that consists of unique time beats and represent the columns.
2. **Entities**: Peer entities that represent the rows.
3. **Measurements:** The common quantity/metric measurements that represent the cell values.
4. **TimeEntityMeasurement(TEM):** The combination of an entity, a beat, and a measurement.

The challenge is to convert the relation schema evolution data to the intermediate representation. We solved this by aggregating the transactions of each entity in each time beat, thus we generated TimeEntityMeasurement for schema evolution data.

Since we were able to convert such a complex format to intermediate representation, we assume that other parallel time-series formats can also be converted.

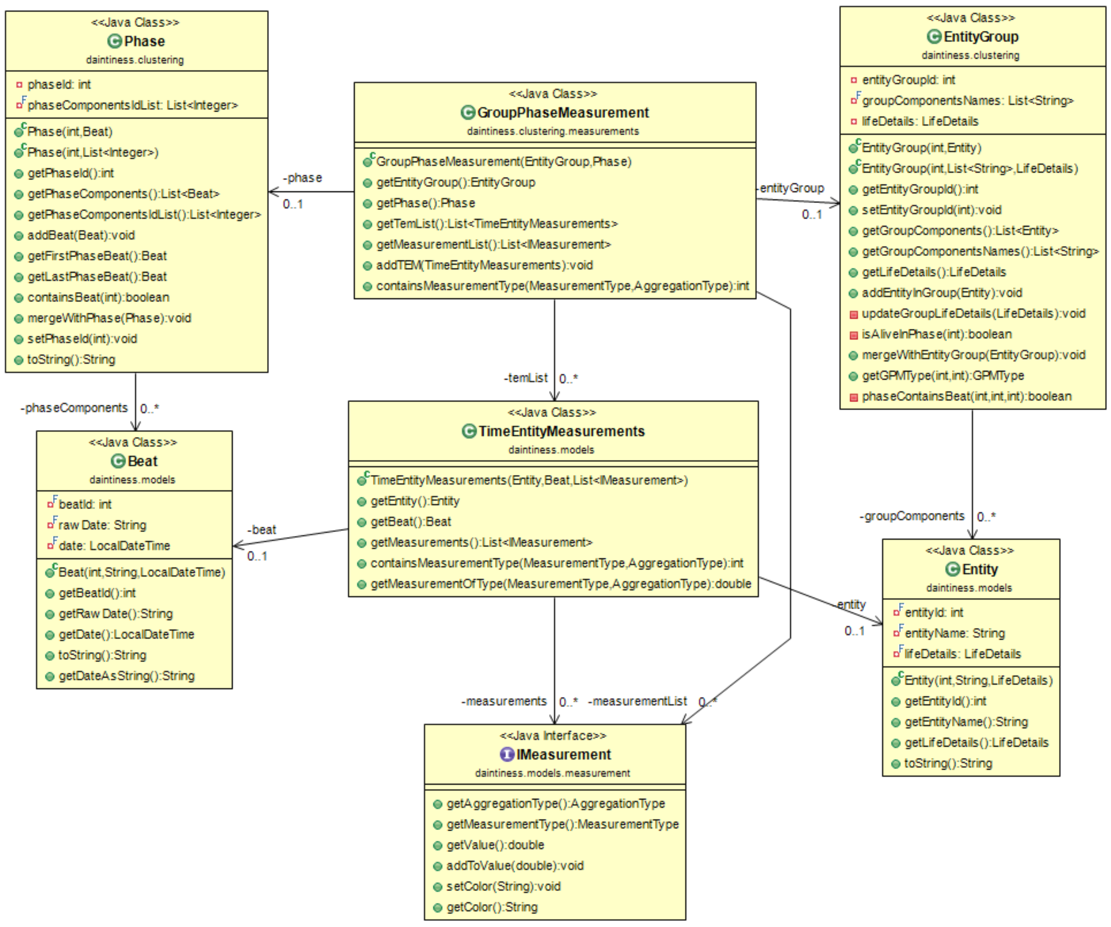
The next challenge we faced was to develop the system in a way that is easily expandable. After all, our goal was to make a universal parallelly evolving time-series visualization tool. Of course, we could not add support for all the possible parallelly evolving time-series formats. So, we tried to make it easy for the developer to add his custom loader/converter to our system in cases he wanted to use our tool with an unsupported format. We created the ILoader interface that sets the rules that must be followed by the programmer to add support for a new format.

Finally, we added the option to summarize data using clustering methods like in [Giac15], to be able to fit the data in a screen because this kind of data tends to be big. In our implementation we use the following terminology for the summarized data:

1. **Phase:** A phase is a group of consecutive Beats.

2. **EntityGroup:** A group of Entities.
3. **GroupPhaseMeasurement(GPM):** Sum of all the TimeEntityMeasurements that map to an EntityGroup and a Phase.

We create the summaries using agglomerative clustering. In the following class diagram, we can identify the relations between the intermediate and the summarized representation.



<<Java Class>>
Phase
daintiness.clustering
phaseId: int
phaseComponentsIdList: List<Integer>
Phase(int,Beat)
Phase(int,List<Integer>)
getPhaseId():int
getPhaseComponents():List<Beat>
getPhaseComponentsIdList():List<Integer>
addBeat(Beat):void
getFirstPhaseBeat():Beat
getLastPhaseBeat():Beat
containsBeat(int):boolean
mergeWithPhase(Phase):void
setPhaseId(int):void
toString():String
<<Java Class>>
GroupPhaseMeasurement
daintiness.clustering.measurements
GroupPhaseMeasurement(EntityGroup,Phase)
getEntityGroup():EntityGroup
getPhase():Phase
getTemList():List<TimeEntityMeasurements>
getMeasurementList():List<IMeasurement>
addTEM(TimeEntityMeasurements):void
containsMeasurementType(MeasurementType,AggregationType):int
<<Java Class>>
EntityGroup
daintiness.clustering
entityGroupId: int
groupComponentsNames: List<String>
lifeDetails: LifeDetails
EntityGroup(int,Entity)
EntityGroup(int,List<String>,LifeDetails)
getEntityGroupId():int
setEntityGroupId(int):void
getGroupComponents():List<Entity>
getGroupComponentsNames():List<String>
getLifeDetails():LifeDetails
addEntityInGroup(Entity):void
updateGroupLifeDetails(LifeDetails):void
isAliveInPhase(int):boolean
mergeWithEntityGroup(EntityGroup):void
getGPMType(int,int):GPMType
phaseContainsBeat(int,int,int):boolean
-phase
0..1
-entityGroup
0..1
-temList
0..*
-phaseComponents
0..*
<<Java Class>>
Beat
daintiness.models
beatId: int
rawDate: String
date: LocalDateTime
Beat(int,String,LocalDateTime)
getBeatId():int
getRawDate():String
getDate():LocalDateTime
toString():String
getDateAsString():String
<<Java Class>>
TimeEntityMeasurements
daintiness.models
TimeEntityMeasurements(Entity,Beat,List<IMeasurement>)
getEntity():Entity
getBeat():Beat
getMeasurements():List<IMeasurement>
containsMeasurementType(MeasurementType,AggregationType):int
getMeasurementOfType(MeasurementType,AggregationType):double
-beat
0..1
-groupComponents
0..*
-entity
0..1
<<Java Class>>
Entity
daintiness.models
entityId: int
entityName: String
lifeDetails: LifeDetails
Entity(int,String,LifeDetails)
getEntityId():int
getEntityName():String
getLifeDetails():LifeDetails
toString():String
-measurements
0..*
-measurementList
0..*
<<Java Interface>>
IMeasurement
daintiness.models.measurement
getAggregationType():AggregationType
getMeasurementType():MeasurementType
getValue():double
addToValue(double):void
setColor(String):void
getColor():String

Figure 17: Class diagram of intermediate and summarized representation models

3.2 Software design and architecture

In this section, we describe the core components of the system and the dependencies between them through the package, the class diagrams, and the class descriptions.

The system's backend consists of **three main interfaces the IDataHandler, the IFileHandler and the IClusteringHandler**. In a nutshell, the IDataHandler keeps the data in intermediate representation, the IFileHandler loads and convert the supported file formats to intermediate representation, and is responsible for storing the data to files, the IClusteringHandler applies clustering methods on the data to create a summarized representation of them. Moreover, there is the IMainController which works like an API, it exports all the main features of the system by combining the core backend components.

The frontend's main component is the Controller class that connects the backend with the frontend.

3.2.1 Software Architecture

The main engine consists of six packages plus three other application packages.

The main packages are the following:

- Maincontroller package contains the API of the backend.
- Models package contains the intermediate representation models.
- IO package contains the file handling code (read/write/convert).
- Data package contains the intermediate representation handler.
- Clustering package contains all the necessary for the summarization of the data code (clustering, models, structures).
- Utilities is a package that contains some globally used enums.

The application packages are:

- Gui package which contains the desktop application.
- App package which contains a command line tool that generates and exports the intermediate representation of the given dataset.
- Experiments package which contains a script that we used to time the tool.

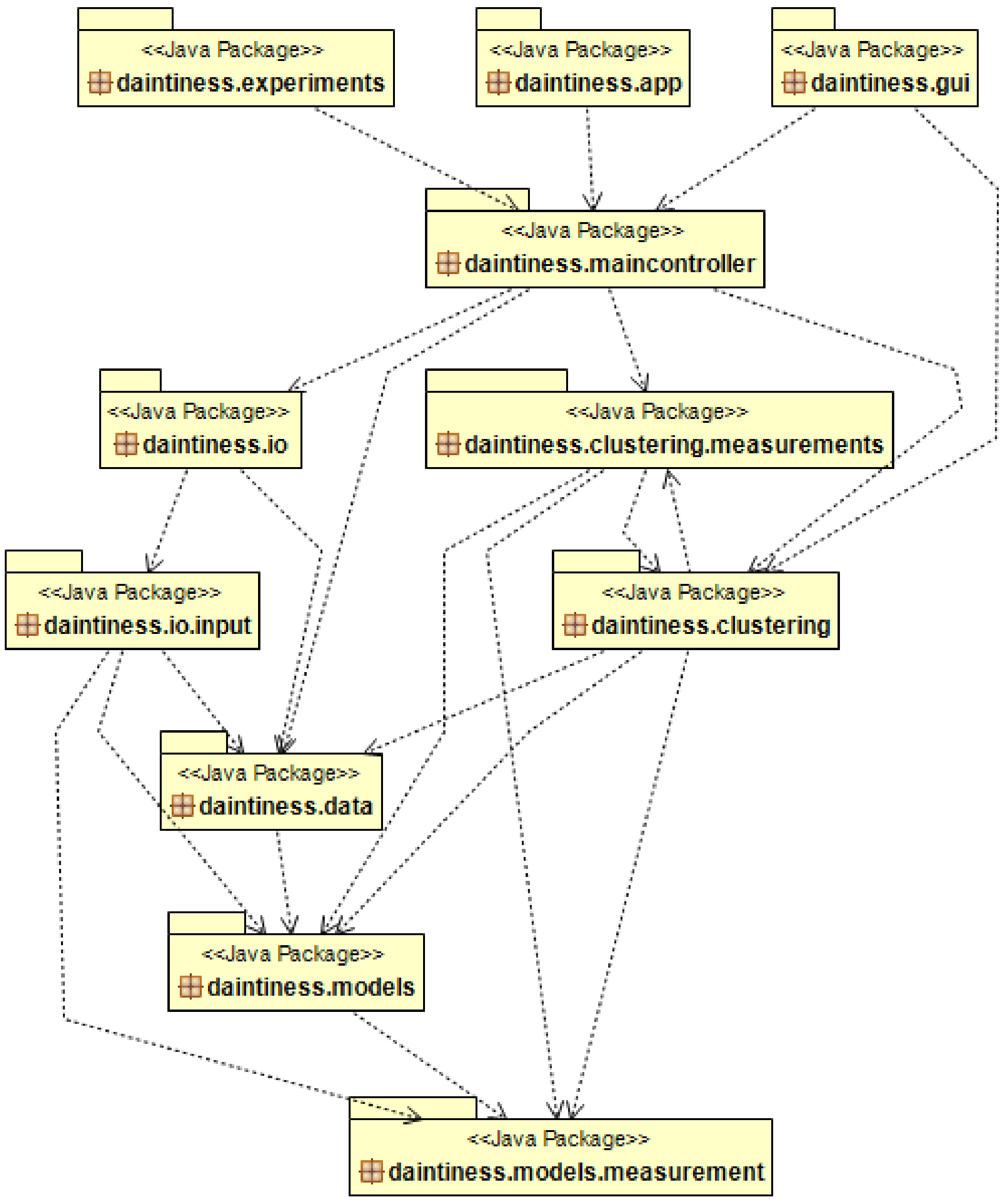


Figure 18: Package diagram

3.2.2 Backend Description

3.2.2.1 Data Models

The system uses a variety of models for the different data representations. These models could be grouped as Intermediate Representation models, Summarized Representation models and General-Purpose Models.

Entity: this class holds the main information about the entities which are the rows of our data

Beat: this class represents the time unit of our data and we could also think of it as the header of its column of our data

LifeDetails: this class describes the life details of Entities and EntityGroups.

IMeasurement: this interface holds the information(value, color, type) of a cell, there are three classes that implement this interface.

- Measurement: this kind of measurement represents the value of a non-empty cell.
- EmptyMeasurement: this measurement holds the information about the cells without value. A cell can be without value because it is a birth or death or active(the entity is alive but there is no value) or inactive (the entity is either dead or yet to be born) cell.

MeasurementFactory: this class helps the generation of IMeasurement type of classes.

TimeEntityMeasurement(TEM): this class packs an Entity, a Beat, and a list of the available IMeasurements. It is the core component of the intermediate representation. It holds all the necessary info of a cell. The Entity represents the row, the Beat represents the column and the IMeasurement represents the value of the cell.

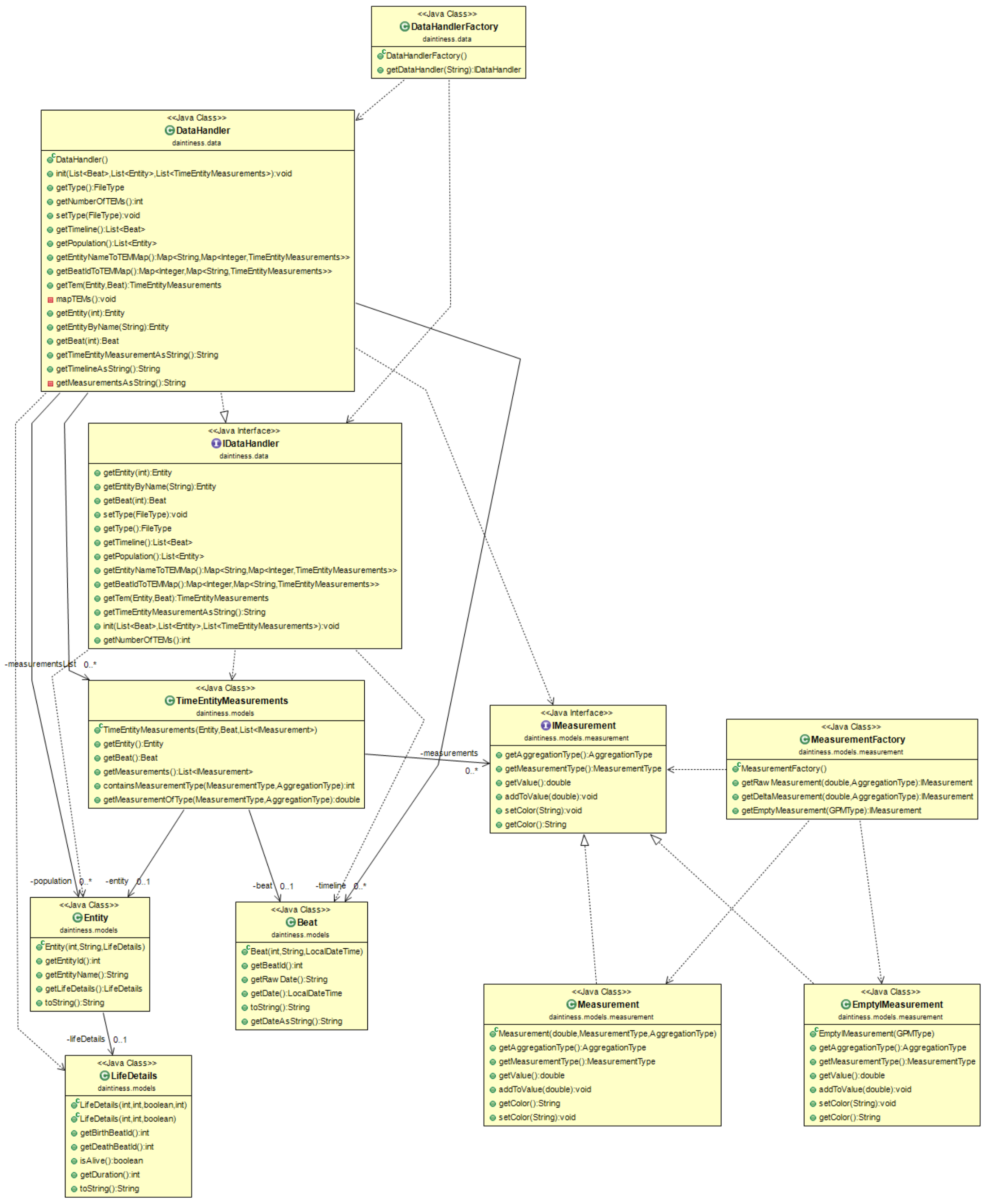
EntityGroup: this class is produced by the entity clustering; it represents a generated group of entities.

Phase: this class is produced by the beat clustering; it represents a generated group of beats.

GroupPhaseMeasurement(GPM): this class packs an EntityGroup, a Phase and a list of the available IMeasurements; it is created by aggregating the TEMs that are contained by the EntityGroup and Phase components.

GroupMeasurements: this class maps an EntityGroup with its GPMs.

ChartGroupPhaseMeasurement: this class is a simplified form of the GroupMeasurements class; the difference is that ChartGroupPhaseMeasurement holds only a specified type of IMeasurements; it is used in the GUI, in file exporting and in data sorting.



<<Java Class>>
DataHandlerFactory
daintiness.data
DataHandlerFactory()
getDataHandler(String):IDataHandler
<<Java Class>>
DataHandler
daintiness.data
DataHandler()
init(List<Beat>,List<Entity>,List<TimeEntityMeasurements>):void
getType():FileType
getNumberOfTEMs():int
setType(FileType):void
getTimeline():List<Beat>
getPopulation():List<Entity>
getEntityNameToTEMMap():Map<String,Map<Integer,TimeEntityMeasurements>>
getBeatIdToTEMMap():Map<Integer,Map<String,TimeEntityMeasurements>>
getTem(Entity,Beat):TimeEntityMeasurements
mapTEMs():void
getEntity(int):Entity
getEntityByName(String):Entity
getBeat(int):Beat
getTimeEntityMeasurementAsString():String
getTimelineAsString():String
getMeasurementsAsString():String
<<Java Interface>>
IDataHandler
daintiness.data
getEntity(int):Entity
getEntityByName(String):Entity
getBeat(int):Beat
setType(FileType):void
getType():FileType
getTimeline():List<Beat>
getPopulation():List<Entity>
getEntityNameToTEMMap():Map<String,Map<Integer,TimeEntityMeasurements>>
getBeatIdToTEMMap():Map<Integer,Map<String,TimeEntityMeasurements>>
getTem(Entity,Beat):TimeEntityMeasurements
getTimeEntityMeasurementAsString():String
init(List<Beat>,List<Entity>,List<TimeEntityMeasurements>):void
getNumberOfTEMs():int
-measurementsList 0..*
<<Java Class>>
TimeEntityMeasurements
daintiness.models
TimeEntityMeasurements(Entity,Beat,List<IMeasurement>)
getEntity():Entity
getBeat():Beat
getMeasurements():List<IMeasurement>
containsMeasurement(MeasurementType,AggregationType):int
getMeasurementOfType(MeasurementType,AggregationType):double
-measurements 0..*
<<Java Interface>>
IMeasurement
daintiness.models.measurement
getAggregationType():AggregationType
getMeasurementType():MeasurementType
getValue():double
addToValue(double):void
setColor(String):void
getColor():String
<<Java Class>>
MeasurementFactory
daintiness.models.measurement
MeasurementFactory()
getRawMeasurement(double,AggregationType):IMeasurement
getDeltaMeasurement(double,AggregationType):IMeasurement
getEmptyMeasurement(GPMType):IMeasurement
-population 0..*
-entity 0..1
-beat 0..1
-timeline 0..*
<<Java Class>>
Entity
daintiness.models
Entity(int,String,LifeDetails)
getEntityId():int
getEntityName():String
getLifeDetails():LifeDetails
toString():String
<<Java Class>>
Beat
daintiness.models
Beat(int,String,LocalDateTime)
getBeatId():int
getRawDate():String
getDate():LocalDateTime
toString():String
getDateAsString():String
-lifeDetails 0..1
<<Java Class>>
LifeDetails
daintiness.models
LifeDetails(int,int,boolean,int)
LifeDetails(int,int,boolean)
getBirthBeatId():int
getDeathBeatId():int
isAlive():boolean
getDuration():int
toString():String
<<Java Class>>
Measurement
daintiness.models.measurement
Measurement(double,MeasurementType,AggregationType)
getAggregationType():AggregationType
getMeasurementType():MeasurementType
getValue():double
addToValue(double):void
getColor():String
setColor(String):void
<<Java Class>>
EmptyIMeasurement
daintiness.models.measurement
EmptyIMeasurement(GPMType)
getAggregationType():AggregationType
getMeasurementType():MeasurementType
getValue():double
addToValue(double):void
setColor(String):void
getColor():String

Figure 19: Intermediate representation models' diagram

3.2.2.2 FileHandler Logic

This group of classes are responsible for the data input, output and the conversion of them to the intermediate representation. It supports a variety of file formats.

FileHandlerFactory: this factory class helps us generate IFileHandler type of classes.

IFileHandler: this interface sets the rules of a FileHandler; it is created this way to give the option to other programmers to use different kind of FileHandlers in our tool.

FileHandler: this class is our implementation of IFileHandler; it is responsible for the data input and output.

CsvReader: this class is responsible for the csv/tsv files parsing; it returns a list (rows) of String arrays(columns).

LoaderFactory: this factory class helps us generate ILoader type of classes.

ILoader: this class is a simple interface that our Loaders use.

SimpleLoader: this class implements ILoader and is used to load csv and tsv files directly as intermediate representation; it also gets custom separators as arguments.

SchemaEvoLoader: this implementation of the ILoader is used to parse schema evo datasets; it parses three files that are contained in the input folder file (SchemaHeartbeat.tsv, tables_DetailedStats.tsv, transitions.csv); SchemaHeartbeat.tsv contains the timeline data (beats); tables_DetailedStats.tsv contains the population data (entities); transitions.tsv contains all the transitions that have occurred in the database's lifetime (these data are converted to TimeEntityMeasurements by aggregating them by transaction type)

TableBeatMetrics: this class is used only by SchemaEvoLoader; it stores the number of different type of transactions that occurred to an entity in specific moment(beat).

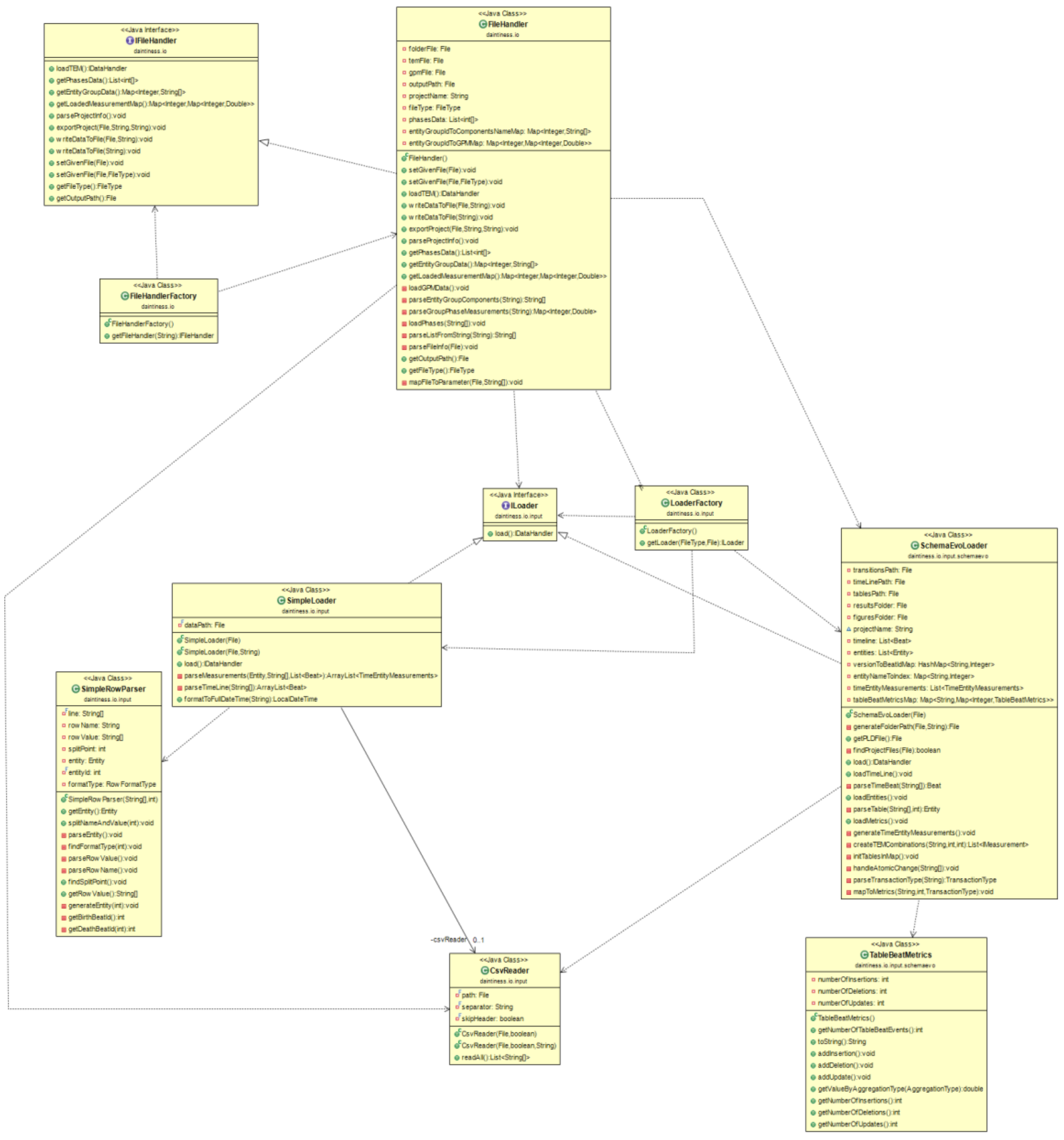


Figure 20: Class diagram for package io

3.2.2.3 DataHandler Logic

This set of classes store and give us access to the intermediate representation.

DataHandlerFactory: this factory class generate IDataHandler implementations.

IDataHandler: this interface set the rules to create different kind of DataHandlers that are compatible with our tool.

DataHandler: this class is our IDataHandler implementation; this class keeps the intermediate representation and give as access to it through a variety of different data structures.

3.2.2.4 ClusteringHandler Logic

This set of classes generate a summarized version of the intermediate representation using clustering methods.

ClusteringHandlerFactory: this factory class helps us generate IClusteringHandler type of classes.

IClusteringHandler: this interface sets the rules of ClusteringHandler to be compatible with our system.

ClusteringHandler: this is our IClusteringHandler implementation; it combines the necessary components for both the beat and the entity clustering.

ClusteringProfile: this class packs a EntityClusteringProfile and BeatClusteringProfile object; in other words, it contains all the necessary parameters for the clustering methods.

EntityClusteringProfile: this class contains the necessary parameters for the entity clustering method (desired number of entity groups & weights).

BeatClusteringProfile: this class contains the necessary parameters for the beat clustering method (desired number of phases & weights).

IEntityGroupExtractor: this interface sets the rules that an EntityGroupExtractor must follow.

- **AgglomerativeEntityGroupExtractor:** this is our implementation of the IEntityGroupExtractor interface and offers agglomerative clustering for the Entities.

EntityGroupExtractorFactory: this class helps us generate objects of the IEntityGroupExtractor implementations.

IPhaseExtractor: this interface sets the rules that a PhaseExtractor must follow.

- **AgglomerativePhaseExtractor:** this is our implementation of the IPhaseExtractor interface and offers agglomerative clustering for the time Beats.

PhaseExtractorFactory: this class helps us generate objects of the IPhaseExtractor implementations.

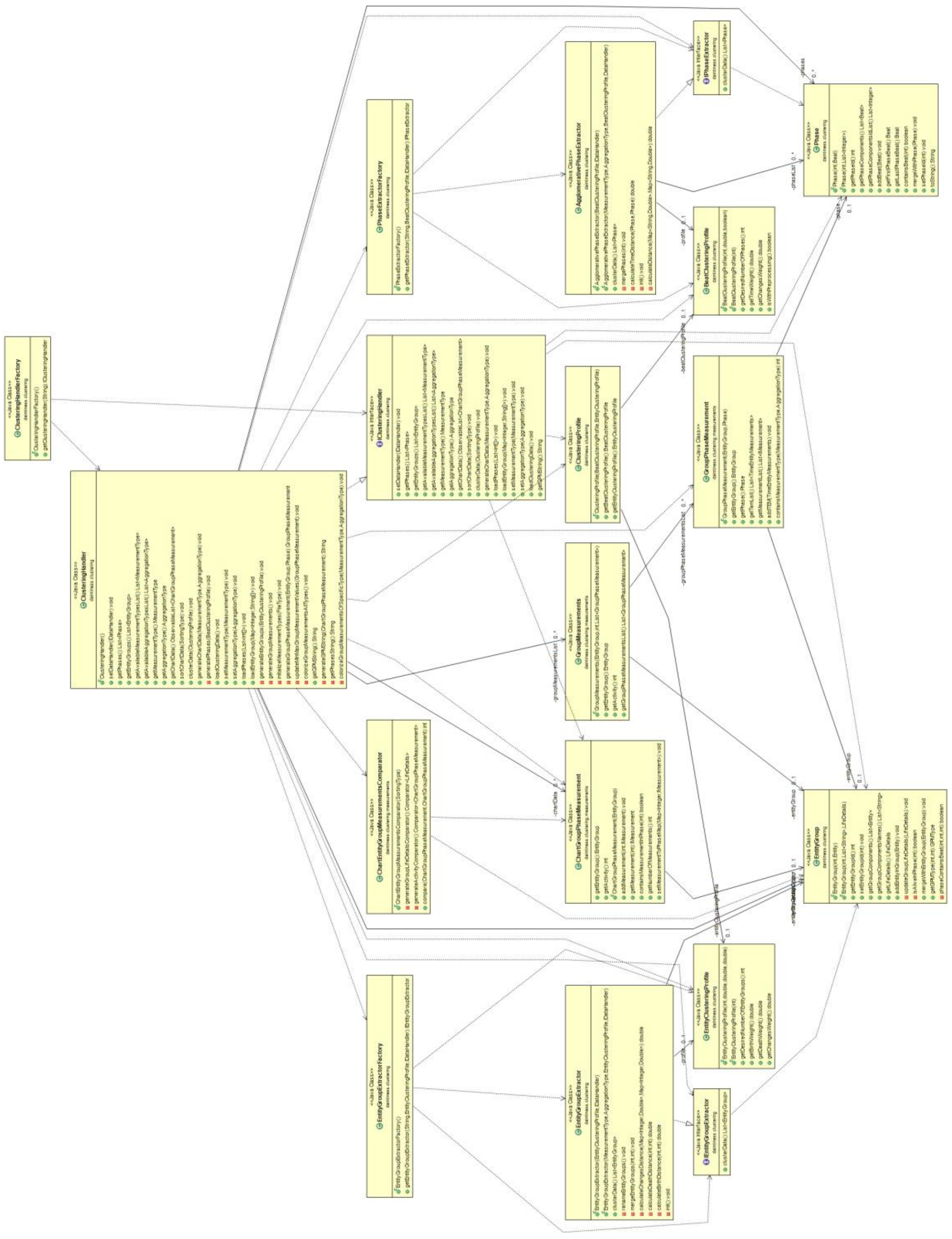


Figure 21: Class diagram of the package clustering

3.2.3 Frontend Description

Our GUI consists of the following set of classes and fxml files.

Main: this is main class of the desktop application; it initializes the app by loading the Scene.fxml file.

Scene.fxml: this fxml file contains the basic design of the main window.

Controller: this class is responsible for the communication between the main engine API and the gui; it also creates and shows dynamically some extra graphic components of the main window.

PLDiagram: this is a class extends the javafx ScrollPane class; it contains the custom diagram (Parallel Lives Diagram) that we created using the TableView class; this diagram was created to visualize the data of parallel time-series.

ClusteringProfileDialog.fxml: this file contains the design of the dialog that we created to enter the clustering parameters to the application.

ClusteringProfileDialogController: this class is responsible for the data initialization and handling of the ClusteringProfileDialog.

EntityGroupDetails: this class creates ScrollPane that shows the details of a given EntityGroup; it is used when we click on an EntityGroup on the PLD.

PhaseDetails: this class creates ScrollPane that shows the details of a given Phase; it is used when we right click on a Phase on the PLD.

3.3 Software testing design

In this section, we describe the testing methodology we used on our system as well as the test cases that we created.

3.3.1 Testing Methodology

To test our system, we used the black box testing methodology. We created tests that check the systems correct functioning in common and in extreme cases.

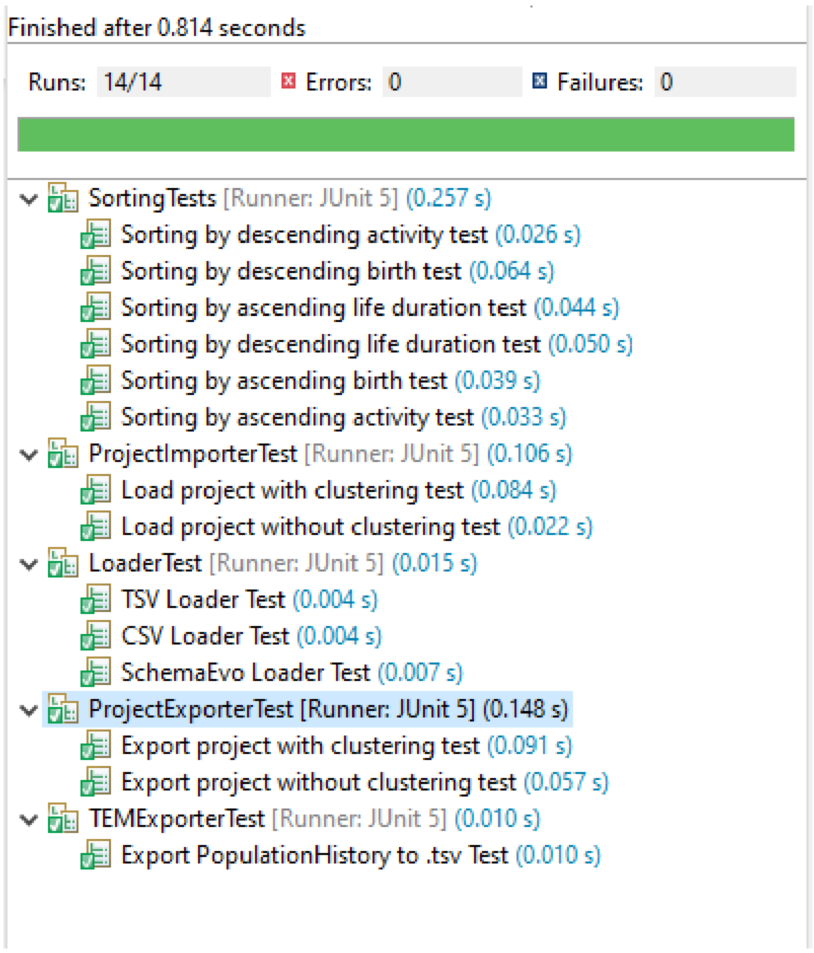
All the tests that we created compare the expected output with the produced output given a specific input. In some cases, we created our own data for the testing purposes.

3.3.2 Detailed test description

For the testing we used "Junit 5" and we have created the following test cases:

- **LoaderTest:** This test was created to check the correct loading and the conversion to intermediate language. We tested three different input formats (tsv, csv and schema evo data). We created mock datasets and we hard-coded the expected values in the unit test.

- **TEMExporterTest:** In this test, we write the intermediate representation to a file and we load it back. Then we compare the initial data with the retrieved data.
- **ProjectImporterTest:** In this test, we load a dataset snapshot (tem.tsv & gpm.tsv) and we compare the expected summarized representation with the loaded one. We test a dataset both with and without clustering.
- **ProjectExporterTest:** In this test, we write a snapshot of the dataset to files (tem.tsv & gpm.tsv) and we try to retrieve it. Again, we test a dataset with clustering and one without.
- **SortingTests:** In this test, we test all the available sorting types. We load and sort the dataset with each one of the available sorting types. We test them by comparing each line with the next one.



```
Finished after 0.814 seconds

Runs: 14/14    Errors: 0    Failures: 0

SortingTests [Runner: JUnit 5] (0.257 s)
    Sorting by descending activity test (0.026 s)
    Sorting by descending birth test (0.064 s)
    Sorting by ascending life duration test (0.044 s)
    Sorting by descending life duration test (0.050 s)
    Sorting by ascending birth test (0.039 s)
    Sorting by ascending activity test (0.033 s)
ProjectImporterTest [Runner: JUnit 5] (0.106 s)
    Load project with clustering test (0.084 s)
    Load project without clustering test (0.022 s)
LoaderTest [Runner: JUnit 5] (0.015 s)
    TSV Loader Test (0.004 s)
    CSV Loader Test (0.004 s)
    SchemaEvo Loader Test (0.007 s)
ProjectExporterTest [Runner: JUnit 5] (0.148 s)
    Export project with clustering test (0.091 s)
    Export project without clustering test (0.057 s)
TEMExporterTest [Runner: JUnit 5] (0.010 s)
    Export PopulationHistory to .tsv Test (0.010 s)
```

Figure 22: Unit tests

3.4 Technical details and user guide

In this section, we list the technologies that we used to develop the system and we describe the installation and user guides.

3.4.1 Technical details

For the development we used the **eclipse IDE (2020-12)** and **Java 11**. For the frontend we used **JavaFX 11** and for the testing we used **Junit 5**. We used ObjectAid that is bundled with the eclipse IDE to generate the package and class diagrams.

We also used **Apache Maven 4** to handle easier the dependencies of the project. Eclipse comes with a maven plugin which is very helpful.

3.4.2 Installation guide

To install the project, the developer needs to follow these steps:

1. Open eclipse.
2. Go to File -> Import.
3. Select Git -> Projects (from Git with smart import)
4. Fill the repository's information. (they can be found in this link: https://github.com/DAINTINESS-Group/PlutarchParallelLives)
5. If everything is ok the IDE must be able to identify the project.
6. Click Finish.
7. Now Right Click on the project.
8. Go to Maven -> Update Project… and click it.
9. In this dialog you <u>must check the following boxes</u>:
    a. In the "Available Maven Codebases" the project must be checked.
    b. Update dependencies.
    c. Update project configuration using from pom.xml.
    d. Refresh workspace resources from local filesystem.
    e. Clean projects
10. Click OK and you are ready.

Notes: You must have installed in your system a jdk 11+ with correct JAVA_HOME configuration. If you are using a different IDE, you may need to find how to import the project from GitHub and how to get the dependencies using Maven(some IDEs doesn't come with built-in maven plugin and you may need to install it separately).

3.4.3 User guide

In this section, we describe the user guide for the desktop application.

1. Run MainGuiApp.
2. The main view will show up.

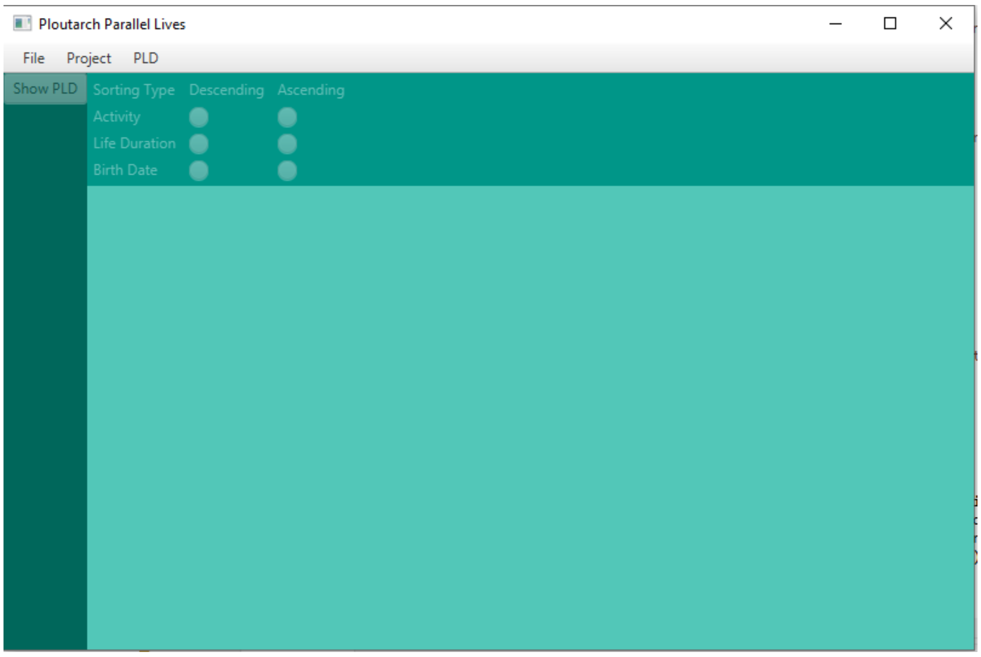
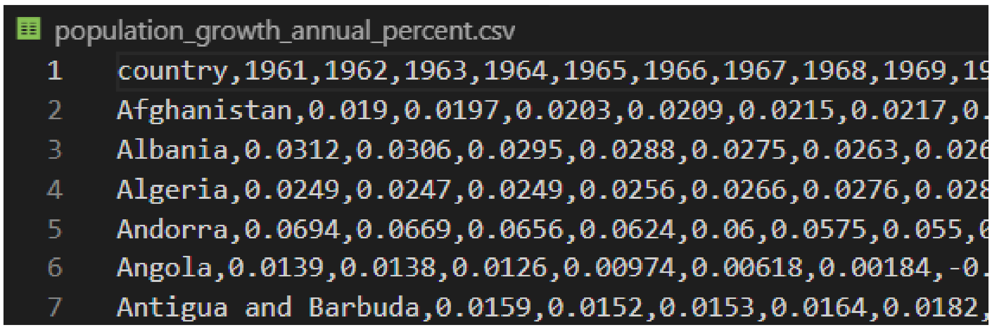


Figure 23: Main view (without data)

3. In this phase, the user must load one of the available file formats:

   i. **Single-file data** (no aggregation needed). These files can be either csv or tsv. The first row contains the time beats separated (columns) and the other rows are the entities (rows). The first column of each row contains the name of the entity and the other columns contain the measurements of this row's entity for every time beat. To load this kind of files, select **File -> Load from file.**



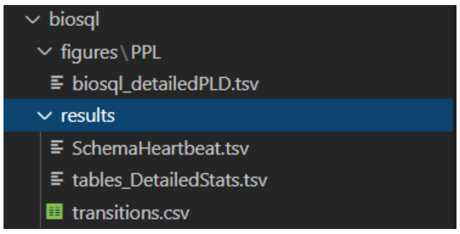
population_growth_annual_percent.csv

```
country,1961,1962,1963,1964,1965,1966,1967,1968,1969,19
Afghanistan,0.019,0.0197,0.0203,0.0209,0.0215,0.0217,0.
Albania,0.0312,0.0306,0.0295,0.0288,0.0275,0.0263,0.026
Algeria,0.0249,0.0247,0.0249,0.0256,0.0266,0.0276,0.028
Andorra,0.0694,0.0669,0.0656,0.0624,0.06,0.0575,0.055,0
Angola,0.0139,0.0138,0.0126,0.00974,0.00618,0.00184,-0.
Antigua and Barbuda,0.0159,0.0152,0.0153,0.0164,0.0182,
```

Figure 24: Simple CSV file

   ii. **Schema evolution data.** This kind of data are part of the Heraclitus tool output. Our tool requires specific folder structure because it is compatible with the Heraclitus output data. The minimum folder structure that is required is in figure 25 (without figures/PPL folders).4

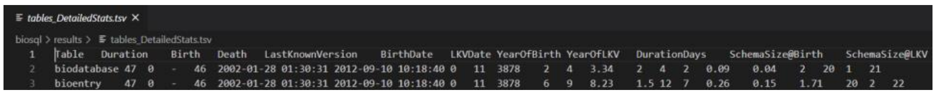
File -> Load from folder



```
biosql
  figures\PPL
    biosql_detailedPLD.tsv
  results
    SchemaHeartbeat.tsv
    tables_DetailedStats.tsv
    transitions.csv
```

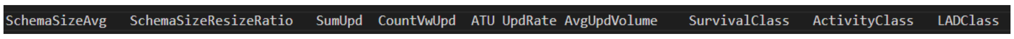
Figure 25: Schema evolution folder structure

The first file we need is **tables_DetailedStats.tsv**. It contains the required for the entity information (name, birth, death, duration).



```
tables_DetailedStats.tsv ×
biosql > results > tables_DetailedStats.tsv
Table	Duration	Birth	Death	LastKnownVersion	BirthDate	LKVDate	YearOfBirth	YearOfLKV	DurationDays	SchemaSize@Birth	SchemaSize@LKV
biodatabase 47  0   -   46  2002-01-28 01:30:31 2012-09-10 10:18:40 0   11  3878    2   4   3.34    2   4   2   0.09    0.04    2   20  1   21
bioentry    47  0   -   46  2002-01-28 01:30:31 2012-09-10 10:18:40 0   11  3878    6   9   8.23    1.5 12  7   0.26    0.15    1.71    20  2   22
```

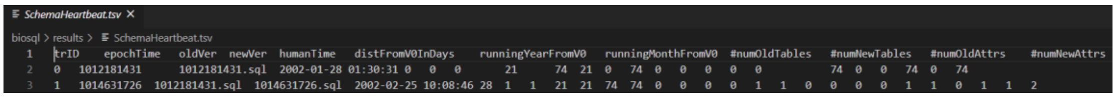
Figure 26: tables_DetailedStats.tsv part 1



```
SchemaSizeAvg   SchemaSizeResizeRatio   SumUpd  CountVwUpd  ATU UpdRate AvgUpdVolume    SurvivalClass   ActivityClass   LADClass
```

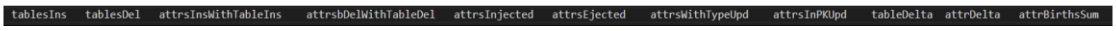
Figure 27: tables_DetailedStats.tsv part 2

The second file we need is **SchemaHeartbeat.tsv**. It contains the required time beats information (id, date, time).



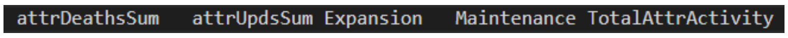
```
SchemaHeartbeat.tsv ×
biosql > results > SchemaHeartbeat.tsv
trID    epochTime   oldVer  newVer  humanTime   distFromV0InDays    runningYearFromV0   runningMonthFromV0  #numOldTables   #numNewTables   #numOldAttrs    #numNewAttrs
0   1012181431      1012181431.sql  2002-01-28 01:30:31 0   0   0       21      74  21  0   74  0   0   0   0   0           74  0   0   74  0   74
1   1014631726  1012181431.sql  1014631726.sql  2002-02-25 10:08:46 28  1   1   21  21  74  74  0   0   0   0   1   1   0   0   0   0   1   1   0   1   1   2
```

Figure 28: SchemaHeartBeat.tsv part 1



```
tablesIns   tablesDel   attrsInsWithTableIns    attrsbDelWithTableDel   attrsInjected   attrsEjected    attrsWithTypeUpd    attrsInPKUpd    tableDelta  attrDelta   attrBirthsSum
```

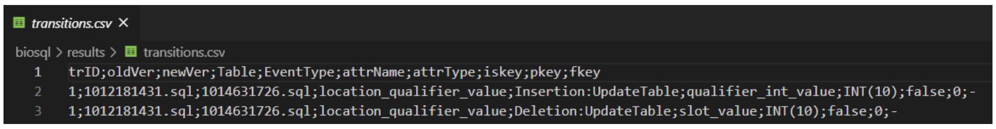
Figure 29: SchemaHeartBeat.tsv part 2



```
attrDeathsSum   attrUpdsSum Expansion   Maintenance TotalAttrActivity
```

Figure 30: SchemaHeartBeat.tsv part 3

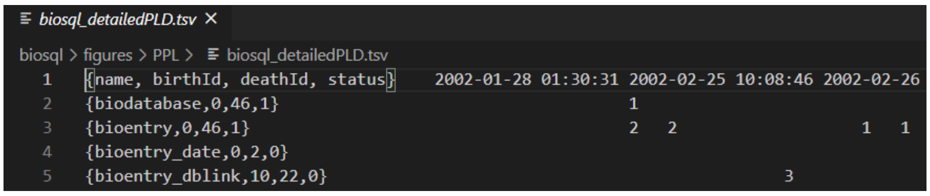
Finally, the third file we need is **transitions.csv**. It contains all the transaction from every time beat to the next one for each entity.



```
transitions.csv ×
biosql > results > transitions.csv
trID;oldVer;newVer;Table;EventType;attrName;attrType;iskey;pkey;fkey
1;1012181431.sql;1014631726.sql;location_qualifier_value;Insertion:UpdateTable;qualifier_int_value;INT(10);false;0;-
1;1012181431.sql;1014631726.sql;location_qualifier_value;Deletion:UpdateTable;slot_value;INT(10);false;0;-
```

Figure 31: transitions.csv

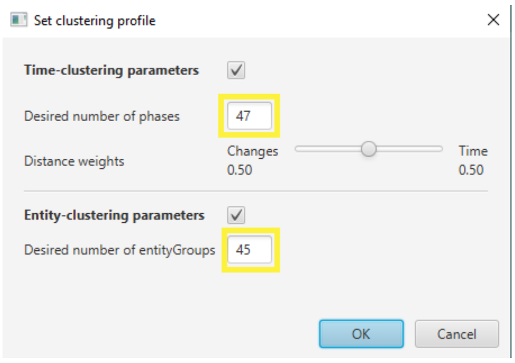
iii. **Converted tsv data** (aggregated data generated by this tool). This format is created by us. We call it intermediate representation or tem (TimeEntituGroups) file. The first line contains the time beats in human date time format. The other line contains entities, the first column of each entity column contains the needed info (name, birth, death, status) and the other columns represent the measurement of an entity in a specific time moment.



```
biosql_detailedPLD.tsv ×
biosql > figures > PPL > biosql_detailedPLD.tsv
{name, birthId, deathId, status}    2002-01-28 01:30:31 2002-02-25 10:08:46 2002-02-26
{biodatabase,0,46,1}                                    1
{bioentry,0,46,1}                                       2   2                    1   1
{bioentry_date,0,2,0}
{bioentry_dblink,10,22,0}                                                3
```

Figure 32: Intermediate representation file

4. When we have chosen the desired dataset, the clustering dialog will load.



Set clustering profile

Time-clustering parameters ✓

Desired number of phases: 47

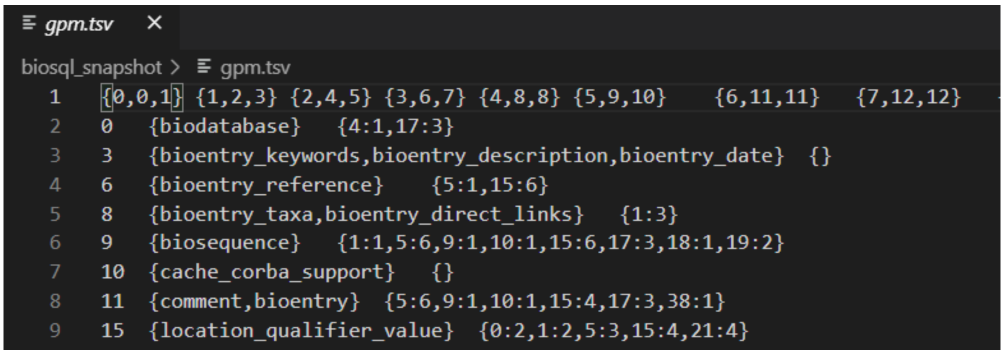
Distance weights: Changes 0.50 — Time 0.50

Entity-clustering parameters ✓

Desired number of entityGroups: 45

Figure 33: Clustering dialog

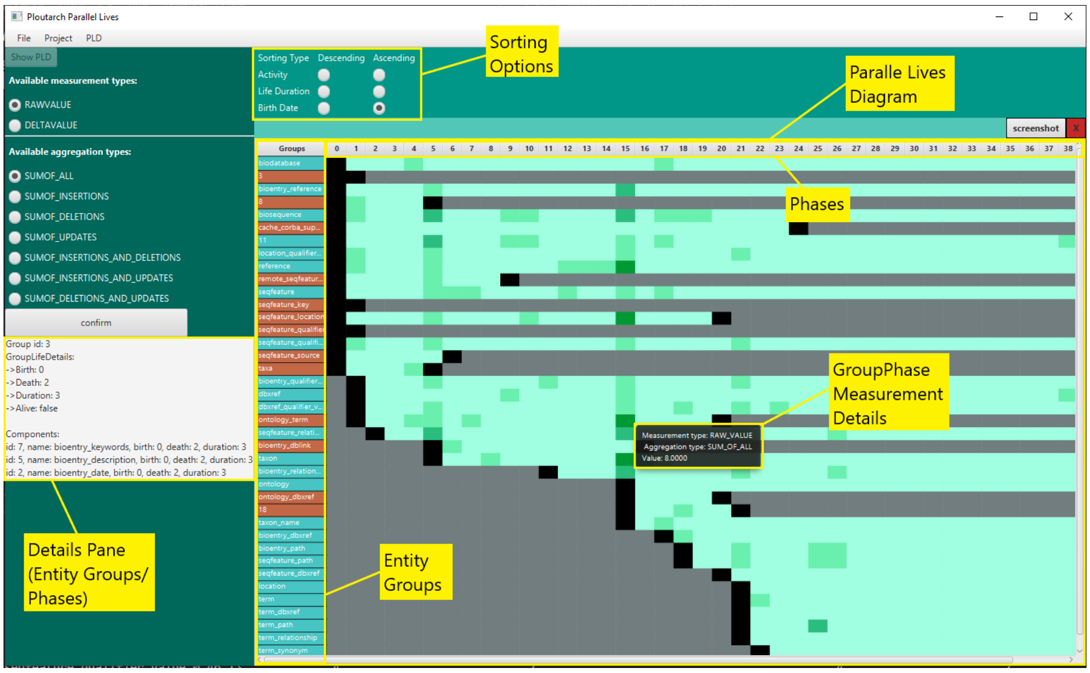
5. In this dialog, we can decide if we are going to **summarize our data** (clustering) to make the fit in our screen. We can set the **desired number of Phases** (columns) and **EntityGroups** (rows) and the phase clustering weights. If we leave the desired numbers unchanged there will be no clustering because the default values are the actual numbers of rows and columns of the dataset.
6. We can **also import a dataset snapshot** which consists of a two-file folder **(tem.tsv: intermediate representation & gpm.tsv summarized representation)**. This way we can skip the clustering process. The gpm file has a special format to keep the phase and entity group information. The first row has tab-separated the phases information **{phase id, first time beat, last time beat }**. The other rows contain the entity groups information **entity group id \tab {list of component entity names} \tab {list of phase id : measurement pairs}**. (Look figure 34)



```
{0,0,1} {1,2,3} {2,4,5} {3,6,7} {4,8,8} {5,9,10}    {6,11,11}   {7,12,12}
0   {biodatabase}   {4:1,17:3}
3   {bioentry_keywords,bioentry_description,bioentry_date}  {}
6   {bioentry_reference}    {5:1,15:6}
8   {bioentry_taxa,bioentry_direct_links}   {1:3}
9   {biosequence}   {1:1,5:6,9:1,10:1,15:6,17:3,18:1,19:2}
10  {cache_corba_support}   {}
11  {comment,bioentry}  {5:6,9:1,10:1,15:4,17:3,38:1}
15  {location_qualifier_value}  {0:2,1:2,5:3,15:4,21:4}
```

Figure 34: GroupPhaseMeasurement file (gpm.tsv)

7. Now we have successfully loaded a dataset in our system and we can do the following things:
   i. We can sort the rows with one of the available sorting types.
   ii. We can show the display the data in the PLD.
   iii. We can export our data as a snapshot (intermediate representation & summarized representation).
8. We can click the “Show PLD” button to show the Parallel Lives Diagram. When the diagram is displayed, we can do a variety of things.
   i. We can export it as a PNG.
   ii. We can sort the data.
   iii. We can display other generated types of our data.
   iv. We can left click on an entity group to get its details.
   v. We can right click on a phase to get its details.
   vi. We can hover on a measurement cell to get its details.
   vii. We can zoom in and out by scrolling the mouse wheel.
9. Parallel Lives Diagram User Guide:
   i. Cell colors: Gray :: inactive entities, Black :: life related beat (life or death), Shades of Green :: the lightest one maps to an active entity without measurement that moment, the other three darker shades map to three different value buckets that are generated according to the min and the max measurement of the dataset.
   ii. EntityGroup colors: Red :: dead EntityGroup, Blue :: alive EntityGroup
   iii. As we see in the EntityGroups column (in figure 35), some cells contain string and other integers. The string (EntityName) is used when the line maps to only one Entity. The integer (EntityGroupID) is used when the line is an EntityGroup that contains more than on Entities.


Ploutarch Parallel Lives
File Project PLD
Show PLD
Available measurement types:
RAWVALUE
DELTAVALUE
Available aggregation types:
SUMOF_ALL
SUMOF_INSERTIONS
SUMOF_DELETIONS
SUMOF_UPDATES
SUMOF_INSERTIONS_AND_DELETIONS
SUMOF_INSERTIONS_AND_UPDATES
SUMOF_DELETIONS_AND_UPDATES
confirm
Sorting Type Descending Ascending
Activity
Life Duration
Birth Date
Sorting Options
Paralle Lives Diagram
screenshot
Groups
Phases
GroupPhase Measurement Details
Entity Groups
Details Pane (Entity Groups/ Phases)

```
Group id: 3
GroupLifeDetails:
->Birth: 0
->Death: 2
->Duration: 3
->Alive: false

Components:
id: 7, name: bioentry_keywords, birth: 0, death: 2, duration: 3
id: 5, name: bioentry_description, birth: 0, death: 2, duration: 3
id: 2, name: bioentry_date, birth: 0, death: 2, duration: 3
```

```
Measurement type: RAW_VALUE
Aggregation type: SUM_OF_ALL
Value: 8.0000
```

Figure 35: Sorted PLD and details

3.5 Software expandability

Our system can be employed in other applications, it can be extended, and it can be modified to the user needs. We will list some tips for some of the possible changes that someone may want to make.

3.5.1 Add new file format support

To make the system compatible with a new dataset format a developer must follow the next steps:

1. Create a proper ILoader implementation and add it as option to the LoaderFactory.
2. Add the new file type in the FileType enums.
3. Modify properly the parseFileInfo method in the FileHandler.

3.5.2 Change the clustering methods

The following process is similar for both PhaseExtractor (column clustering) and EntityGroupExtractor (row clustering).

1. Create a proper IPhaseExtractor implementation and add it as option to the PhaseExtractorFactory.

2. Either add a parameter in the generatePhases method of the MainController to be able to choose between the different methods or just replace the current method with the desired one in the generatePhases method.

Chapter 4. Experimental validation

In this chapter, we describe the experimentation methodology that we followed to calculate the performance of our system and we discuss the results.

4.1 Experimentation methodology

The experiment we tried to test the performance of our system is the following. We want to check whether the conversion process of a complex parallelly evolving time-series format to the intermediate language is performing well.

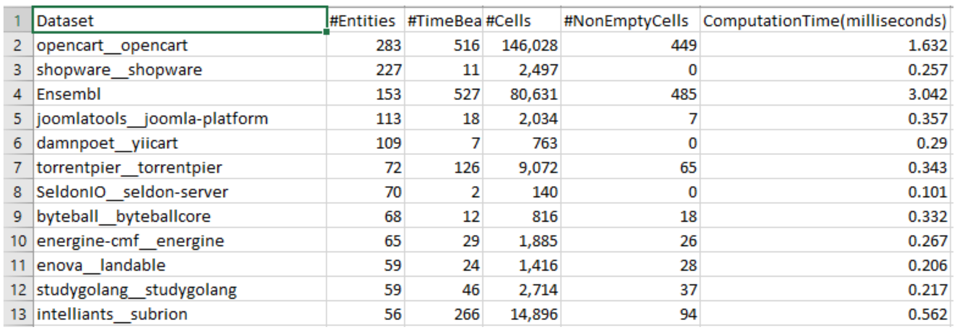
The most complex format we have in our hands is the relational schema evolution format, so we timed our system using 196 different datasets of different sizes. To time the datasets, we created a script that applies the following steps for each dataset:

1. Load the dataset.
2. Convert the data to the intermediate representation.
3. Write the data to new file using the intermediate representation.
4. Time the computation time.
5. Repeat the steps 1-4 ten times.
6. Save the mean computation time.

We also stored the following features for each dataset:

- Number of Entities
- Number of Beats
- Number of Cells (Entities x Beats)
- Number of Non-Empty Cells

We used the above data to create a tsv file that can be subsequently used to analyze the performance of the system.



1	Dataset	#Entities	#TimeBea	#Cells	#NonEmptyCells	ComputationTime(milliseconds)
2	opencart__opencart	283	516	146,028	449	1.632
3	shopware__shopware	227	11	2,497	0	0.257
4	Ensembl	153	527	80,631	485	3.042
5	joomlatools__joomla-platform	113	18	2,034	7	0.357
6	damnpoet__yiicart	109	7	763	0	0.29
7	torrentpier__torrentpier	72	126	9,072	65	0.343
8	SeldonIO__seldon-server	70	2	140	0	0.101
9	byteball__byteballcore	68	12	816	18	0.332
10	energine-cmf__energine	65	29	1,885	26	0.267
11	enova__landable	59	24	1,416	28	0.206
12	studygolang__studygolang	59	46	2,714	37	0.217
13	intelliants__subrion	56	266	14,896	94	0.562

Figure 36: Sample of the experiments results

For the experiments we used a Dell Inspiron 7559 laptop with the following specifications:

- CPU: intel i7-6700HQ 2.60 GHz
- RAM: 8GB
- 1TB HDD

4.2 Detailed results

We use scatter plots with different features as x-axis and the time needed to load the input, convert it in an intermediate representation and store this intermediate representation to a file, which we call *computation time* trying to find a correlation between them. Unfortunately, most of our datasets were small. About 180/196 datasets have less than 1000 cells. Nevertheless, we have created several scatter plots using all the available datasets:

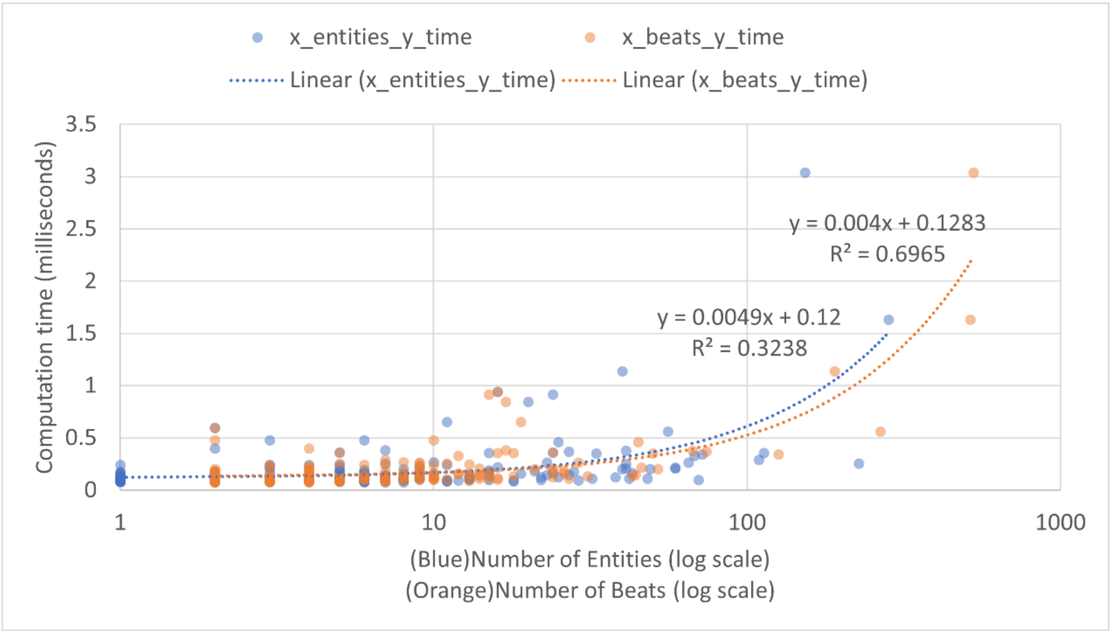


Figure 37: Scatter plot (entities/beats, computation time)

In Figure 37, we depict the computation time of the 196 tables as a function of the number of the peer entities appearing in each input dataset for the blue points, and, as a function of the number of time-beats appearing in each input dataset for the orange points.

The performance of the system seems to be very weakly connected to the number of both entities and time-beats for small volumes of them and seems to be affected only above the threshold of 100 entities/time-beats. However, the performance seems to be slightly more affected by the number of entities.

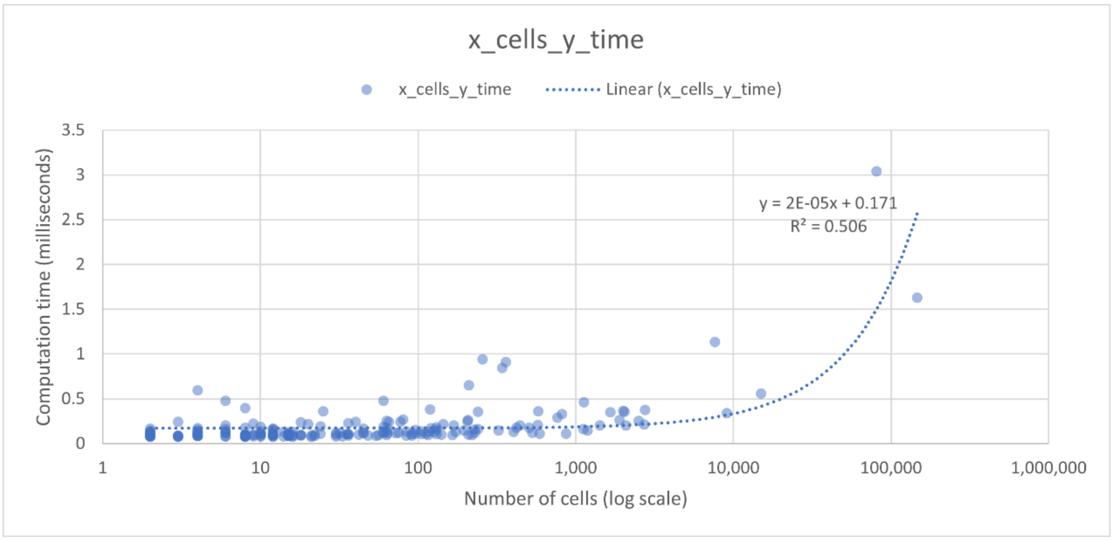


Figure 38: Scatter plot (cells, computation time)

In Figure 38, we depict the computation time of the 196 tables as a function of the number of the cells appearing in each input dataset.

The performance of the system seems to be very weakly connected to the number of cells for small volumes of them and seems to be affected only above the threshold of 10000 cells. This makes sense, because the number of cells equals the number of entities multiplied by the number of time-beats.

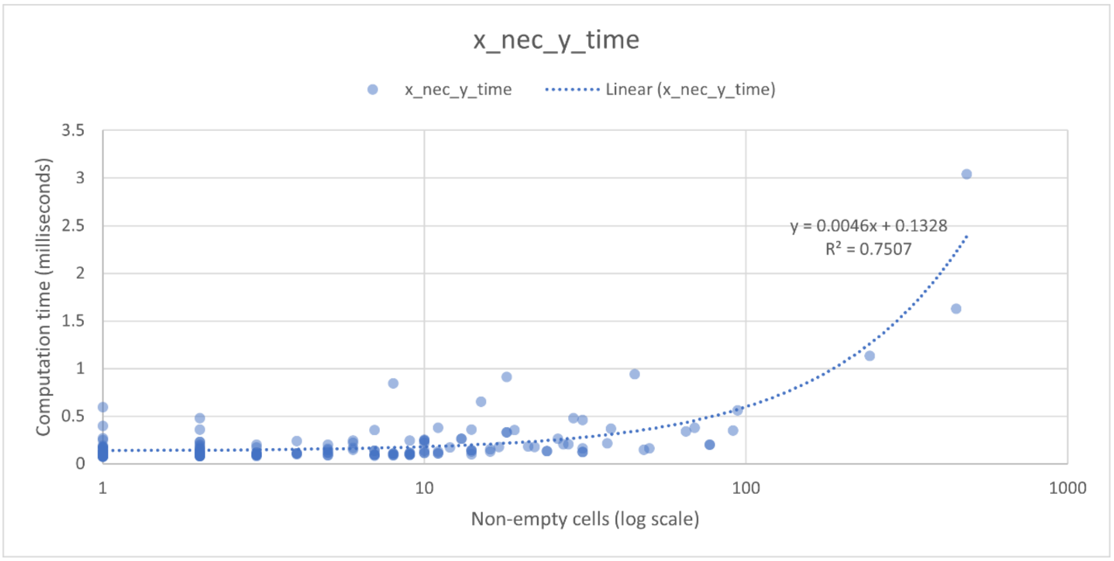


Figure 39: Scatter plot (non-empty cells, computation time)

In Figure 39, we depict the computation time of the 196 tables as a function of the number of the non-empty cells appearing in each input dataset.

The performance of the system again seems to be very weakly connected to the number of non-empty cells for small volumes of them and seems to be affected only above the threshold of 100 non-empty cells.

These scatter plots do not tell us much. The small values in x-axis are so small that their insignificant in our experiments and we could assume that these tiny computation times are more related with the system load (the moment of the timing) than with the datasets.
That is why we tried to do the same scatter plots with only the twenty highest values of each feature.

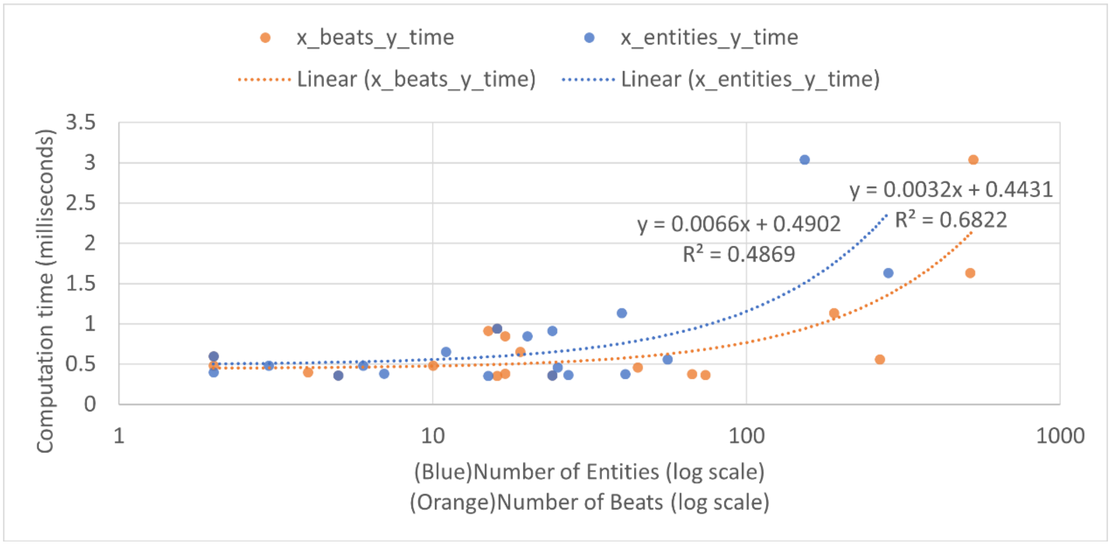


Figure 40: Scatter plot top-20 (entities/beats, computation time)

In Figure 40, we depict again the computation time of the 196 tables as a function of the number of the peer entities appearing in each input dataset for the blue points, and, as a function of the number of time-beats appearing in each input dataset for the orange points.
In this scatter plot it is clearer than before that the number of entities affect the system's performance more than the number of time-beats. However, we do not have enough big datasets to be sure.

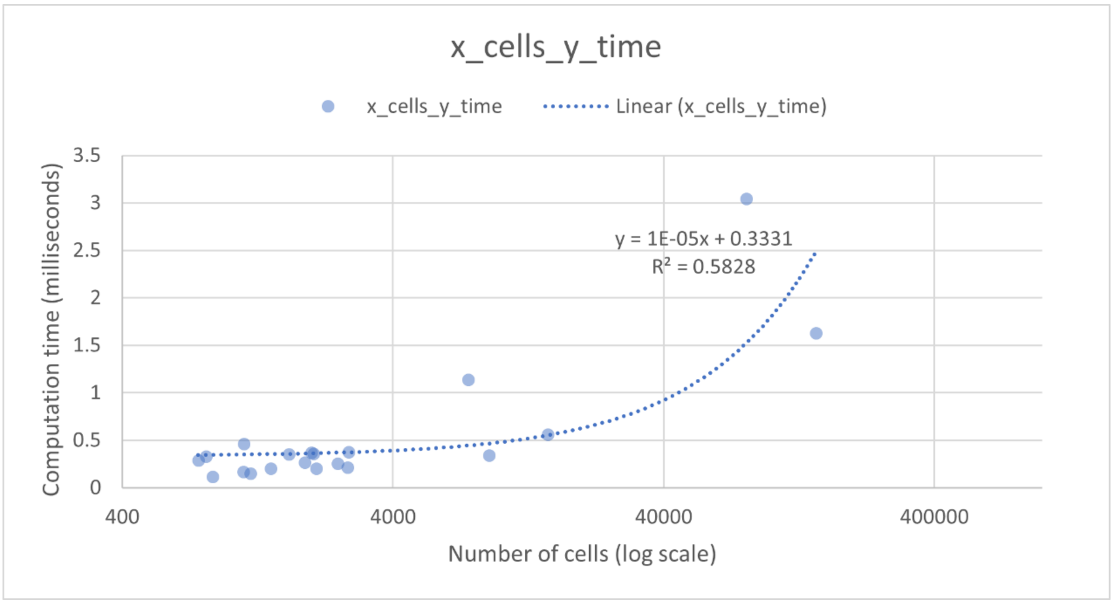


Figure 41: Scatter plot top-20 (cells, computation time)

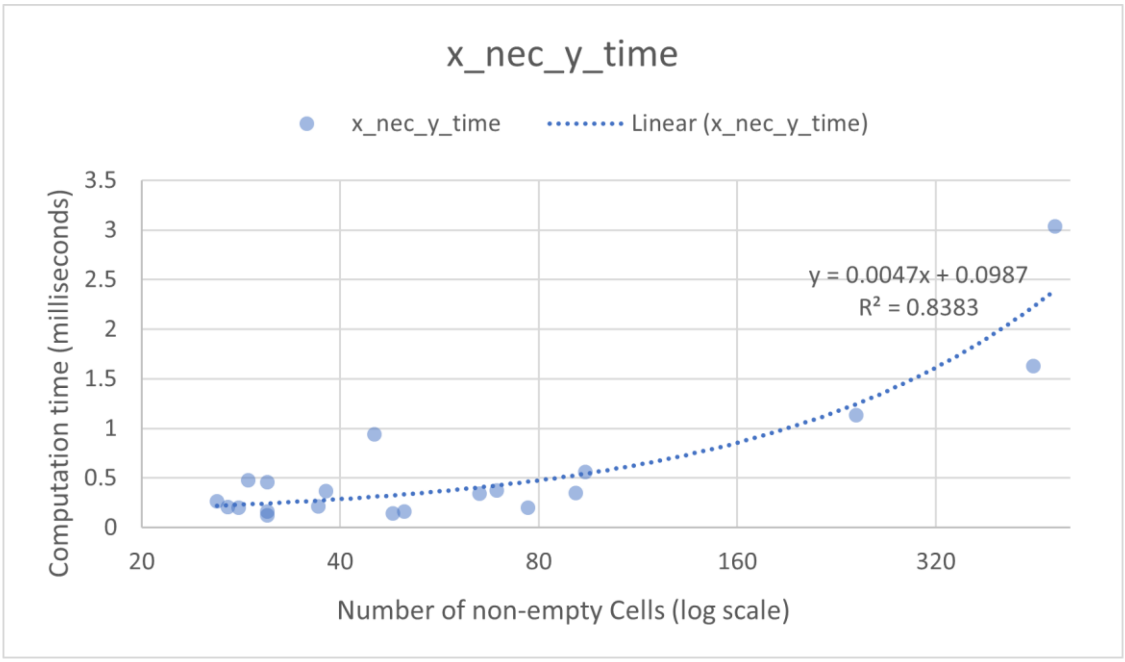


Figure 42: Scatter plot top-20 (non-empty cells, computation time)

Finally, in the Figures 41 and 42, we do not get any new information about our data, they are very similar with the Figures 38 and 39.

Chapter 5. Conclusion

In this chapter, we conclude the contribution and the results of this Thesis and list some ideas about the system's future improvements and expansions.

5.1 Summary and conclusions

The goal of this diploma was the creation of a strong parallel time-series visualization tool. Giachos has already created such a tool in his MSc study [Giac15] but the resulting tool came up with some format restrictions. It supports only the relation schema evolution format. Therefore, we recreated it from scratch in a more generalized version that supports more parallelly evolving time-series formats.

To make this possible we had to create a new format for our system, the intermediate representation. It was designed in a way that it would be easy to convert other parallelly evolving time-series formats to it.

We developed our system in a way that it would be easy to attach a custom loader/converter to it to extend its compatibility. Our first version of this system is compatible with the commonly used GapMinder format and the relational schema evolution format.

Both formats can be visualized using our Parallel Lives Diagram implementation. Our PLD comes with a set of features (zooming, sorting, save as image etc.) to make easier the parallel time-series analysis.

Sometimes these datasets can be big in size and they cannot be fully displayed in a screen. As Giachos did in [Giac15], we also used agglomerative clustering to create summaries of the datasets. This process helps us reduce the data to a desired size that now is easier to be studied. Our system also allows to import/export from/to file the clustering data to skip the step of the clustering in case it is already done.

We tested the correct functionality of our system and finally, we tried some experiments to validate the performance of our system on loading schema evolution data, converting them to intermediate representation and storing the intermediate representation to a file. Unfortunately, we did not have enough big datasets to be sure of the results, but according to the available datasets we found that the system's performance is slightly more affected by the number of entities.

5.2 Future work

There are several opportunities to extend and improve our system.

The first thing that should be changed in the future is the JavaFX TableView that we use to implement the Parallel Lives Diagram. We noticed that our system breaks when we try to display datasets that have more than 150 columns. This problem is caused by the implementation of the TableView, we found out that there is an issue [https://github.com/javafxports/openjdk-jfx/issues/409] in the JavaFX repository in GitHub which basically says that TableView has restriction in the column's representation. That is why we would suggest replacing this component with a custom one or with a JTable (JavaFX supports Swing components).

In the future we could make our GUI more beautiful and add more features to it like adding a second diagram to display the data without clustering, enrich the details Pane etc.

A complete CLI tool that converts the data to intermediate representation and generates summaries of them would be useful.

These are the most important parts that can be improved in the future.

Our GitHub Repository: https://github.com/DAINTINESS-Group/PlutarchParallelLives

Reference

[Giac15] T. Giachos. Biography Synopses for Evolving Relational Database Schemata. MSc Thesis, Dept. of Comp. Sc. and Eng., 2015, Available at https://www.cs.uoi.gr/wp-content/uploads/publications/MT-2015-18.pdf

[KrBK11] Milos Krstajic, Enrico Bertini, Daniel A. Keim, CloudLines: Compact Display of Event Episodes in Multiple Time-Series, IEEE Transactions in Visualization and Computer Graphics, 17(12), pp. 2432-2439, December 2011

[WaBJ16] James Walker, Rita Borgo, Mark W. Jones, TimeNotes: A Study on Effective Chart Visualization and Interaction Techniques for Time-Series Data, IEEE Transactions in Visualization and Computer Graphics, 22(1), pp. 549-558, January 2016

[KuSt12] A. Kuhn and M. Stocker, "CodeTimeline: Storytelling with versioning data," *2012 34th International Conference on Software Engineering (ICSE)*, Zurich, 2012, pp. 1333-1336, doi: 10.1109/ICSE.2012.6227086