VIS Framework Description

Project acronym: *CIMPLEX*
Project full title: Bringing CItizens, Models and Data together in Participatory, Interactive SociaL EXploratories.
Grant agreement No.: 641191

Due-Date:	Month X
Delivery:	Month Y
Lead Partner:	Deliverable Lead Partner
Dissemination Level:	Public or other
Status:	Draft / Final
Approved:	Steering Committee
Version:	Version_number

DISCLAIMER

DOCUMENT INFO

0.1 Authors

Authors	Institution	e-mail

0.2 Document History

Date	Version	Editor	Change	Status

Table of Contents

VisFramework Documentation

Documentation for the visFramework javascript visualization framework.

For information on how to set up and run the framwork see README.md.

Configuration

The client configuration is stored in config/config.js. Depending on device type, display size and anticipated user expertise, different configuations may offer advantages over others.

The VisFramework automatically recognizes mobile devices and loads the respective configuration, but users may also change it via buttons.

Default Configuration

`Config()` is a constructor that takes an ID and description object and returns a configuration object with default values.

```
class Config {
    constructor(id, description) {
        // unique ID
        this.id = id;
        // general description and attributes
        this.description = description;
        // general interface
        this.UI = {
            // title may contain HTML
            title: "Cimplex Visualization",
            ...
        };
        // filter configuration
        this.filter = {
            ....
        };
        // views' configurations
        this.timeline = {

        };
        ...
    }
}
```

Custom Configurations

You can create custom configurations by using this constructor and then overwriting only the specific attributes you want to change:

```
config.mobile = new Config("mobile", {
    name: "mobile phone configuration",
    expertise: "novice",
    device: "mobile",
    devicePerformance: "low"
});
config.mobile.UI.viewColumns = [12, 12, 12, 12];
config.mobile.UI.viewRows = 2;
config.mobile.UI.viewLimit = 2;
config.mobile.UI.viewDragAndDrop = false;
...
```

View Configurations

Views may have their own configuration inside the `Config()` constructor:

```
this.graph = {
    edgeBundling: false,
    showLabels: true,
    labelLimit: 20,
    showCommunities: true,
    // directed links may be shown as undirected to reduce clutter
    linkDirection: true,
    // direction of links can be indicated via "doubleLines", "halfArrows"
or "curvature"
    linkDirectionIndicator: "doubleLines",
    ...
};
```

Users may change some of those attributes via buttons in the view's title bar or settings panel.

Architecture

The following subsections explain the basic structure of the VisFramework.

Overview

Figure 1 shows an architecture diagram. The orange parts, i.e. services and respective connectors, have to be implemented by data service providers.

Administrators may change the Framework Configuration (blue) and third party developers may implement custom view modules.

The user interacts with the framework through the toolbar and the views.

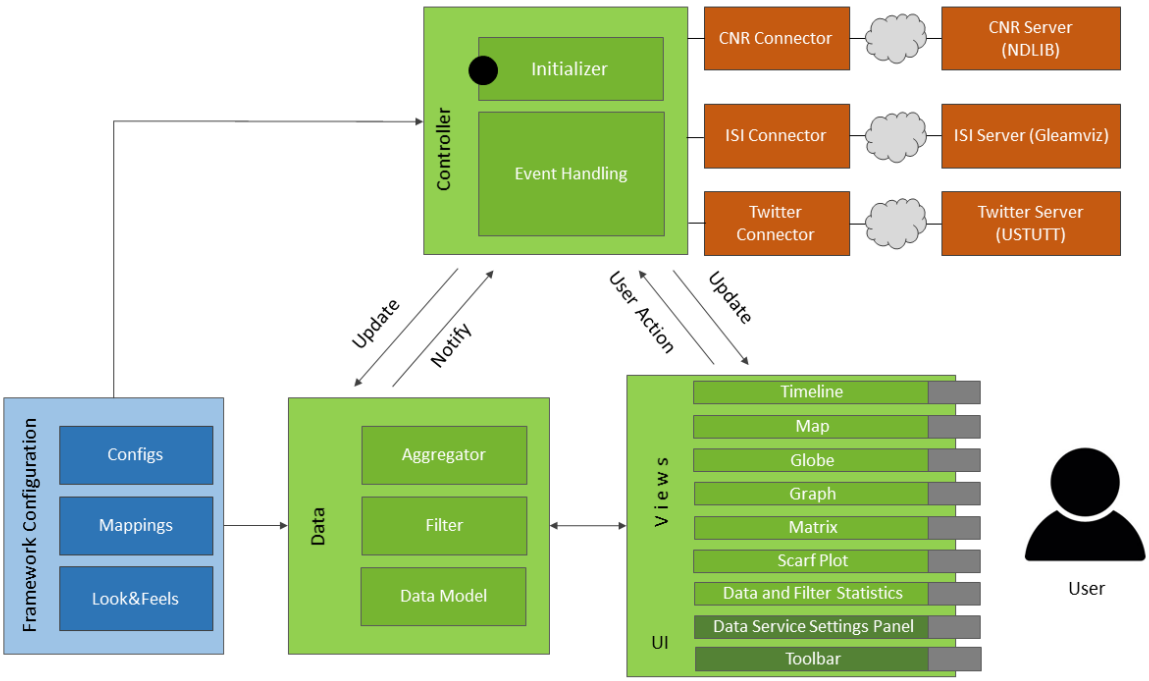

Figure 1 The VisFramework's architecture.

Configuration File

See the configuration section above.

Controller

The controller (/controller/) initializes the framework UI and event management.

Connectors

Connectors provide an interface between the VisFramework's data model and online data services. We implemented the class connectorType in order to simplify the creation of custom connectors. The file types/_connectorTypeExample.js shows how to use this class.

Data

The dataModel/ directory contains all data management files. They are explained in the following subsections.

Model

The model.js file contains a list of regions, a list of transitions and a number of general data properties, e.g. the total timespan of the data.

Filter

The filter is implemented in filter.js and stores filered results in filteredData.js objects.

Views can offer brushing or selection interactions and then use the filter events to apply certain filters to the data. Currently we support filtering on a time span, reactangular geographic area, specific regions and a single community.

Aggregator

The file aggregator.js contains function for time dependent aggregation of transitions.

Community Detection

The file community.js implements jLouvain community detection.

Visual Mapping Functions

The visual mappings from data values to colors, sizes, etc. has to be provided by the connectors. They are then cached in config/mapping.js and can be accessed by views using the global MAPPING object.

Views

You can create views using the viewType class. The file types/_viewTypeExample.js contains example code for this task. Views have access to the data and to mappings (see above). ViewType takes care of window management entirely.

Views have complete freedom in how they display their content, they may use HTML, Canvas, SVG, WebGL, etc. Canvas and WebGL are prefered for many shown items due to performance issues caused by having many DOM elements.

Hovering, clicking or tapping a view's title shows a short instruction manual for it.

Libraries

The VisFRamework uses external and internal libraries that are listed below.

External

The table below lists all used external libraries and their respective uses.

Name	Used for
Bootstrap	general page layout
Colorbrewer	color palettes that are used in color mapping
Crossfilter	data filtering
d3 v3	utility functions, scaling, interaction, colors and timeline.
d3 v4	updated packages for some functionality
d3forcebundle	force directed edge bundling with d3
Font Awesome	icons
gLayers	canvas Layer for Leaflet
jLouvain	community detection
jQuery	HTTP requests
Leaflet	geographic map
Mustache	HTML logic-less templates
Sortable	view drag and drop to reorder views
inflection.js	word inflection

Internal

The table below lists our own libraries and their respective uses.

Name	Used for
vectorSimilarity	contains functions which compare vectors in order to be able to sort them by similarity
WebWorkerPool	controls the creation of webworker threads
edgeBundlingWorker	calculates the force directed edge bundling in a separate thread
dfkiDecoder	decodes data from DFKI data services

Name	Used for
lib.js	utility functions
ui.js	UI functions

Build System

The build system automatically minimizes and concatenates all JavaScript code in order to maximize page load performance.

It is based on Node.js and executed via the following commands:

```
npm install
npm run build:development
```

or, for building a minified version

```
npm install
npm run build:production
```

Any new javascript files need to be added to scripts/build.js.