VRIJE
UNIVERSITEIT
BRUSSEL

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Computer Science

GUIDEAMAPS 2.0

Thijs Spinoy

June 2019

Promotor:
Prof. Dr. Olga De Troyer

Advisor:
Jan Maushagen

Sciences and Bio-Engineering Sciences

Abstract

Samenvatting

Acknowledgements

Declaration of Originality

Contents

List of Figures

Listings

List of Tables

1 Introduction

How to visualize, understand and remember a big amount of information which is written down in a long text? People want to store as much information as they can in their brains because the more they know, the less they have to look up and the faster they can proceed. Students for example, make schemes and summaries of their study material. The reason why they do that is because it is easier to learn and remember nicely visualised stuff in comparison to plain text. Not only the way of learning new subject material, but also teamwork can be enhanced by means of a visualization. If you write down a structure of a computer program in words, it is more difficult to discuss that structure than when the same words are translated to a drawing.

The goal of this thesis is to create a tool in which it is possible to represent linked data in a visual manner. It should not only be possible to visualize your data, but also to edit existing data as well as extending the representation with additional data. The solution is based on a visualization tool created by Janssens (2013), called GuideaMaps. This application was mainly built to provide support for the requirement elicitation for serious games. We present a reprocessing of the first version of GuideaMaps, which can be used for other purposes as well and which has a bunch of interesting improvements under the hood.

1.1 Problem statement

For the first version of GuideaMaps, the main goal was to develop the following:

> A tool that allows the different people (and with different background) involved in the development of a serious game (e.g., against cyber bullying) to brood over the goals, characteristics and main principles of a new to

> develop serious game. The tool should be easy to use and usable in meetings. Therefore, we want to explore the characteristics and capabilities of a tablet (i.e. iPad). (Janssens, 2013)

By specifying the goal in this way, end users of the application are restricted to some factors. First, they need an iPad to be able to use the application. It has to be an iPad, and not an other kind of tablet with a different operating system, because GuideaMaps was created and designed for iOS only. In some cases this can be a hard restriction, e.g. suppose one of the participants of the meeting does not have an iPad. Then there is no solution for that person to let him use the application.

Further, the tool is mainly created for the purpose of serious games. This means that the nodes in the visualization have an almost fixed layout. You can edit the background color, but you cannot define your own representation of a node (e.g. change the length and width). Not being able to do that is a limitation in the sense that for some purposes this default visualization may not be very useful. In those cases, users will often search for other visualization tools where is possible.

1.2 Research questions

The problem statement discussed in the previous section indicates that the first version of GuideaMaps comes along with some restrictions. Therefore, the following research questions about an optimized version of the application can be posed:

Question 1

Can we create a version of GuideaMaps that works on all devices and all different operating systems?

Question 2

Can we make that application generic in such a way that it can be used for different purposes? If so, is it possible to make it customizable so that the user can define its own layout for the visualization?

This thesis presents a solution, called *GuidaMaps 2.0*, for the problem statement with the research questions taken into account. How the tool gives an answer to the research questions mentioned above is explained into detail in the continuation of this thesis.

2 Related work

There exist lots of ways to visualize data and the relation between data. In this chapter, we discuss some of the possibilities related to what we need for GuideaMaps.

2.1 Mind maps

The most well-known technique to visualize related data is to create a mind map (a.k.a. idea map). This technique is mainly used to show the relation between portions of information and for brainstorming purposes. Other applications where this technique is used are note-taking, problem solving, etc. (Balaid et al., 2016)

Mind maps are created by writing the main idea in the middle of the drawing, while all sub-ideas are placed around that center node. Each sub-idea is connected with its parent by means of a line. Hence, this kind of visualization is not difficult to create or understand. Its simplicity is one of the reasons why it is used a lot in practice.

Because mind maps is not a new concept, but one that most people already know quite well, we will not discuss it into further detail.

2.2 Knowledge maps

According to Balaid et al. (2016), *knowledge maps* is an umbrella term for tools and techniques like mind maps. O'Donnell et al. (2002) defined the concept as follows:

> Knowledge maps are node-link representations in which ideas are located in nodes and connected to other related ideas through a series of labeled links.

This kind of representing information has for example a positive impact on students. The paper of O'Donnell et al. (2002) learns us that students using knowledge maps are better in remembering the main ideas of the subject in comparison to the ones that study from the text without the visualization. For GuideaMaps it is less important that the users remember the content of the visualization, but on the other hand, we have a node-link representation with the main idea in the center.

Next to mind maps, concept maps is a second technique included under the umbrella of knowledge maps. Concept maps are similar to mind maps in some sense but they do have some different characteristics. First, the purpose of a mind map is to associate ideas, topics or things, while concept maps illustrate relations between concepts. Further, the structure of a concept map is mostly hierarchical and visualized like a tree. On the other hand, mind maps sometimes have a radial layout and not hierarchical. (Davies, 2010)

Hence, we can state that GuideaMaps makes use of knowledge map visualization and more specifically some kind of combination of mind maps and concept maps.

3 Requirements

The application should meet a lot of requirements in order to deliver some quality to the users. End users of a system often qualify a system as *good* if it provides the right kind of functionality, if it is easy to learn and easy to use. Hence, there are some functional and usability requirements the system should meet in order to be evaluated as *good* by the end users.

3.1 Functional requirements

Device- and OS-independent

The application should run on different kinds of devices (e.g. tablets, laptops and desktops) and operating systems (Android, iOS, MacOS and Windows). While version 1.0 was only designed for an iPad, our version should provide a solution for this restriction. One of the possibilities to solve the problem is to create the application in the browser. Hence, the user must not be limited to a particular device anymore, the tool can be used on any machine. The only restriction on the used device is that it needs to have a screen that is large enough because it is not very convenient to work with the visualization on small screen areas, e.g. on smartphones. The application could run on smartphones but it is not recommended nor required to use it on devices with relatively small screens.

Genericity

The tool should be generic in such a way that it is possible to create a different design for the nodes and the links. The end user should not be restricted to the predefined design of GuideaMaps. Instead, a developer can create its own implementation for the nodes and the links, which then can be *plugged in* into

the system. The core of the application should be completely separated from the parts that are customizable for the end-users.

Rights
There should be two possible modes in the system: an end user mode and a map creator mode. A map creator has the rights to change everything about the nodes, while the end user is restricted in which data he can edit. This requirements can be seen as a security mechanism to not let a particular end user mess with the data.

Zoom the visualization
As a user, you sometimes want to zoom in or out such that you get less or more information at the same time on the screen. For example, a feature to zoom is very useful in situations where you want to compare different parts of the visualization. The scrolling gesture is probably the best gesture for this action, because this is a well-known way to zoom in applications (e.g. Google Maps). Users don't like it when every application has a different gesture for the same action. Hence, it won't help them if we chose for another gesture than scrolling in GuideaMaps. Also, using the same gesture as in other applications improves the memorability and learnability of our tool.

Zoom to fit
A variation on the zooming feature is zoom to fit (a.k.a. zoom to bounding box). With custom zooming, the user can set the zooming level to meet its needs. On the other hand, zoom to fit adapts the zooming level and moves the content of the application until everything fits on the screen.

3.2 Usability requirements

Next to the functional requirements, the application must meet a lot of other requirements, e.g. in terms of usability. In the bachelor course *User Interfaces* (De Troyer, 2016) two definitions of usability were provided:

Definition 1
Usability is a measure of the ease with which a system can be learned and used, its safety, effectiveness and efficiency, and attitude of its users towards it.

Definition 2
Usability is the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

In order to conform these definitions, we can formulate several usability requirements the application should really meet. Later in this paper, when we explain the implementation details, we will come back to these requirements and discuss how we managed to meet them.

Intuitiveness

It is important to keep in mind that the tool will not only be used by people with experience in Computer Science. It doesn't matter whether or not the user has a background in Computer Science, he should be able to easily learn to use the system in short time. Therefore, it is important to choose for clear, not misunderstandable icons on buttons where a click on this button invokes a certain action. Some examples of these actions are (1) adding a child node, (2) explore and edit a node and (3) expand/collapse a node. Good icons[1] for each of the mentioned actions are shown in figure 3.1. The icons are not ambiguous, they can only be linked to one particular action and thus the user knows exactly what to expect when clicking on the button. Well chosen icons are one of the factors in the design that help users to remember how to use the system. If they recall the meaning of the icon immediately when they see it, the users will experience the system as easy to learn and easy to remember.

If the user knows it is not possible to add child nodes, we would state that a plus- and minus-icon are also possible to indicate the possibility to expand or collapse a node. If the visualization allows to add child nodes, the plus-icon should be used for this action and not for the expand- or collapse-action.

(a) Plus (b) Explore/Edit (c) Expand (d) Collapse

Figure 3.1: Good icons for the following actions: (a) add child node, (b) explore and edit node, (c) expand node and (d) collapse node.

Customization

End users should be able to customize the visualization a bit to their needs. It should be possible to change the background color of nodes, to add child nodes and to remove nodes they don't need anymore.

[1] `https://fontawesome.com/`

4 Implementation

4.1 Technologies

A number of technologies assisted in the developing process of the application. In this section, each of these technologies is discussed to explain its importance.

4.1.1 ReactJS

ReactJS is an open source library to create user interfaces[1]. One of its main goals is to provide the best possible rendering performances[2]. Performance is high because ReactJS allows developers to break down the user interface into different components. Each component has its own *state*, which contains information about the content of the component. This state can be updated while the application is running and if such an update is made, only this component is re-rendered instead of re-rendering the complete UI[3]. Hence, this involves a huge benefit for the performance. Next to that, it is also not very hard to learn to code in React when comparing to other frameworks (e.g. AngularJS). If the developer knows HTML and JavaScript, he will be able to code in ReactJS quickly.

Because of these benefits, ReactJS is the framework in which the GuideaMaps application is implemented. Each node is considered as a separate and unique React Component. The most important reason for implementing the nodes like this is performance: if the state of the node is updated, only this node is re-rendered and not the complete UI.

[1] `https://reactjs.org/index.html`

[2] `https://medium.com/@thinkwik/why-reactjs-is-gaining-so-much-popularity-these-days-c3aa686ec0b3`

[3] `https://facebook.github.io/react/docs/why-react.html`

4.1.2 d3

Another helpful tool is d3. With d3-hierarchy[4], it is possible to transform JSON-data into hierarchical data. Having this kind of data makes it much easier to create a tree- or cluster-structure. In the case of GuideaMaps, a clustered visualization is very useful. The *main*-node (a.k.a. the root node) is then positioned at the center, such that its child nodes can be places around it. Hence, the farther a node is away from the center, the lower it is in the hierarchy. Also, the visualization will not be messed up by positioning the nodes in this way, because every node has exactly one parent. This means there won't be a spaghetti of links where you cannot see which node the link comes from and which node it is pointing to.

4.1.3 Tailwind CSS

The style of the application is very important for the end user. Everything should look pretty and, as already mentioned in section 3.2, the possibilities should be straightforward and visible. While developing and creating a beautiful style for applications and websites, the code for these styles can become a big part of the implementation. Hence, a good framework is necessary to reduce the lines of code to a reasonable number. It also helps to improve the readability of the code.

Tailwind CSS[5] is a framework that assists developers to style their application. The difference with more famous frameworks, like Bootstrap, is that Tailwind CSS has no default theme. If you want to use a Bootstrap-feature, this eventually comes along with other features you don't always wanted and it can be quite hard to undo the part you didn't ask for. With tailwind on the other hand, you can grab only the features you want, without side-effects. Figure 4.1 shows an example with two small listings. The first uses inline style while the second makes use of tailwindCSS.

```
<div
  style={{
    position: absolute,
    border: 1px solid black,
    borderRadius: 0.25rem,
    padding: 0.5rem,
  }}
/>
```

Listing 4.1: Normal CSS, no tailwind.

```
<div
    className={
      'absolute border
      ↪ border-solid
      ↪ border-black rounded
      ↪ p-2'
    }
/>
```

Listing 4.2: With tailwind CSS.

Figure 4.1: Difference when using tailwind CSS or not.

The figure illustrates the difference to implement four css property-value pairs in

[4] https://github.com/d3/d3-hierarchy
[5] https://tailwindcss.com/

normal css and implementing the same four with tailwind. In the case of normal css, we need four lines of code to retrieve the intended result. On the other hand, with tailwind css, we add some classes providing the same style. The classnames can be placed on a single line instead of four. This example proves that the number of lines can be decreased.

4.2 Structure

With the technologies mentioned in the previous section, the most important pillars the application relies on are discussed. Now it is time to have a look at the structure of the code, such that it is clear how all elements work together. Figure 4.2 shows a visualization of the structure of the code.

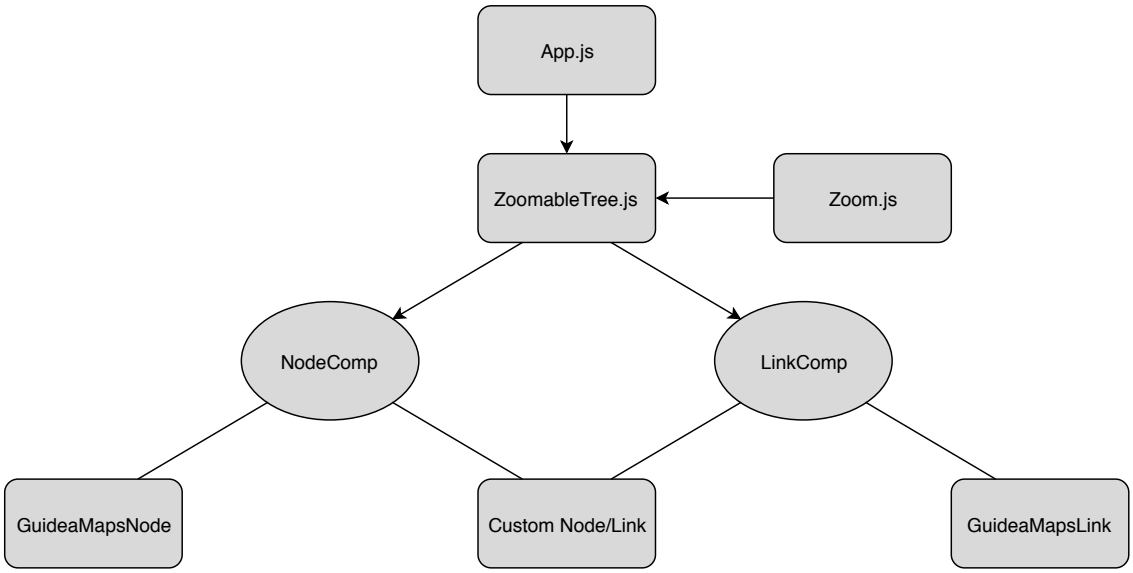

Figure 4.2: Structure of the application.

The application is developed in such a way that it can be used as a library for other purposes than GuideaMaps as well. The general, always-returning part of the code can be found in App.js, ZoomableTree.js and Zoom.js, where the layout of the nodes is defined as well as the implementation to allow the user to zoom the visualization in and out. When talking about the library, the layer of the NodeComp and LinkComp is very interesting and important. For the GuideaMaps, an implementation for a GMNode and GMLink is provided. Each implementation describes what every node and every link should look like in the visualization (TODO: provide more explanation about this somewhere else + coupling with requirements and RQs).

The strength of the library can be seen when a particular user would like to have a different representation for the nodes or the links or both. In that case, new components (e.g. MyCustomNode and MyCustomLink) should be implemented. In the code, only one line should be adapted: in App.js, ZoomableTree is called

with a certain number of props. Two of these props are NodeComp and LinkComp, which are set to GMNode and GMLink, respectively, by default. Hence, the only action that is required to *plug in* an other component is replacing GMNode by MyCustomNode and GMLink by MyCustomLink. Figure 4.3 shows the part of the code in App.js that should be adapted as explained. Note that NodeComp and LinkComp are not the only props that are passed to ZoomableTree. The others are omitted to improve the readability.

```
<ZoomableTree
    NodeComp={GMNode}
    LinkComp={GMLink}
/>
```

Listing 4.3: Default components.

```
<ZoomableTree
    NodeComp={MyCustomNode}
    LinkComp={MyCustomLink}
/>
```

Listing 4.4: Custom components.

Figure 4.3: Two listings showing how to use the library.

4.3 GuideaMapsNode

Now the overview of the structure of the application is discussed, we will consider the implementation details of the GuideaMaps visualisation in this section.

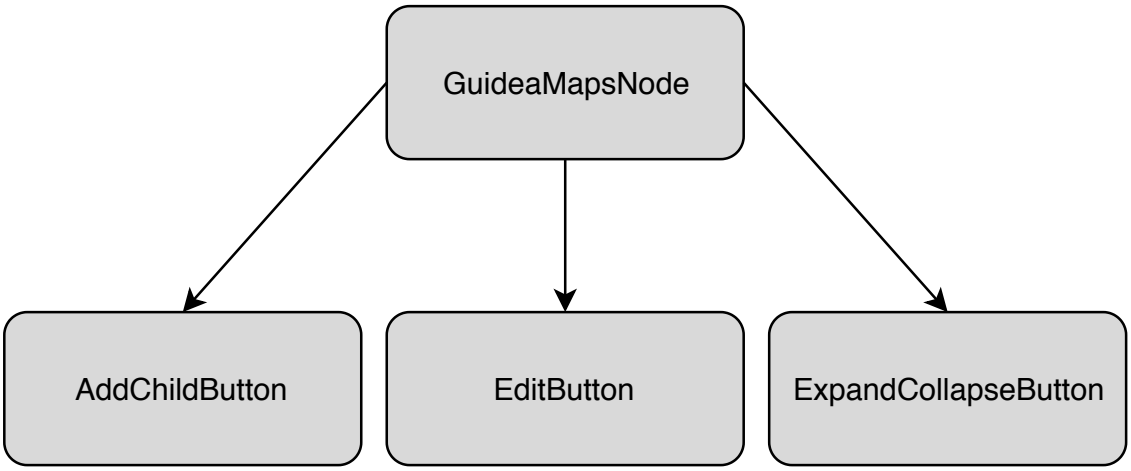

Figure 4.4: Structure of GuideaMapsNode.

As you can see in figure 4.4, the structure of a GuideaMapsNode node quite easy. Each node consists of three html *div*-elements: a title-div, a content-div and a controls-div.

4.3.1 Title

The title-div is positioned at the top of the node. It consists of the title text if this is set by the map creator. Otherwise, it shows *Insert title* in italic to remind the

user that he still had to set a title for this node. An end user can set the title if no title exists yet, but he can never change it.

Next to the title-text, the title-div contains an icon placed near the right border. This icon indicates whether all information, a title and some content is provided or not. The icon that is shown also depends on whether the information in its child-nodes is filled in or not. We distinguish three possible situation:

1. The data of the node itself and of all of its children are correctly filled in. In this case the icon will be a completely filled circle.

2. The data of the node itself and of all of its children are all empty. In this case the icon will be an empty circle.

3. In all other cases, the data of the node itself or of at least one of its nodes is (partially) filled in. Then the icon will be a semi-filled circle.

This icon can assist the user in determining whether all information is filled in. For example, if he checks the root node and sees that the circle is completely filled, he does not have to check every child-node to know that all information is filled in. On the other hand, suppose only one node is not yet provided of some content. If the user then starts from the root node and always follows the child-node with a semi-filled circle, he will find the incomplete node much faster than in the case he has to check all the nodes.

4.3.2 Content

The content of each node is just some text describing the information corresponding to the title. As long as no content is provided by the user, a description is shown in italics to assist the user in which content is expected. The background color of the content-div is made customizable.

4.3.3 Controls

The last part of a node is the controls-div. With *controls* we mean the different actions that a user can take concerting the particular node. The controls consist of three buttons: one to add a child node, one to "open" the node to view and edit the data and one to expand or collapse the node to show or hide the child nodes, respectively. When a node is collapsed, all child nodes on all lower levels in the hierarchy are hidden. On the other hand, when a node is expanded, only the child-nodes of the next level in the hierarchy are shown.

4.4 Other visualizations

Describe Plateforme DD here.

5 Evaluation

6 Conclusion & future work

6.1 Conclusion

6.2 Future work

Describe future work here.

References

Balaid, A., Rozan, M. Z. A., Hikmi, S. N., & Memon, J. (2016). *Knowledge maps: A systematic literature review and directions for future research.*

Davies, M. (2010). *Concept mapping, mind mapping and argument mapping: what are the differences and do they matter?*

De Troyer, O. (2016). *Computer Science, Lecture Notes: User Interfaces.* Vrije Universiteit Brussel.

Janssens, E. (2013). *Supporting requirement elicitation for serious games using a guided ideation tool on ipad.*

O'Donnell, A. M., Dansereau, D. F., & Hall, R. H. (2002). *Knowledge maps as scaffolds for cognitive processing.*