SUSTAINABLE
BUILDINGS

Architecture Document

Client: Sustainable Buildings

Version 1.0

A visualizing tool for OrientDB

Authors:
Niels Bugel
Albert Dijkstra
Antal Huisman
Carlos Isasa
Yona Moreda
Emanuel Nae

TA:
Ai Deng

Contents

1 Introduction

Sustainable Buildings is a company that develops the next generation of energy management systems. The goal of this is to reduce energy consumption in office buildings, ease the job of building managers, and make the working environment of building users healthier and more productive. The users can view and manage the energy consumption of multiple buildings from one dashboard. In order to do this, Sustainable Buildings is using semantic data, mainly due to their primary needs, such as distinguishing different types of data, storing user-provided data and establishing relationships between the varying entities. The semantic data is being stored by using OrientDB, because it allows the data to be represented as a graph-oriented model. Our aim is to provide a tool that helps the user to visualize the information stored from the database as a tree-structured graph, this making the data much more readable and understandable.

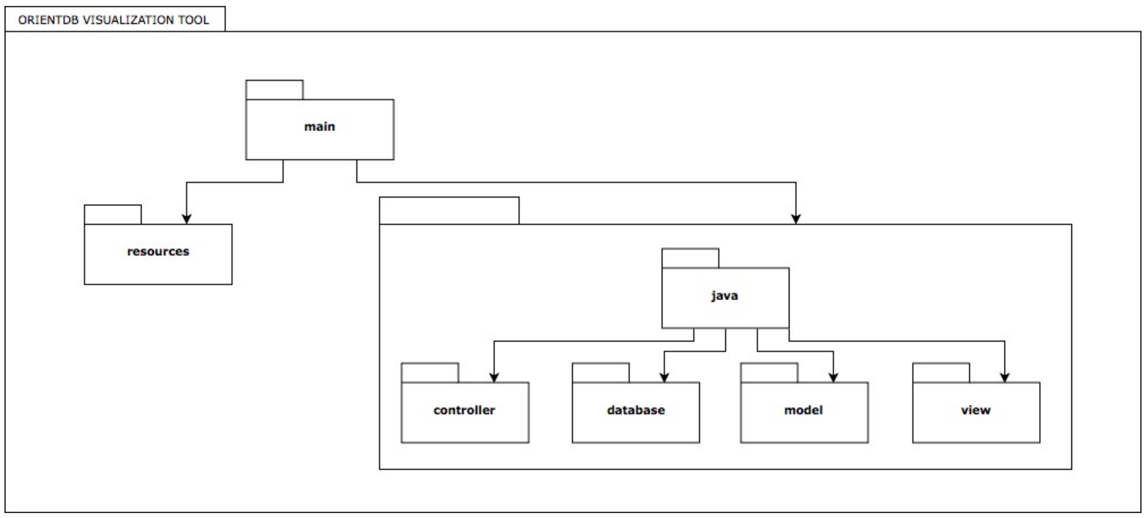
2 Architectural overview

For this project we will be working with the model, view, controller pattern. This means that, generally, our application will be split up into four parts.

- The model is responsible for the internals of the application. This means that the model handles all the calculations. It also handles the data between the database and the view/controller.
- The view will get information from the model and display this on the screen. This means that almost no calculations will be done here.
- The controller is responsible for handling the interaction between the view and the model. For example, the view may contain some buttons. These buttons will have a certain event when clicked. Such an event might be that something is being changed in the model.

By using this pattern, there is a clear distinction between front-end and back-end. It also provides us with a structured way to build our program.
On top of that we also have a folder called database. This folder contains all the classes that are relevant for retrieving data directly from the database.

2.1 Component diagram



2.2 Back-end

The back-end will consist of two parts:

- First we have the classes that contain all the relevant data from the database. We will call this the model.
- Second, since we are working with orientDB, we somehow need to convert that data into these useable classes for java. This is the part that we call the translator.

2.2.1 Model

The model is subdivided into multiple different parts.

- **Place**
  This category holds all the data for the following places: locations, buildings, floors, rooms, areas and cells. Each place contains a list with their subsequent children. A place also contains a list of connected entities.
- **Organization**
  This holds all the information about an organization. Each organization contains a list with their corresponding locations

- **Entity**
  An entity is a sensor that is described by multiple traits.

2.2.2 Translator

The translation between the database to the java application is done by a group of translator classes. Those classes have two functions. First these classes query the database. Then they put the data inside the java classes as described above. This makes sure that the front-end does not need to work with the database directly, but can work with the classes that are defined to store that data.

2.2.3 Database

databaseManager
This class is responsible for the database connection. First it connects to the orientDB database. Then it initialises all the classes that are responsible for retrieving specific data from the actual database. **databaseData** There are a few classes that are responsible for loading the actual data from the database. These classes are:

- OrganizationData: First loads all the organizations, then create for each of these organizations an Organization class. Next it loads for each of the organizations the connected locations and adds them to the organization.
- PlaceData: Loads all the children of a specific locations. Performs a sort of depth-first-search to explore all the children of a location.
- EntityData: Loads all the entities connected to a specific place.

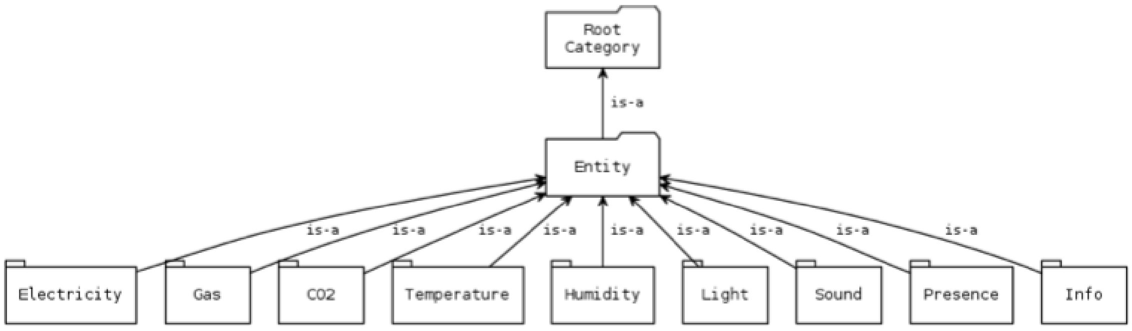
These classes load data from the database and put this data in the respective java classes. That way all the classes representing the data in the database are initialized.

Attributes
The problem with the database is that not all vertices always have the same attributes. Because of this we created the attribute classes. These take care of retrieving all available attributes from the database and leaving out the ones that are not in the database.

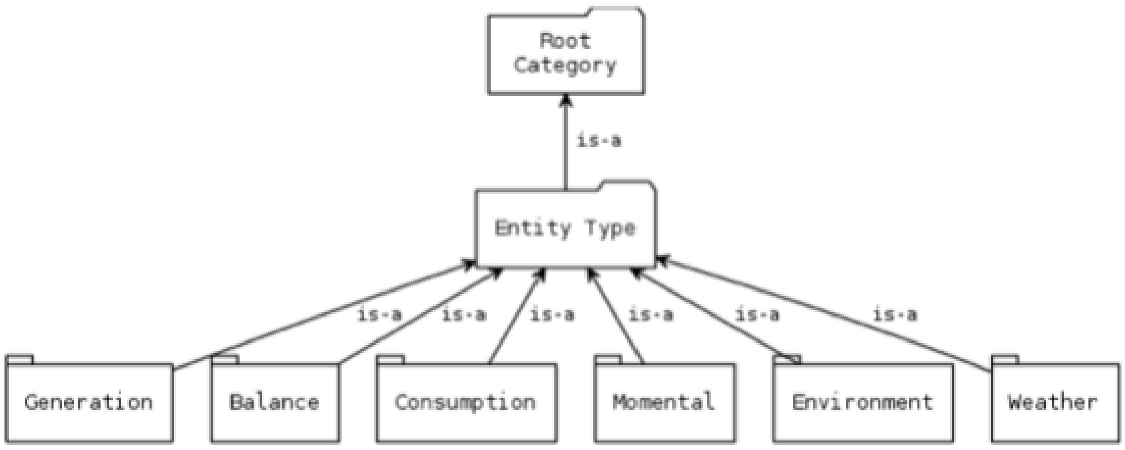
2.2.4 OrientDB

The OrientDB database provided by Sustainable Buildings can be divided into four architectural levels:

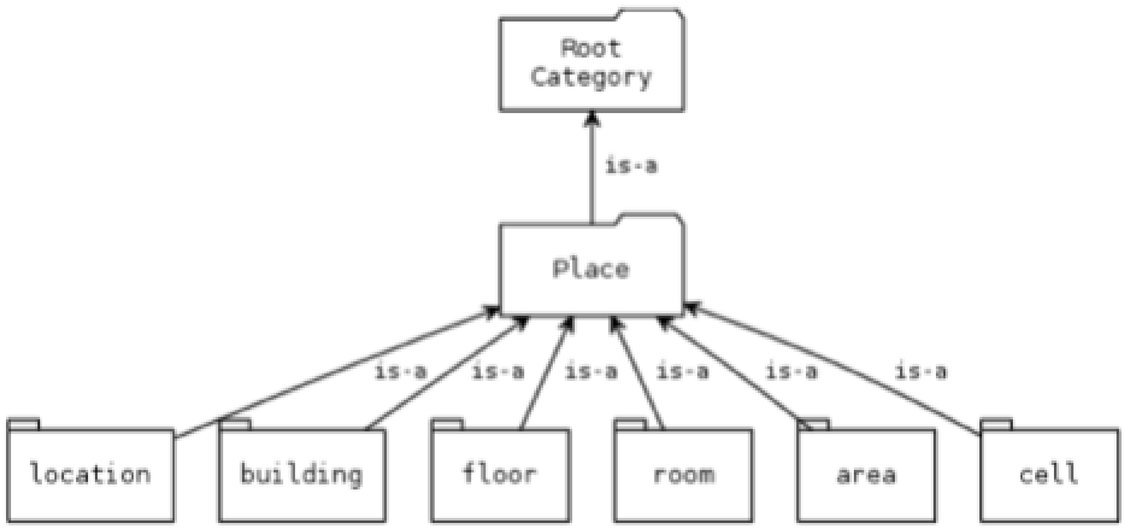
- Entity level



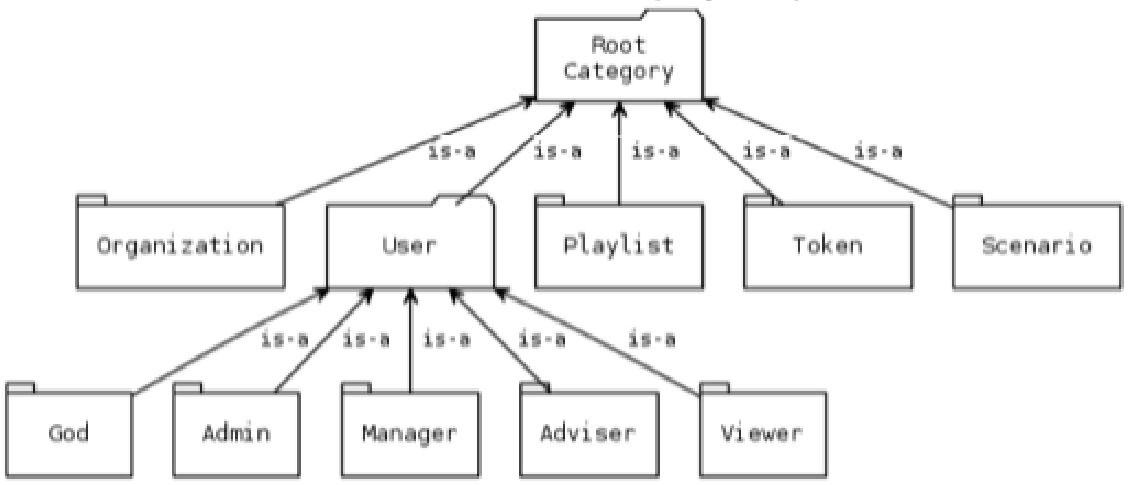
- Entity-type level



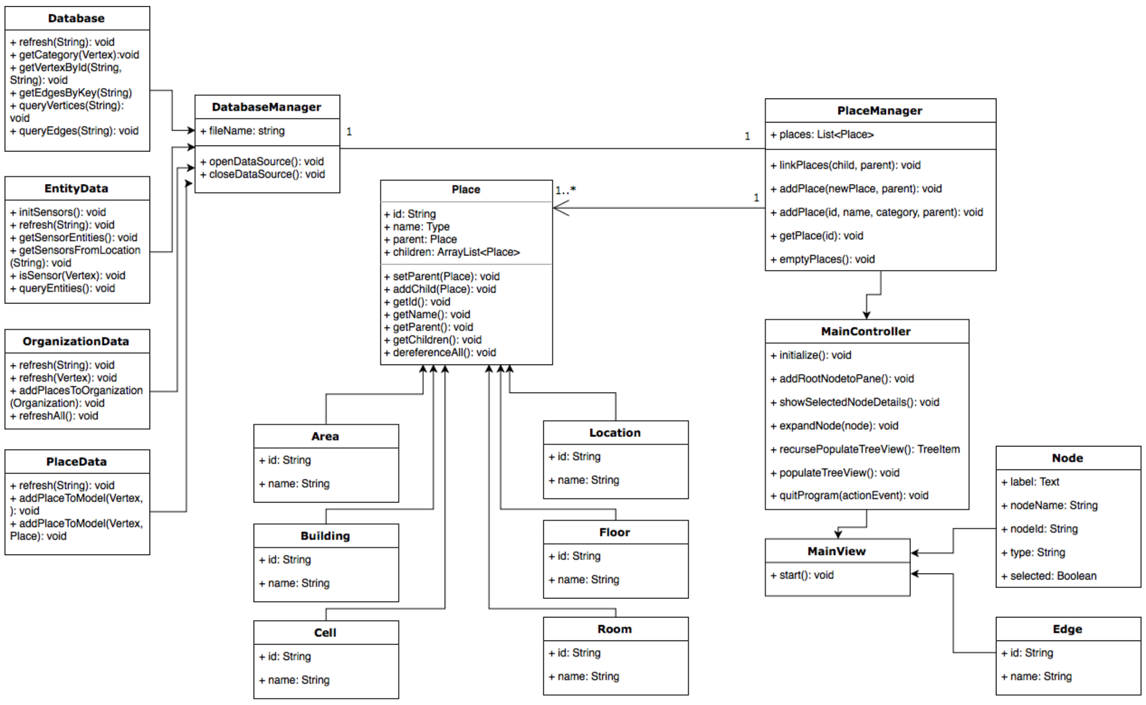
- Location level



- Organization level



2.2.5 Class Diagram



2.3 Front-end

The front-end uses the JavaFX software platform to build the graphical interface for the database application. In addition, to facilitate the creation and organization of GUI components, a JavaFX development tool called Gluon Scene Builder is put to use. The Gluon Scene Builder allows a quick design of JavaFX graphical user interfaces without the need for writing code. After Gluon Scene Builder was used to design and organize the basic components in the user interface, the corresponding design is complemented with a java front-end code that will enable full functionality of the JavaFX GUI components present in the design.

The view component is first split into two categories. One category includes components that are mainly used for building the background. These components are the menu bar, status bar, the left, center and right panels. The second category includes view components that considered to be dynamic and their appearance solely depends on the data from the database. These components are defined with java classes and have java code. They are the rectangular nodes, the lines that represent edges and finally some intermediate GUI layers such as the VBox which are used to manage these elements.

The layers described above are illustrated in Figure 3.

The first category elements (the background/static[1] GUI elements) are described through a markup language called FXML. Thus, a file with an FXML file formate is created and is used to define the rules in which the background GUI elements are organized using tags. The Gluon Scene Builder uses this FXML file to create and manage these background layers and achieve a design that is polished and user-friendly. Then these design components are linked to a Java Controller class which defines the actions must be performed for different action events.

2.3.1 Graphing the Tree

In order to graph the tree, a layout was designed to pack certain elements together and organize them together in tree structure. This layout design was primarily created to tackle the issue of node occlusion and to allow the management of nodes while retaining a tree-like configuration.

First, the tree elements are broken up into JavaFX GUI components such as the JavaFX `Rectangle`, `Label`, `Stackpane`, `VBox`, `Pane` and `Line` (`Edge`). A node is represented by a `Stackpane` that packs a JavaFX label and a JavaFX rectangle together as a stack where the label is placed on top of the rectangle.

Further up the `Stackpane`, there is a Pane that is used to pack sub-trees as component. This `Pane` packs the node (`Stackpane`), the children nodes (`VBox`) and the edges (`Line`), together as unit. This container pane allows for an abstraction and to treat the graph as collection of sub-trees organized in vertical boxes (`VBox`).

As shown in Figure 1, it can be seen that the nested organization of these elements is analogous to the structure of a tree, and this organization is what enables the program to recognize each sub-tree as a constituent and organize the data into a tree structure.

[1]Static here refers to the fact that these background elements do not change with the given data and can be seen as the GUI layer that is a skeleton for the application.

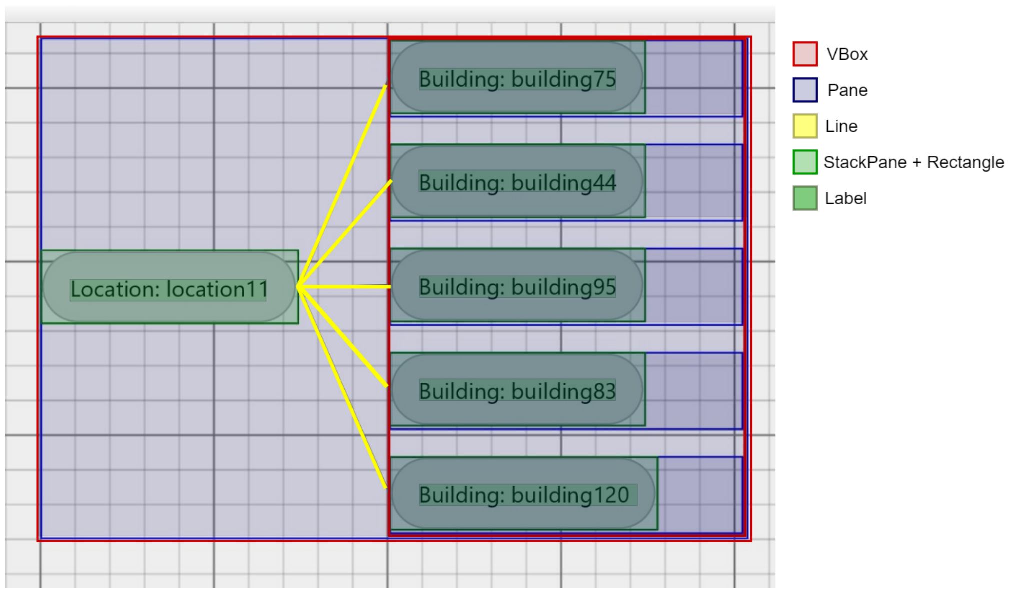


Figure 1: A graphing layout for nodes that accounts for node occlusion

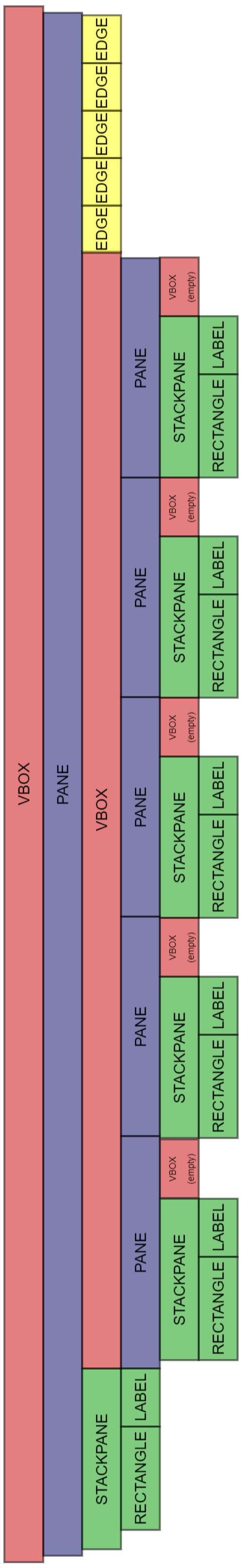


Figure 2: Hierarchy of GUI components for the tree example shown in Figure 1

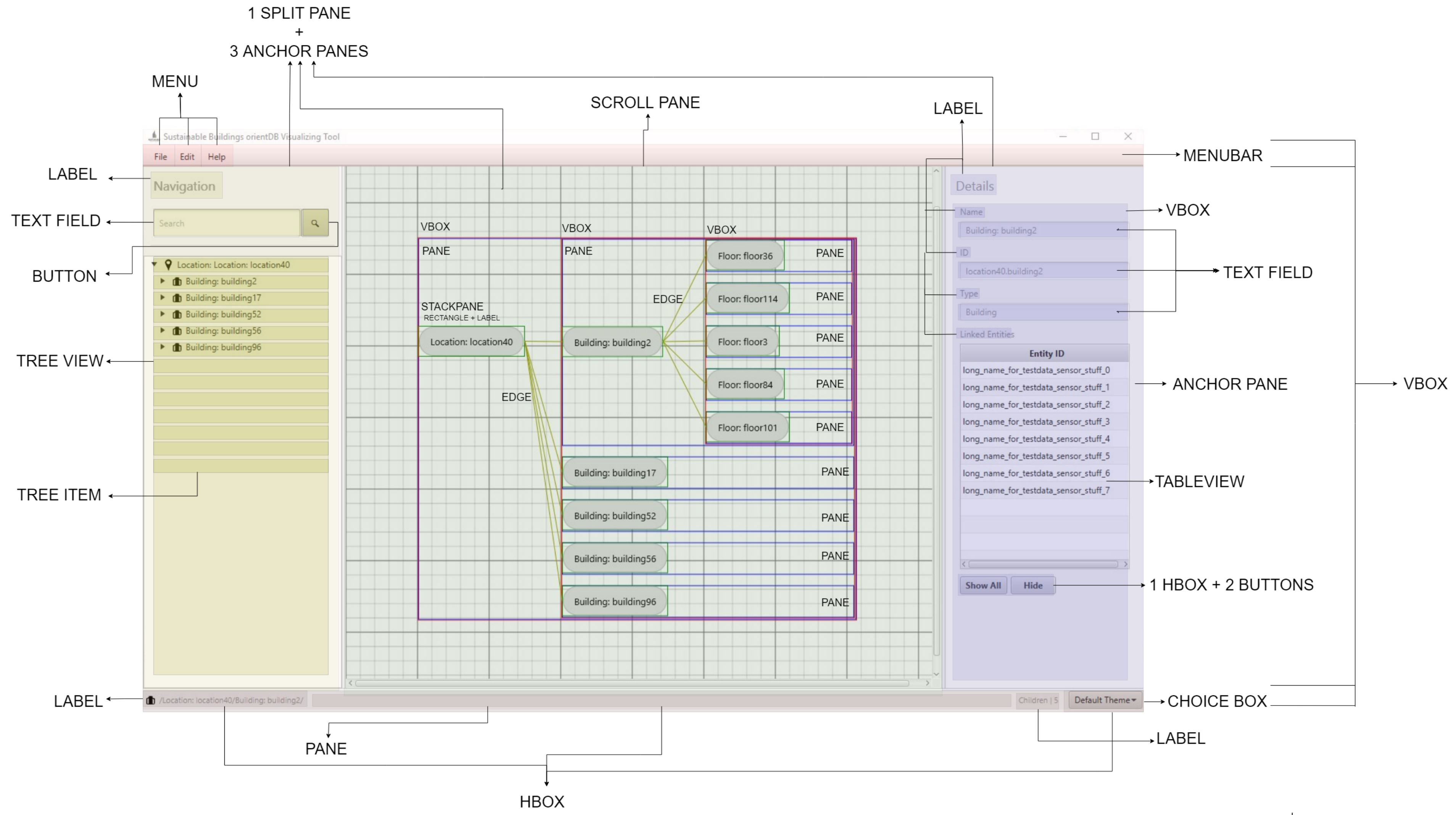


Figure 3: The underlying JavaFX layers of the visualization tool

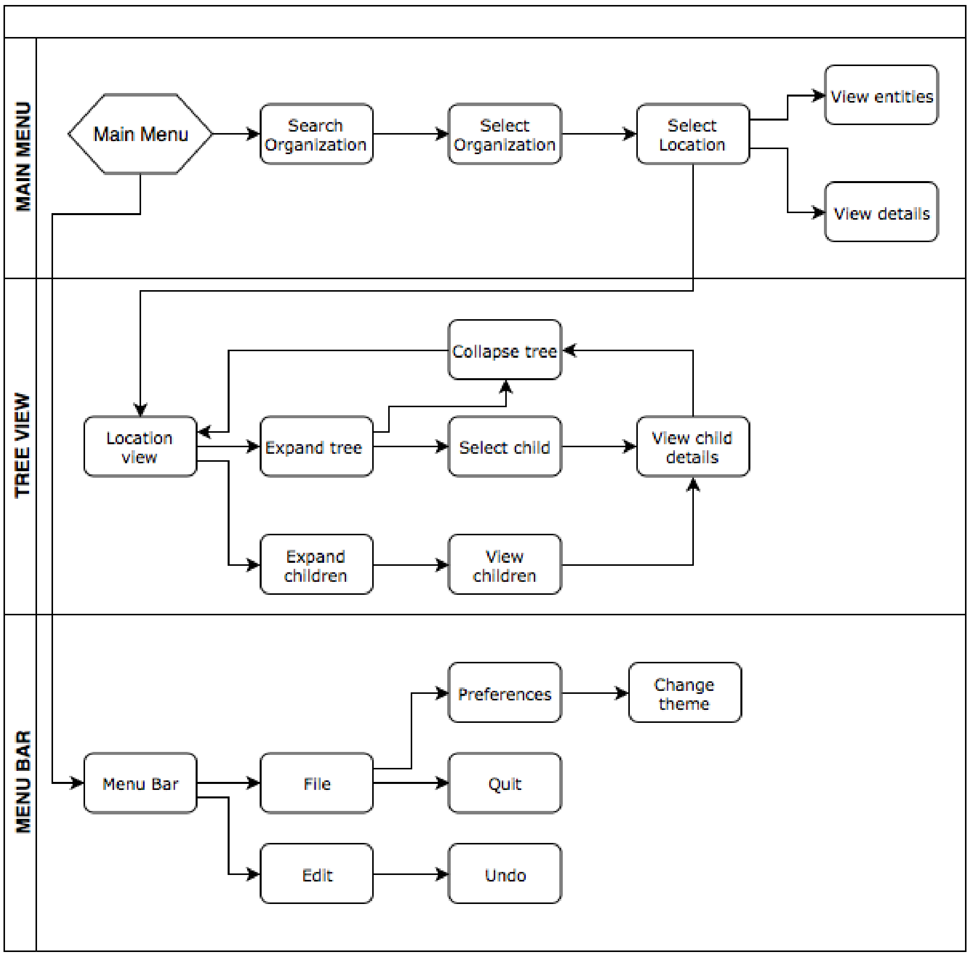


Figure 4: User-flow Diagram

2.3.2 User-flow Diagram

In the above diagram, it is shown what actions the user can perform inside the application, starting from the main menu.

3 Technology stack

3.1 Languages

- Java

3.2 Libraries

- JavaFX
- OrientDB

3.3 Building Tools

- NetBeans
- Eclipse
- Intellij IDEA
- Gluon Scene Builder

3.4 Version Control

- Github

4 Team Organisation

The team is divided in three teams.

- The front-end team. This team is responsible for the user interface. This means that they develop the part of the application that shows the data on the screen and handles the interaction with the user, e.g. buttons and menus. Since we are using the MOV pattern, this team will be responsible for both the view and the controller.
- The back-end team. This team is responsible for the part of the application that handles the data and actions between the OrientDB database and the user interface. Since we are using the MOV pattern, this team will be responsible for the model.
- The all-round team makes sure that everything is well structured and consistent. They will mostly do back-end work, however, if help is needed for the front-end then they will help there as well.

Team composition:

Team	Who	Responsibility
Front-end	Emanuel Nae Yona Moreda	View/controller, with emphasis on the view
Back-end	Carlos Isasa Antal Huisman	Model
All-round	Albert Dijkstra Niels Bugel	Model, view and controller, with emphasis on the controller and model

5 Change Log

Date	Who	Section	What
5 March 2019	Albert Dijkstra	Document	Created document and layout
5 March 2019	Niels Bugel	Document	rewrote Team organisation, wrote Architectural overview.
7 March 2019	Carlos Isasa	Technology	added some building tools
7 March 2019	Emanuel Nae	Introduction	Wrote introduction
15 March 2019	Yona Moreda	Front-end	Wrote the front-end section
15 March 2019	Albert Dijkstra Niels Bugel	Back-end	Wrote the back-end section
27 March 2019	Albert Dijkstra Niels Bugel	Back-end	Made the back-end more clear
30 March 2019	Yona Moreda	Front-end	Made Figure 3 Revised Front-end section
1 April 2019	Emanuel Nae	Architecture	Made component and class diagrams
15 April 2019	Emanuel Nae	Architecture	Updated diagrams
29 April 2019	Niels Bugel	Architecture	Updated back-end section
29 April 2019	Yona Moreda	Front-end	Updated Figure 3
16 May 2019	Yona Moreda	Front-end	Made Figure 1 and 2 Updated Figure 3
28 May 2019	Emanuel Nae	Front-end	Added diagrams + review doc