Visualising networks in the human brain

0. Summary

To understand and communicate networks of modules, functional groups of brain tissue, that do not necesserally reside in only one region of the brain, inferred from eeg and meg data in human brains ThoughtWeave was built. To understand the module networks (what causes them to occupy which brain regions and their effects there) better, researchers relate them to other properties of the eeg/meg sensor data. ThoughtWeave makes observing and comparing these modules and properties easier, by offering a way to take these properties, and use them to configure a 3D visualisation of the eeg/meg sensors and their interrelation.

1. Overview

Brain research is moving from a biological understanding to a functional understanding of the brain. Identifying functional modules and how they interrelate is one such approach to improve functional understanding of the brain.

The goal of ThoughtWeave is to help researchers find relationships in

- EEG/MEG data,
- the relationship of this data to the functional modules, and
- the relationships between the modules

Many applications already exist to assist researchers in viewing biological structures and electrical charge in the brain. However, no application exists yet that emphasizes viewing these functional modules and their relationships.

ThoughtWeave tries to accomplish this by offering a graphical user interface to let the researchers visualize their eeg/meg based measures. These measures, that among other things identify these modules, are computations based on either single eeg/meg sensors, relations between two sensors, or as in the case of modules, whole groups of eeg/meg sensors. The measures of these sensors and their relations are drawn as properties of a network diagram. Modules can be visualised in several ways, as will be shown in later chapters.

To ensure that ThoughtWeave fulfils its goal of helping the researchers, a set of requirements were agreed upon. Using these requirements a user interface can be then be described. Based on the user interface description, the design of ThoughtWeave itself can be presented.

The user interface reads the data, so the data itself must be understood first. The user interface visualises this data, so decisions must first be made on how this visualisation is done. The visualisation is based on the data, so the data must be treated before the visualisation. But because all sections contribute to fulfilling the requirements, the requirements must be treated first.

The following sections treat these issues,

Section 2 describes the requirements;	what requirements were agreed on?
Section 3 treats the data in detail;	what is represented and in what format?
Section 4 treats the means of visualisation;	how will properties be represented visually?
Section 5 treats the user interface;	how can data be presented visually, in a way that fulfills the requirements?
Section 6 treats the design of ThoughtWeave;	what components does ThoughtWeave consist of and how do they make the user interface possible?
Section 7 concludes that the goals are met.	ThoughtWeave indeed fulfills the requirements

2. The requirements

As stated in the overview it was necessary to establish requirements that could serve as objectives for fulfilling the goal.

This section presents the requirements that were set. The requirements were set before the start of the project, at the start and after development started.

- The requirements before the assignment served as a project description and were acquired in a discussion of W. de Haan and D. van den Berg.

- The requirements at the start of the project were acquired during an unstructured interview of W. de Haan, taken by G. Sales Calvano.

- The requirements taken after development started were acquired during several conversations and using feedback from W. de Haan.

But before these acquired requirements can be understood it is necessary to first have some knowledge of the context. Then the requirements agreed upon by van den Berg and de Haan can be understood, and finally the requirements acquired by S. Calvano and de Haan can be understood.

2.1. Context

As stated in the overview; neurology is moving from research on the anatomic structure of the brain to research on its functional organisation. The research group requesting ThoughtWeave try to discover this organisation by identifying sets of cooperating brain regions they call modules.

2.1.1. Modules in the brain

This operation of brain regions consists of neurons transmitting electrical charge, which in turn excite other neurons to transmit electric charge too. A neuron will only send this charge for a short duration, and then it becomes insensitive to excitement for a short while. Thus the signal it transmits becomes a pulse.

As neurons start to transmit these pulses they can start to fire synchronously. The neurons trigger eachother precisely when they become sensitive again, thus causing a rhythm of activation within a whole group of neurons.

Such a group is called a module.

The grouping of neurons can be observed by measuring the displacement of the electrical charge of these pulses, using eeg and meg sensors. These sensors measure mainly cortical activity (the cortex is the surface of the brain), due to the limited depth at which the magnetic and electrical fields can be measured.

It turns out that these modules form scattered accross the cortex, and not just in specific regions. And some regions tend to synchronize more within the module, and some tend to synchronize with regions outside the module as well.

2.1.2. The effect of Alzheimer's disease

Alzheimer's disease gradually damages the brain. The exact mechanism is still not entirely known. There is good evidence though that it is caused by a protein that should not be produced, and breaks apart. One of the parts of the broken protein fit a type of receptor that is found on some cell membranes and tells the cell to kill itself.

This causes some parts of the brain to die off, and zones in the brain get a much lower cell density.

Some Alzheimer patients suffer huge deterioration in their mental capabilities shortly after the disease hits them, others have huge deterioration in their brain, but with little damage to their mental capabilites.

It appears that some regions in the brain are relatively redundant and some are crucial. The research tries to understand how the modularization of brain regions changes during the progression of Alzheimer's disease. Can brain regions be identified that are critical to mental functioning, can brain regions be found that are critical to module formation and are they the same?

2.1.3. Emphasizing the differences and similarities between brains

In essence, what the research group does is study the differences and similarities between brains. Brains at different stages of Alzheimer, are compared for various computed and directly measured properties. Properties that can be about a single measurement location, or about a relation between these measurement locations.

2.2. Requirements agreed on earlier

Several requirements were agreed upon earlier in a discussion between D. van den Berg and W. de Haan. They are quite clear and therefore presented here without further writing.

The presented requirements were originally written in Dutch. For the original see appendix A.

Technical requirements
[1 NodeCount512] The program should be able to visualise up to 512 nodes well. More nodes are desireable but not strictly necessary.

[2 DataFormat] The input format of the networks is ASCII text as used on the department of clinical neuropsychology of the VUMC.

[3 VisualisationClear] Both networks with weighted connections and unweighted connections should be presented in a visually appealing way. The difference must also be clear.

[4 LabelOptions] It should be possible to label different nodes, through a label file. It should be possible to turn labels on and off.

[5 ImageSave] The image should be saveable in an every day format (JPG, BMP or PNG).

[6 AccessibleProject] All source code will be open source, handed in, be programmed clearly, documented, and easily recompiled.

Requirements for an extended project

[7 ColorsAdaptable] Visualisation of colors of background, nodes and edges should be easily adaptable.

[8 ShapeAdaptable] The shape of nodes and edges should be easily adaptable

[9 DragDropNode] The application will have a drag drop interface in which nodes can be dragged.

[10 Layouts] The network will have several network layouts; (a) circular, (b) manual, (c) coordinate-driven and (d) anatomical.

[11 Layouts3D] The application has 3D visualisation for the aforementioned layouts.

[12 Rotate3D] The application has 3D rotation capabilities for all aforementioned layouts.

[13 Movies] It is possible to make a movie of consecutive layouts. The movie can be saved in a common format.

2.3. Acquired Requirements

During several interviews of de Haan some aditional requirements were found. Some were deduced from the requirements in the interview and observations on the research team.

Some were accepted and implemented, some were rejected.

2.3.1. Accepted requirements

[14 Comparison] The goal of ThoughtWeave is to compare various measurements on brains. I.e. compare different measurements on the same brain, or the same measurement on different brains.

[15 IrrelevantHidden] Because the data presented can get quite large, making it quite confusing, it is necessary to make irrelevant aspects transparent.

[16 ReferenceBrain] They want a model of a brain to make it easier to see what brain region was actually measured.

[17 FlexibleFormat] During development of ThoughtWeave the computed properties on measurement locations changed continuously. Also several measures for computing groups and relationships between measurement locations were added. Thus the information in the files changed continuously as well. ThoughtWeave is able to handle these changes, within the limits of the file format used by the researchers at the start of the project.

[18 Highlighting] It should be possible to highlight specific nodes and/or links visual properties, by setting their visual properties/labels manually.

2.3.2. Rejected requirements

Though it would have been valuable, blinking was rejected as a means of visualising data properties.

Fileformats themselves were often subject to small yet problematic changes. However, ThoughtWeave will only accept one file format.

3. The research data

As stated in the overview it was necessary to understand what data was to be visualised. This section elaborates on what data is visualised and how it is represented in files.

As stated in the overview section various properties were computed for both individual sensors and relationships between sensors. The requirements elaborated on the data format in two objectives;

[2 DataFormat] The input format of the networks is ASCII text as used on the department of clinical neuropsychology of the VUMC.

[17 FlexibleFormat] During development of ThoughtWeave the computed properties on measurement locations changed continuously. ... ThoughtWeave is able to handle these changes, within the limits of the file format used by the researchers at the start of the project.

To fulfil these requirements it is important to understand what data these formats contain specifically.

The research group uses a file format for properties computed for single sensor zones and for relationships between sensor zones.

To assist fulfilling requirement [10 Layouts] a file format was introduced to allow for specifying visualisation properties for the sensors. This file type can contains things like locations of the sensors, labels, etc. For more information on visualisation see the next chapter.

3.1. Node data

These files contain lists of properties followed by a value per sensor. The order in which sensors occur in the file identifies them.

3.2. Relationship data

It is possible to compute/measure relationship properties between nodes. For such a relationship property, a relationship data file contains a “from to” matrix for, where row index and column index denote the from/to node, and these node indexes follow the same sequence as in node data.

3.3. Visualisation data

Visualisation data contains default values for x, y, z locations, labels for nodes, default colors, etc.
This file format was created to allow researchers to specify the locations of the sensors and their name, but was made more general, because it was very easy to accomplish and made them much more useful in their application.

Each row in the file contains a label, followed by all the visualisation values of one sensor. Sensors occur in the same sequence as in node data files.

3.4. Conclusion

In order to fulfil [2 DataFormat] and [17 FlexibleFormat] it’s necessary to

4. Visualisation

As stated in the overview it was necessary to define the way ThoughtWeave would create a visualisation. The requirements section established various requirements on visualisation;

[3 VisualisationClear] Both networks with weighted connections and unweighted connections should be presented in a visually appealing way. The difference must also be clear.

[4 LabelOptions] It should be possible to label different nodes, through a label file. It should be possible to turn labels on and off.

[7 ColorsAdaptable] Visualisation of colors of background, nodes and edges should be easily adaptable.

[8 ShapeAdaptable] The shape of nodes and edges should be easily adaptable

[10 Layouts] The network will have several network layouts; (a) circular, (b) manual, (c) coordinate-driven and (d) anatomical.

[11 Layouts3D] The application has 3D visualisation for the aforementioned layouts.

[14 Comparison] The goal of ThoughtWeave is to compare various measurements on brains. I.e. compare different measurements on the same brain, or the same measurement on different brains.

[15 IrrelevantHidden] Because the data presented can get quite large, making it quite confusing, it is necessary to make irrelevant aspects transparent.

[18 Highlighting] It should be possible to highlight specific nodes and/or links visual properties, by setting their visual properties/labels manually.

This section presents how the research data is visualised, such that the aforementioned requirements can be met. How the visualisation is actually configured by the user, thus completely meeting these requirements, is dealt with in the next chapter.

Before showing how the networks can be visualised clearly [3 VisualisationClear] and in a way they can be compared [14 Comparison], or [18 Highlighting] specific nodes can be highlighted, it is necessary to understand which visual properties can be used to reflect the networks. Before the visual properties of the visualisation can be described, and how they meet the other aformenioned visualisation requirements, first the general structure of the visualisation must be described.

4.1. The general structure of the visualisation

To visualise the sensors, and their relationships, they are drawn out as a network. A node represents a sensor, a link between two nodes represents the relationships between the two sensors those nodes represent. Data properties of the sensors and their relationship properties can be visualised by representing them in the visual properties of the nodes and the links.

For clarity, sensor data properties and sensor relationship properties will be collectively refered to as source properties.

4.2. Visual properties

Position

[10 Layouts] [11 Layouts3D]

The x, y and z coordinate of nodes can be used in various layouts and they can be made to reflect meaning

Color

[7 ColorsAdaptable]

Color components red, green and blue can be used to reflect meaning on both links and nodes. These color components can be mixed.

Form
[8 ShapeAdaptable]

The length of spikes, and the size of nodes can be given meaning.

The thickness of a link can be given meaning.

Hiding
[15 IrrelevantHidden]
Nodes and links can reflect meaning with a degree of transparency.

Nodes and links can alsof reflect meaning by being either visible or invisible. This has the same outcome as transparency in its extremes.

Though this is in itself useful to have an extra invisibility property it is also forced by performance constraints.
One of the requirements demands the capability to visualise 512 nodes. Visualising each relation requires approximately 130 000 links. This turned out to be computationally intractable using Java3D. In the case of transparency the graphics engine still needs to take into account the transparent shape. When a shape is invisible it does not exist in the graphics engine.

Most importantly invisible shapes aren't even loaded into the graphics engine at initialization.

Label

[4 LabelOptions]

Each node can be given a label.

4.3. Conclusion

A brain data network can be visualised clearly by configuring a visualisation such that its visual properties reflect the brain data networks properties clearly. Thus [3 VisualisationClear] is met when the network can be easily configured by the researchers.

By configuring the visualisation such that the visualisation properties of multiple brain data networks are represented, brain data networks can be compared. Thus [14 Comparison] is met when the network can be easily configured by the researchers.

Any node or link in the visualisation can have their specific visual properties set to different values. Thus [18 Highlighting] is met when nodes and links can be easily configured using a user interface.

5. User Interface

As stated in the overview ThoughtWeave was to help researchers find relationships in their data. To do this a user interface was required, that allowed them to visualise their data, explore it and communicate it, such that ThoughtWeave fulfilled the requirements that were agreed upon.
Section 3 described the data formats used by the researchers, and section 4 described a framework for visualisation. Section 3 and 4 could meet several requirements from section 2 when a gui was presented in this section.

Other requirements in section 2 this section will deal with are;

[5 ImageSave] The image should be saveable in an every day format (JPG, BMP or PNG).

[12 Rotate3D] The application has 3D rotation capabilities for all aforementioned layouts.

[13 Movies] It is possible to make a movie of consecutive layouts. The movie can be saved in a common format.

[16 ReferenceBrain] They want a model of a brain to make it easier to see what brain region was actually measured.

To visualize the data it must be accessed on the hard disk and mapped to visual properties. Visual properties that are not used and other global graphical constants must be set to sensible values. Important nodes and important links should be emphasized.

To explore the visualised data, it is necessary to look at the 3D visualisation from various directions. To communicate the visualised data, the visualisation must be saveable.

To get good footage/images of the data, it must be visualised first and then explored.

Data must be loaded before it can be visualised, so it must be treated before the gui components for mapping data to visual properties can be treated. Before emphasizing values can be understood it is necessary to understand how general mappings can be made and sensible values are set on visual properties that are not used.

Finally some global visualisation settings like background color in [7 ColorsAdaptable] and [16 ReferenceBrain] require user interfaces to set them. This issue is independant of other issues, and no other issues depend on it.

The following sections treat these issues such that understanding each section depends only on understanding sections treated earlier.

Section 5.1. treats loading the data
Section 5.2. treats global visualisation settings
Section 5.3. treats setting visual properties
Section 5.4. treats emphasizing special nodes and links
Section 5.5. treats observing the visualisation
Section 5.6. treats saving screenshots and recording movies
Section 5.7. concludes the goals in the requirements were met

5.1. Loading data

It was necessary that the researchers could load their data into the application. Otherwise there would not be much to visualise obviously [2 DataFormat][17 FlexibleFormat].

ThoughtWeave allows loading sets of data files contained in a folder. This includes files with visualisation data. Using File-> Open under the menu, the user can browse for a directory containing all data files and any optional visualisation files. The data in these files is now available throughout ThoughtWeave, whenever some mapping or other task requires it.

5.2. Global visualisation settings

As stated in the introduction to this section, the requirements in section 2 required a way to set some global visualisation settings; a 3D model of the brain, as a reference for the location of the sensors [16 ReferenceBrain], and some way to set the background color [7 ColorsAdaptable]. This section presents these two topics in the next two subsections.

5.2.1. Show brain

The researchers wanted to have the option of displaying a brain as a reference for where the sensors are [16 ReferenceBrain]. A brain can be toggled under the menu option Options -> Show Brain.

5.2.2. Background color

Visualisation of colors of the background needed to be adaptable [7 ColorsAdaptable]. The background color can be adapted in two ways; using the keyboard and using a form. The form is accessible under Window-> Background Color.

5.3. Setting Visual Properties

As stated in the introduction to this section, the gui must offer options for mapping data properties to visual properties, and setting the remaining visual properties to sensible values. But before this can be understood it is necessary to understand the mapping editors that offer these options.

The first subsection treats the mapping editors, the second subsection treats the data mapping types, the third subsection treats setting sensible values to unused visual properties.

5.3.1. The mapping editors

A mapping editor allows editing a list of mappings, and selecting a visualisation data file as a source for labels. A mapping sets visual properties based on data properties. A mapping can map values for both nodes and links.

There are two mapping editors; the node mapping editor, and the link mapping editor. The node mapping editor allows creating mappings for the nodes, and the link mapping editor allows creating mappings for links.

5.3.2. Data Mapping types

As stated in the introduction it was necessary to offer options to map data from the files onto visual properties, to allow the researchers to visualise their data [3 VisualisationClear], and compare it [14 Comparison].

This section treats those data mappings. To select data a source selector is used. To select a visual property a visual property selector is used. Before data mappings can be understood these must be briefly explained first. Then the actual data mappings are treated.

5.3.2.1. Selectors

To map data properties onto visual properties, the data properties and the visual properties must be selected. A data property is selected using a source selector, and a visual property is selected using a visual property selector.

A visual property selector offers all unused visual properties from a drop down menu.

A source selector offers a list with all loaded files. From each file the available data properties can be selected. In the case of visualisation data files these are the visualisation properties x, y, z, red, green, blue etc. In the case of node data files they are all the data properties the researchers have put in there.

5.3.2.2. Property Mapping

A property mapping takes the property selected in a source selector and maps onto a property selected in a visual property selector. To prevent problems due to huge differences between the values or very small differences, the property mapping transforms all property values such that the smallest value becomes 0, and the highest becomes 1.

The values can be transformed differently if required by changing the values in the min and max boxes. The min value is becomes 0 when transformed to visual property, the max value becomes 1. All values in between min and max values are scaled between 0 and 1 proportional to their distance to the values in min and max.

This option can be used not only to visualise and compare measured and computed data, it can also be used to take visualisation data files and display their contents, thus offering the option of showing manual and anatomic layouts in all available dimensions [10 Layouts] [11 Layouts3D].

5.3.2.3. Module Mapping

Module mapping is a custom mapping dedicated to visualising values that denote group membership, though it can visualise other data properties as well.

The typical case; the researcher takes a data property indicating the module a sensor is clustered with, and the mapping tries to assign the sensors a color indicating the module it belongs to. The color palette is selected to make the differences between colors as big as possible.

The general case;

It takes a single data property (selected using a source selector) and maps it onto a number of selected visual properties, such that each unique data value maps to a unique combination of values for the visual properties. The combinations are chosen to maximize the difference between them.

The way unique combinations are chosen with maximized difference is by taking a hyper cube with the same number of dimensions as there are visual properties. The hyper cube is partitioned into the smallest number of smaller hyper cubes equal to or larger than the number of unique data properties, while partitioning the entire volume inside the large hyper cube. On a 2D hyper cube this will form a grid of squares, on a 3D hyper cube a grid of cubes will form, etc.

The combinations are taken from the coordinates of the corner points of the smaller hyper cubes.

5.3.2.4. Range

A range tests a selected data property on a threshold. If it is larger than this threshold its corresponding visual propety is set to 1, otherwise it is set to 0.

It has several options for setting such a threshold. The n highest data values can be used to determine the threshold. The n lowest data values can be used to determine the threshold. And the threshold can be set manually.

5.3.3. Constant Mapping types

As stated it was required to be able to set sensible values on unused visual properties. There are two mapping types dedicated to setting such unused visual properties. The constant mapping, and the circular mapping. The circular mapping is one of the required layouts [10 Layouts].

5.3.3.1. Constant

A single visual property can be selected and set to a constant value.

5.3.3.2. Circular

As stated in the requirements a circular layout was required [10 Layouts].

The circular layout allows setting two selected visual variables using a circle function. In the case of x and y the nodes will get a circular layout [10 Layouts], but other visual properties can be used as well. For instance z [11 Layouts3D], or color components.

5.4. Highlighting specific information

The requirements stated the need to highlight specific nodes and links by setting their visual properties and labels manually [18 Highlighting]. Right clicking on a link or node brings up its visual properties override form. Setting an option will override the values set by any mappings, layouts or global visualisation constants. Removing the visualisation override property will restore the value given by the mappings/layout/global visualisation constants.

A node can be moved by dragging and dropping it. This will cause the motion of the mouse cursor to be interpreted as the x, y coordinates of the clicked node. This meets [9 DragDropNode].

5.5. Observing the visualisation

As stated in the introduction of section 5 and in the requirements, section 2, it was necessary to be able to observe the visualisation. A requirement was the posibility to rotate the visualisation [12 Rotate3D]. An extra feature was added that allowed moving the vantage point as well.

The first subsection deals with moving the vantage point (i.e. the virtual camera), the second subsection deals with rotating the visualisation.

5.5.1. Moving the camera

The camera can be moved in several ways. It can be moved forward, backward, upward, downward and sideways, and it can be rotated along all axes. To see how the reader is referred to the user manual.

5.5.2. Rotating the visualisation

As stated in the requirements (section 2) and the introduction to this section, it was necessary to be able to rotate the application [12 Rotate3D]. The application has 3D rotation capabilities for all aforementioned layouts, along all axes (x, y and z). To see how the reader is referred to the manual.

5.6. Storing the visualisation
As stated in the introduction of section 5 and in the requirements, section 2, it was necessary to be able to store a visualisation in two ways. Saving an image, and recording a movie. Both ar treated in the next two subsections.

5.6.1. Save image

Images of the visualisation should be saveable in an everyday format [5 ImageSave]. An image can be saved under File-> Create Image. This will open a SaveAs dialog box allowing the user to select a folder and a filename to save the file containing the image under.

5.6.2. Record movie

It is possible to make movies of consecutive layouts [13 Movies]. Selecting File-> Record starts recording of a movie. Any layout changes while recording will be saved in the moview. The user can stop recording a movie by clicking File-> Stop Record.

5.7. Conclusion

Using mappings and a data loading feature, all data usage requirements and all visualisation requirements and all mapping requirments have been met using the data specifications of section 3 and the visualisation specifications of section 4. Thus [8 ShapeAdaptable], [3 VisualisationClear], [14 Comparison] and [10 Layouts] [11 Layouts3D] are met. Several parts of [7 ColorsAdaptable] are also met.

The global visualisation setting requirements have been met using the option to show a brain and offering options to set the background color; [16 ReferenceBrain] is met and the final aspects of [7 ColorsAdaptable] are also met.

Individual nodes and links can be highlighted meeting [18 Highlighting] and dragged dropped meeting [9 DragDropNode].

Visualisations can be viewed, by rotating them, or by moving the virtual camera, meeting [12 Rotate3D].

Visualisations can be saved using the save image and record image options, meeting the [5 ImageSave][13 Movies] requirements.

This leaves but [1 NodeCount512] and [6 AccessibleProject], which will be treated in the conclusion.

6. Design

Overview
As stated in the overview a design for ThoughtWeave was necessary that made the user interface of section 5 possible. This section presents that design.

ThoughtWeave was to visualise data supplied by the researchers as a 3D network diagram. To offer 3D visualisation a 3D engine is necessary. To interact with the 3D diagram a 3D gui is necessary. To visualise data on a network, a representation of this network is necessary.

To treat the visualisation of the network the 3D gui foundation it is built on must first be understood. To treat the 3D gui foundation, the 3D engine it is built on must be treated first.

That is why;

Section 6.1. treats	the 3D engine briefly,
Section 6.2. treats	the 3D gui,
Section 6.3. treats	the network,

Section 6.4. concludes this section.

6.1. 3D Engine

To understand 3D related topics in the following sections it was necessary to have an overview of the 3D engine.

Java3D was used as the 3D engine. To understand the other sections, some key concepts of the Java3D api must be treated.

Java3D (like most 3D engines) uses a tree structure called the scene graph to store 3D shapes and their spatial relationships.

Throughout ThoughtWeave;

- Leafs of this tree are atomic shapes or behaviors that change the tree while attached

- the branches either
    - group children to form some semantic union.
    - group children to change their spatial relationship to their parent collectively
    - behaviors that change the tree

The spatial structure of an atomic shape, i.e. which polygon is where, is described by a Geometry object. The appearance of an atomic shape, i.e. how the Geometry object should be painted/drawn is described an appearance object.

There are two types of atomic shapes used in ThoughtWeave;

1. Shape3D

A basic shape that has one geometry and an appearance that is applied to the geometry.

2. Morph

Morph is like Shape3D, in that it has an appearence, but unlike Shape3D it has multiple Geometries. Each Geometry contains the same polygons, but has different coordinates for the coordinates in those polygons. These geometries can be mixed by setting a weight array. The geometry displayed is a Geometry that takes the weighted average of all stored Geometries.

There are two types of branches used;

1. BranchGroups
BranchGroup just groups a set of children. It denotes a semantic unit. It is the only tree node that can be added or removed to/from a visible branch of the scene graph.

2. TransformGroups

A Transform is a spatial relation. It has a position, and orientation and a scale. A TransformGroup is a group that applies a transform to its children, giving them a spatial relation to the parent of the TransformGroup.

Finally there is Behavior

Behavior is a type of scene node that is allowed to change other scene nodes when the scene graph is visible.

6.2. The 3D Gui

As stated in the overview of the design it was necessary to understand the 3D gui before the design of the network representation could be understood. This section presents the 3D gui, capable of visualising 512 nodes [1 NodeCount512].

To understand the 3D gui, the 3D gui components, and how they interact with mouse events in the context of a typical gui environment must be described. The 3D gui component used by the design is Shape3DView. It receives its mouse events via Shape3DViewManager, through a process called picking.

Once this is understood, it is possible to treat how it visualises large networks of 512 nodes.

The first subsection treats the gui component, Shape3DView, the second subsection treats how the mouse events reach the gui components through Shape3DViewManager, and the third section shows how visualisation of large networks is possible. The final section is the conclusion.

Shape3DView

Shape3DView is a component that can represent something with visual properties, and it is a 3D gui components that generates mouse events for MouseListeners.

Visual properties
It has two groups of visual properties; Material and Space propeties.

- Material visual properties are red, green, blue and transparency,
- Space visual properties are x, y, z, shape, size/thickness. visible is a special property.

Materials and Spaces can be implemented in various ways and are added to the Shape3DView as plugin components.

Shape3DViewManager & Picking

Delivering the right mouse event to the right 3D gui component is dealt with by Shape3DViewManager. When a mouse event arises, it asks PickCanvas, the canvas onto which the 3D scene is drawn, the 3D object under the mouse coordinates. Shape3DViewManager retrieves the Shape3DView corresponding to that 3D object, if any, and passes on the mouse event to the matching handler function on the Shape3DView.

Visualisation of large networks

As stated in section 2, the application must be able to visualise 512 nodes.

the problem
These 512 nodes are connected by 130 816 links. And each relationship file will contain this much links.

A naive approach where all 130 816 links are shown is both impractical and intractable. It took huge amounts of both memory and time. A MacBookPro3,1 took more than 15 minutes to load, and the result was an unclear tangle of links.

The research group and the development team concluded that showing large numbers of links would is not practical.

Two things cost a lot of time;

1. loading the file with the shape of a link 130 000 times

2. adding even so much as a TransformGroup for 130 000 links

the solution
The chosen solution was to not load link shapes unless a researcher explicitely set a link to visible.

Whenever a link is not loaded yet and is assigned a value for some visual property, this value is stored for that visual property in either its Material plugin, or in its Space plugin (whichever is responsible for that visual property).

When the link is set to visible, it first loads its shape. Its LinkView object makes a call on init in Shape3DView, passing the file containing the 3D model of a link as argument. Shape3DView calls Shape3DFactory. Shape3DFactory checks whether the filename has yet been loaded. If not, the file is loaded and put in a map with as its key the filename and as its object, the Shape3D in the file. It returns the file under stored in the map under filename, returning it immediately if it had already been loaded.

Then Shape3DView assembles its scene graph branch and adds the Shape3D to it.

Finally it adds itself to its parent.

Conclusion

A 3D gui is presented using Shape3DView as a 3D user interface component, with Shape3DViewManager for supplying its mouse events. By only loading the graphics of a component if it is visible, the 3D gui can handle the vast amount of data generated by analysis of 512 nodes [1 NodeCount512].

6.3. Network

The network visualisation is built up out of several components, and has gui components to configure it.

To understand the network the components need to be understood. These components are the nodes and the links. To understand the nodes and the links it is necessary to understand their underlying design pattern, model view control, first.

Once the network components are understood it is possible to understand the gui components that configure it.

The first subsection deals with model view control.
The second subsection can the deal with nodes and links
The third subsection treats the network as a whole and the gui components to configure it

6.3.1. Model View Control

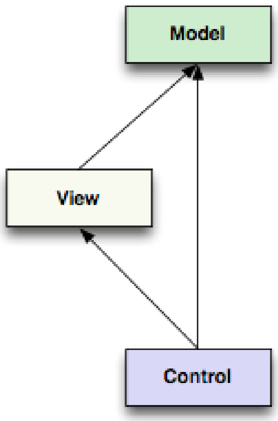
Many descriptions of the model view controller descriptions exist, that are quite similar yet still have slight differences. For the purpose of this document an unambiguous description of the MVC is required.

This section defines the MVC pattern for the purpose of this document.

Model
The model is the gui representation of the logic/data. It does not event listen to view or control, nor does it make function calls on them.

View
The view is a (posibly composite) graphical user interface component. It has no semantics about the user’s task, it has no knowledge of any other component, other than the model. It can event listen to the model, and it can make calls on the model.

Control
The control sets up the view and the model. It calls the model and the view. It gives meaning to user interface commands of the view, and changes in the model, by event listening them. This can lead to function calls on either model or view. Model or view never make calls on control, nor do they listen to control. They are unaware and independent of control.



6.3.2. Nodes/Links

As stated it was necessary to understand the nodes and links. Now that the model view control pattern is explained in the previous section they can be treated.

Nodes and links both represent logic/data, and both are visualised in 3D. This means they need some underlying model component to bind them to any logic/data, and they need some 3D graphical user interface view component to allow the user to interact with it.

The 3D GUI view component needs to make use of the model component. The node and link control components depend on both the view component and the model component. So first these view and model components must be treated.

Once the model component is treated, the view component can be treated. Finally when both have been treated, the node and link control components can be addressed.

Model

A Model contains a set of visual property values and a set of data properties. A model also has a set of values that contain all overridden visual property values.

Shape3DView

Shape3DViews contain either a Shape or a Morph. A Shape3DView offers mouse event handling functions. It is registered with Shape3DViewManager which handles clicking on the 3D scene and dispatches mouse events to the correct Shape3DView.

Node

There is no Node class. Nodes are composed of 3 classes; NodeModel, NodeView and NodeControl.

NodeModel

NodeModel derives from Model. The only overriden functionality is it's constructor, which initialises all the available view properties for nodes.

NodeView
NodeView derives from Shape3DView. It can update its Shape3DView based properties based on NodeModel.

NodeControl

NodeControl constructs the node by creating a NodeModel, and a NodeView. It handles mouse events from NodeView.

It creates a NodeDragBehavior when the left mouse button is down, and destroying it when the drag drop operation is over

It creates a PropertyOverrideSetEditor on a right click which allows overriding global visual properties set by mappings in the NetworkMappingManagerView, a window which allows setting mappings for the entire network.

Link

Links do not have a distinct class. Links are composed of 3 classes; LinkModel, LinkView and LinkControl, similar to nodes.

LinkModel

LinkModel derives from Model. The only overriden functionality is it's constructor, which initialises all the available view properties for links.

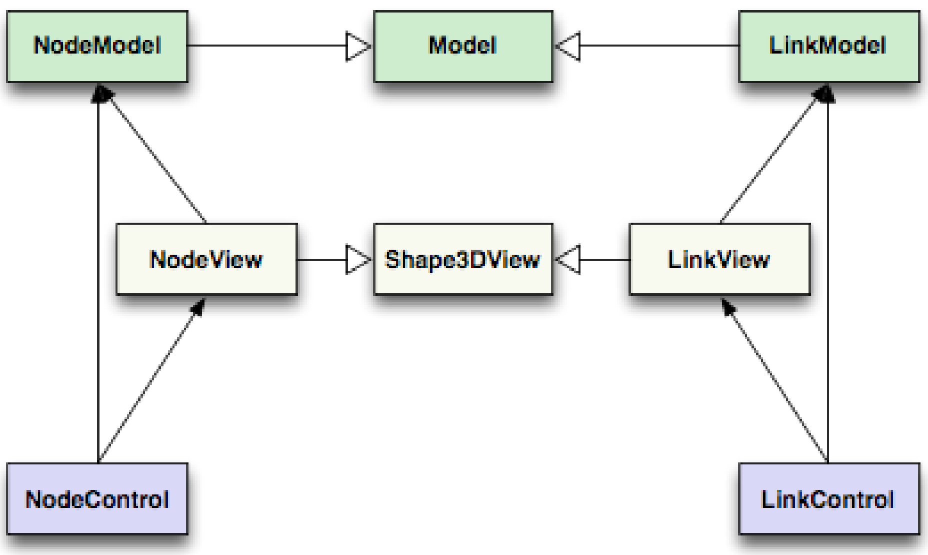
LinkView

LinkView derives from Shape3DView. It can update its Shape3DView based visual properties based on its LinkModel. It also assures that its Shape3DView is positioned between the two nodes it connects.

LinkControl

LinkControl constructs the node by creating a LinkModel, and a LinkView. It handles mouse events from LinkView.

It creates a PropertyOverrideSetEditor on a right click which allows overriding global visual properties set by mappings in the NetworkMappingManagerView, a window which allows setting mappings for the entire network.



6.2.3. Network

NetworkControl manages the components that together represent the whole network. It can be configured using two user interface components.

Before NetworkControl can be understood the components it uses must be understood. The previous section dealt with nodes and links, but NetworkControl also uses ModelManagers. ModelManagers apply mappings to Models, which can be both NodeModels and LinkModels. Before these ModelManagers can be understood, mappings must be understood.

NetworkControl requires a user interface component to load data, and a user interface component to configure mappings. Once NetworkControl is understood, these components can be treated.

The first subsection treats Mappings
The next subsection treats ModelManagers
The third subsection treats NetworkControl and its user interface components.

Mapping

A Mapping is applied to a Model. It can access the Model's data properties and use them to define the Model's visual properties. No real Mapping exists. Just an interface describing it, called NetworkMappingInterface.

NetworkMappingInterface contains a function apply(ModelInterface model) that applies the mapping to the passed Model. Thus it can be applied to a set of Models.

ModelManager

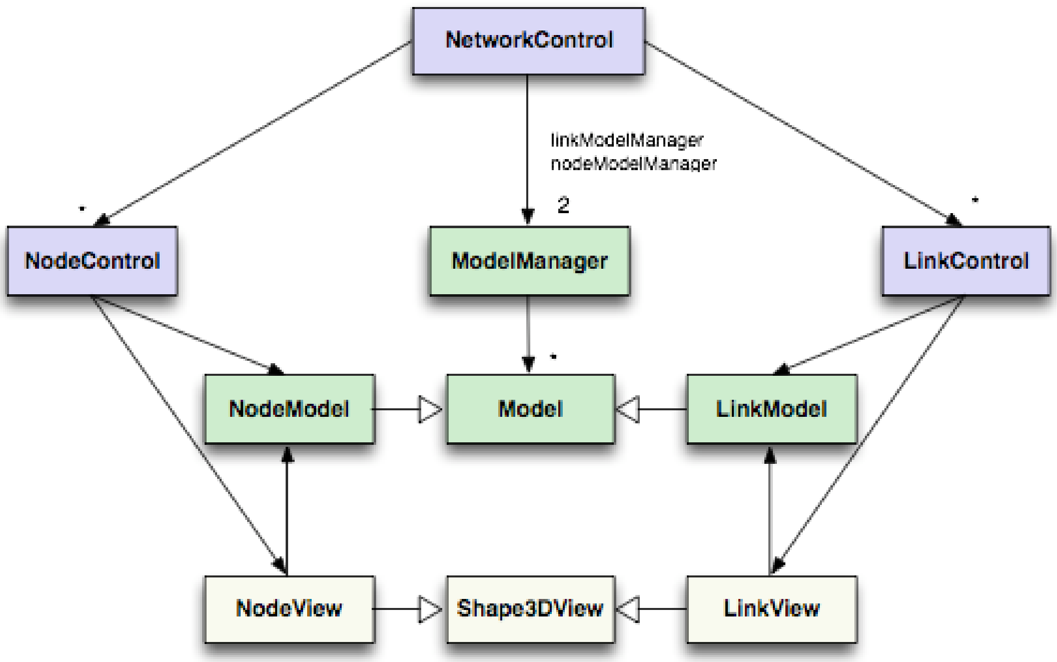
ModelManager contains and manages a set of Models. It can apply a NetworkMapping to all the nodes it contains. It can also compute various statistics, such as the highest value of some property in all the Models it contains.

NetworkControl

NetworkControl manages all nodes and links. It initialises a number of NodeControls and LinkControls and then stores their Models in two ModelManagers. One manages the Models of the NodeControls, the other manages the Models of the LinkControls.

It assures the NodeViews and LinkViews are kept up to date by listening to changes in the ModelManagers.

Data on the network is loaded using NetworkDataControl. Mappings are added using NetworkMappingManagerView. They are treated respectively in the next subsections.



NetworkDataControl

NetworkDataControl loads data into a NetworkControl.

- It presents a FileChooser to the user.
- The user selects a directory.
- For each file in the directory
    - Files with .vd extension are used to construct a VisualisationNodeDataSet
    - Files with .nd extension are used to construct a BrainWaveNodeDataSet
    - Files with .rd extension are used to construct a LinkDataSetFile
- all loaded DataSets are loaded using NetworkControl.load( )
    - NetworkControl.load delegates the loading to the correct ModelManager
        - ModelManager loads the correct data into the correct Model

For file specifications see the manual.

NetworkMappingManagerView

NetworkMappingManagerView manages all the NetworkMappings, as a gui on top of a ModelManager. It maintains a list of gui components for interfacing with the different mappings configured by the users. It is also used to add new mappings, and passes these mappings the ModelManager they are operating on.

6.4. Conclusion

As stated in the overview of the design section first presented the 3D Engine concepts necessary for understanding the 3D gui. Then the 3D gui, capable of visualising 512 nodes [1 NodeCount512], was presented necessary for understanding the network representation. Finally a network representation was presented that allowed for the implementation of the user interface described in section 5.

This section itself presents the documentation required for [6 AccessibleProject].

7. Conclusion

The overview stated that this section could conclude the requirements in section 2 were met.

Section 5 showed how the data formats from section 3 could be used to visualise using the visualisation framework of section 4.

Section 6 presented a design [6 AccessibleProject] for the gui, able to visualise up to 512 nodes well [1 NodeCount512].

Finally all source is open source, programmed clearly and handed in [6 AccessibleProject].

Thus all requirements from section 2 are met and ThoughtWeave accomplishes its goal.

Appendix A

Technische Requirements

[1] Het programma moet netwerken tot 512 nodes goed visueel kunnen weergeven.
 Meer nodes is wenselijk, maar in dit stadium niet strikt noodzakelijk.

[2] Het invoerformaat van de netwerken is ASCII-tekst zoals gebruikt op de afdeling klinische neurofysiologie van het VUMC (bijlage 1).

[3] Zowel netwerken met gewogen connecties als niet-gewogen netwerken moeten visueel aantrekkelijk worden weergegeven. Het verschil moet ook duidelijk zijn.

[4] De afzonderlijke nodes moeten gelabeld kunnen worden door middel van een label-file (bijlage 2). Labels moeten aan- en uitgezet kunnen worden.

[5] De afbeelding moet op te slaan zijn in een alledaags formaat (JPG, BMP, of PNG).

[6] Alle sourcecode zal open-source zijn, opgeleverd worden, overzichtelijk geprogrammeerd zijn, gedocumenteerd zijn en eenvoudig te hercompileren zijn.

Requirements voor een vervolg project

[7] Visualisatie van kleuren van, achtergrond, nodes en edges moeten gemakkelijk aan te passen zijn.

[8] De vorm van de nodes en edges moet gemakkelijk aan te passen zijn.

[9] De applicatie heeft een drag-en-drop interface waarbinnen de nodes gesleept kunnen worden.

[10] De applicatie beschikt over verschillende netwerk-layouts voor visualisatie: circulair, handmatig, coordinate-driven, anatomisch.

[11] De applicatie beschikt over 3D-visualisatie voor alle eerdergenoemde layouts.

[12] De applicatie beschikt over 3D-rotatiemogelijkheden voor alle eerdergenoemde layouts.

[13] Het is mogelijk een filmpje te maken van opeenvolgende weergaven. Het filmpje kan worden opgeslagen in een gebruikelijk formaat.