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 three 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.

2.1 Back-end

The back-end will consist of two parts:

- First we have the classes that contain all the relevant data from the database.

- 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.

2.1.1 Java classes

The java classes are subdivided into multiple different parts.

- **Place**
  This category holds all the data for the following places: locations, buildings, floors, rooms, areas and cells.
- **Organization**
  This holds all the information about an organization. An organization owns certain locations.
- **Type-id**
  A type-id is an entity that is described by multiple traits.

2.1.2 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 worry about anything related to the database.

2.2 Front-end

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

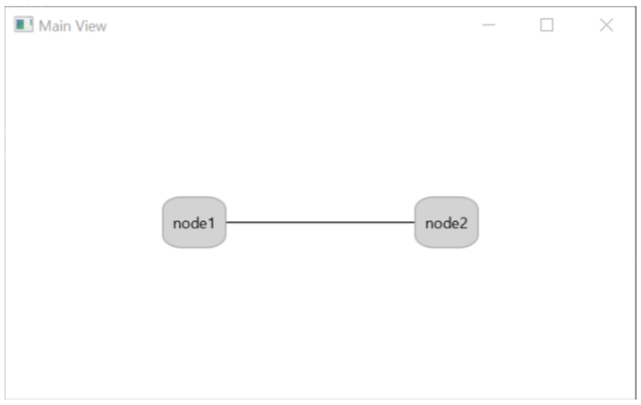

Figure 1: A demo JavaFx program connecting two sample nodes

The view package contains a collection of different JavaFX components to represent the model that is queried from database. Different JavaFX shape units are organized into classes for a low coupling and high cohesion structure. These components include Panes for layout, Rectangles for nodes, Lines for edges. To illustrate these basic components, a demo application program is shown in the above Figure 1.

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