Cognitive State Representation and Visualization of Human Brain

Software Design Document

v1.1

Simple Labs

Atakan KAYA, 1746114

Barış NASIR, 1746288

Özlem Ceren ŞAHİN, 1746668

Bahattin TOZYILMAZ, 1746395

December 29, 2013

PREFACE

This document contains the software design details for the “Cognitive State Representation and Visualization of Human Brain” software. The document is prepared according to the “IEEE Standard for Information Technology – Systems Design – Software Design Descriptions – IEEE STD 1016 – 2009”. This Software Design Documentation provides a complete description of all the system design and views of the “Cognitive State Representation and Visualization of Human Brain”.

Revision History

Version	Date	Sections Changed, Added, Deleted	Type of the Change	Brief description
1.0	01.12.2013	-	A	Initial version
1.1	29.12.2013	5.3.1	D	Language change
1.1	29.12.2013	5.3.*	M	Language change
1.1	29.12.2013	6	A	-

*A: Added, M: Modified, D: Deleted

CONTENTS

1 OVERVIEW

1.1 SCOPE

The software to be produced is “Cognitive State Representation and Visualization of Human Brain”. In this project it is aimed to connect different simulation, computer graphics and image processing technologies. At the end of the project, a software enhancing the graph structure (by simplifying it with down sampling and quantization) will be implemented in Unity3D game engine. To visualize our work and make the software more attractive, the project is designed to include some extra features like zoom in/out, rotation, showing brain parts seperately.

Initially, the project was planned to be fully implemented with OpenGL. However, after meeting with the customer and our supervisor, we decided to use a game engine to implement visual part of the project. After doing some researches we decided to use Unity3D game engine.

Unity3D game engine is selected for this project because it includes occlusion culling feature that renders only what can be seen, level of detail support and build size stripping. Also, it supports DirectX 11, shader model 5.0 and OpenGL.

1.2 PURPOSE

This document describes how “Cognitive State Representation and Visualization of Human Brain” will be structured to satisfy the requirements identified in the Software Requirements Specification document prepared by Simple Labs. Team in their senior software project. It includes modifications over initial design document.

Requirements Specification document determines software, hardware, functional and nonfunctional requirements decided to be satisfied and gives a general idea how the system will work. This document covers the details and different aspects of the project in a comprehensive way and conceptualizes the overall product that will be formed accurately.

In the design process, it is intended to design an effective and modular product that will satisfy the needs and constraints of the project. It is also aimed to explain the functional, structural and behavioral features of the system by using specific types of UML diagrams such as class, sequence, state diagrams. In order to support these diagrams, graphical user interface prototypes are also provided in the document.

1.3 INTENDED AUDIENCE

This document is intended for both the stakeholders and the developers who build the system.

1.4 REFERENCES

1. IEEE. IEEE Std 1016-2009 IEEE Standard for Information Technology – System Design – Software Design Descriptions. IEEE Computer Society, 2009.

2. StarUML 5.0 User Guide. http://staruml.sourceforge.net/docs/user-guide(en)/toc.html

2 DEFINITIONS

3D	3 Dimensional
API	Application Programming Interface
FC	Functional Connectivity
fMRI	functional Magnetic Resonance Imaging
OpenGL	Open Graphics Library
GLU	OpenGL Utility Library
GLUT	OpenGL Utility Toolkit
GPU	Graphichs Processing Unit
IEEE	Institute of Electrical and Electronics Engineers
METU	Middle East Technical University
MVPA	MultiVoxel Pattern Analysis
POD	Plain Old Data
RAM	Random Access Memory
RTTI	Run-Time Type Information
SDD	Software Design Description
SRS	Software Requirements Specification
UML	Unified Modeling Language

3 CONCEPTUAL MODEL FOR SOFTWARE DESIGN DESCRIPTIONS

3.1 SOFTWARE DESIGN IN CONTEXT

The aim of this project is visualizing the fMRI data on a 3D graph to increase the understandability of the complex data. The fMRI data includes the brain response of a human in response to some particular circumstances as showing the picture of a red apple.

The main role of the final 3D graph will be visualizing voxels and edges which show the related voxels. Besides graph may include a brain image as background and five main lobes of the brain, which are frontal, parietal, occipital, limbic, temporal lobes, in different colors.

Since the fMRI data is very large and complex, time and space will be main constraints.

The target audience of this project is mostly academicians and medical institutes. Cognitive state representaion and visualization of human brain is fundamentally important in neuroanatomy, neurodevelopment, cognitive neuroscience and neuropsychology.

This project will be implemented in Unity3D Game Engine with using its OpenGL libraries. C # will be used as the programming language.

3.2 SOFTWARE DESIGN DESCRIPTIONS WITHIN THE LIFE CYCLE

3.2.1 INFLUENCES ON SDD PREPARATION

The key software life cycle product that drives a software design is typically the software requirements specification.

The requirements in the SRS like product perspective,interface requirements, functional and non-functional requirements and also the demands of the stakeholders specify the design of the project.

3.2.2 INFLUENCES ON SOFTWARE LIFE CYCLE PRODUCTS

As said before, the key software life cycle product that drives a software design is typically the software requirements specification. However during the preparation of this Software Design Description document or the implementation stage of the project, some requirements may change and this results in the change of SRS and SDD.

Besides, SDD influences test plans and test documentation of the Cognitive State Representation and Visualization of Human Brain project.

3.2.3 DESIGN VERIFICATION AND DESIGN ROLE IN VALIDATION

Verification and validation will be tested after preparation of the test cases. All system parts will be tested against these cases. It will be checked for whether the requirements are fulfilled or not.

4 DESIGN DESCRIPTION INFORMATION CONTENT

4.1 INTRODUCTION

This is an SDD document for Cognitive State Representation and Visalization of Human Brain project. Through the document, detailed design cases are explained and depicted using UML diagrams.

4.2 SDD IDENTIFICATION

This Software Design Document is written on the request of Ceng 491 instructors to be able to guide the development process of Cognitive State Representation and Visalization of Human Brain. It is written by Simple Labs team collaboratively. Below table presents the authorships of sections.

Section	**Author**
1.*	Barış Nasır
2.*	Barış Nasır
3.*	Özlem Ceren Şahin
4.*	Atakan Kaya
5.1	Bahattin Tozyılmaz
5.2	Atakan Kaya
5.3	Bahattin Tozyılmaz, Barış Nasır, Özlem Ceren Şahin, Atakan Kaya
5.4	Özlem Ceren Şahin
5.5	Barış Nasır
5.6	Atakan Kaya
5.7	Bahattin Tozyılmaz
5.8	Bahattin Tozyılmaz, Barış Nasır

The date of issue of the initial version of this document is December 29, 2013.

4.3 DESIGN STAKEHOLDERS AND THEIR CONCERNS

The design stakeholders for our project are Prof. Dr. Fatoş Yarman Vural and her research group. Our project is shaped by their research and requirements.

The major concerns of design stakeholders can be listed as:

- They want to observe brain using this tool.
- They want the edge size to be reduced and a smooth 3D image to be rendered.

- They want the project to be completed in time.
- They want to be kept informed about the process.

4.4 DESIGN VIEWS

Our project has emerged from a need of an efficient, simple and smooth drawing of brain data. To achieve this, filtering techniques are required. And since the filtering techniques are under research and development phase by the Image Processing Laboratory of METU, an object-oriented and an easily extendable approach is preferred. To point this, a contextual view that determines the services required, a logical view that draws the relations between basic entities, a dependency view and a patterns use view that defines the relation between subsystems, an interface view that gives insight about how the end product will be, an interaction view that depicts the flow of information and an algorithm view that focuses on the algorithms used is required.

4.5 DESIGN VIEWPOINTS

In this document, the contextual viewpoint focuses on the services by use case diagrams to define the usage of features by the actors. Then, a logical viewpoint defines the classes and the relationships between them. In the dependency viewpoint, the relationships of interconnections and access among entities are specified. Later, patterns use viewpoint depicts how subsystems of the project are connected. In the interface viewpoint the relations of the UI modules and a mockup visualization is provided. Then, interaction viewpoint explains the interactions between several objects of the project. Finally, algorithm viewpoint defines the required algorithms throughout the project.

4.6 DESIGN RATIONALE

In this document, design features are chosen to improve reusability and provide extensibility. This is vital since the related projects at Image Processing Laboratory are under development, the reqirements for them can change.

4.7 DESIGN LANGUAGES

Throughout the document, UML use case diagrams, UML component diagrams, UML class diagrams, UML sequence diagrams and ER diagrams are used.

5 DESIGN VIEWPOINTS

5.1 INTRODUCTION

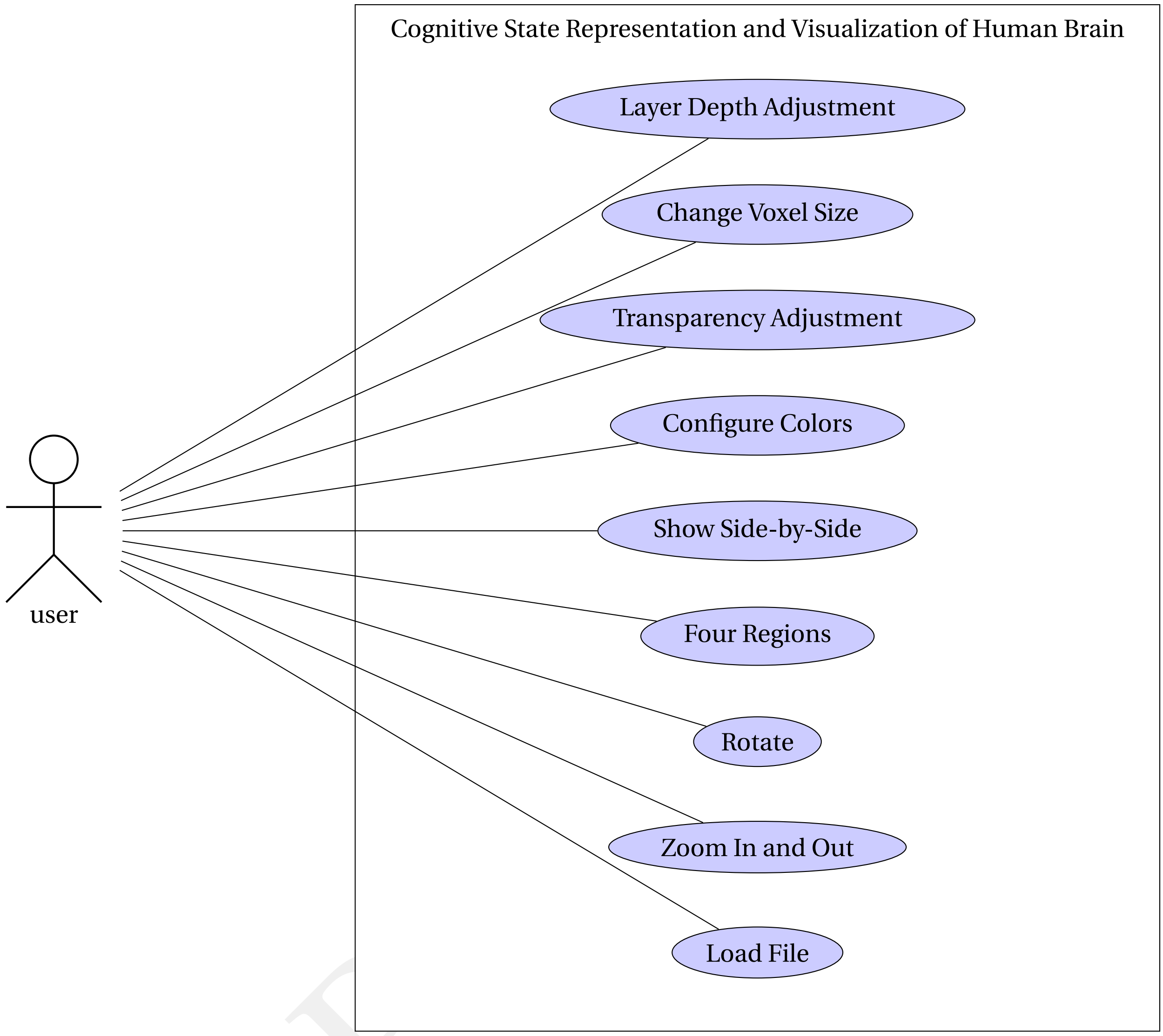
In this part, seven main design viewpoints will be explained.

- Context Viewpoint
- Logical Viewpoint
- Dependency Viewpoint
- Patterns Use Viewpoint
- Interface Viewpoint
- Interaction Viewpoint
- Algorithm Viewpoint

During the explanation of these viewpoints, UML diagrams will be used to increase understandability.

5.2 CONTEXT VIEWPOINT

This section of Software Design Description focuses on the services provided. The context is defined by reference to actors.



5.2.1 DESIGN CONCERNS

The use cases provided in this section depicts the offered services for the actor.

5.2.2 DESIGN ELEMENTS

Actors

- User: The user that uses the program

Services

- Load File: The user loads the data taken from fMRI machine.
- Zoom In and Out: The user can zoom in and zoom out to 3D image of the brain.
- Rotation: The user can rotate the image 360 degree around it's center to any direction.
- Four Regions: The user can do selection between four brain lobes. Only the selected lobes will be visible.
- Show Side-by-Side: With this feature, the user can see the brain from different sides in one window.
- Configure Colors: Changes color distribution for color blinds.
- Transparency Adjustment: Changes transparentcy of the voxels so deeper layers can be seen.
- Change Voxel Size: Changes unit size of voxels in case size is miscalulated.
- Layer Depth Adjustment: Allows user to view inner layers of brain.

5.2.3 EXAMPLE LANGUAGES

In this section, UML Use Case Diagrams are used.

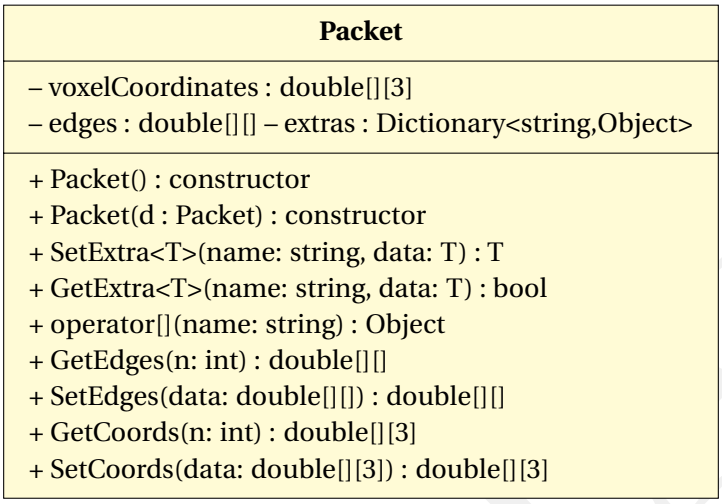
5.3 LOGICAL VIEWPOINT

This section of the Software Design Document focuses on logical foundations of the project. The logical foundations of the project are the classes and relations between them. In this section, those classes, their methods and interactions will be explained in detail.

Extensibility is a must for the project. Since this project will be used in a highly active research area, it is essential that novel ideas be implemented easily. Project team aims to achieve this with a highly algorithm and data agnostic approach.

5.3.1 PACKET CLASS

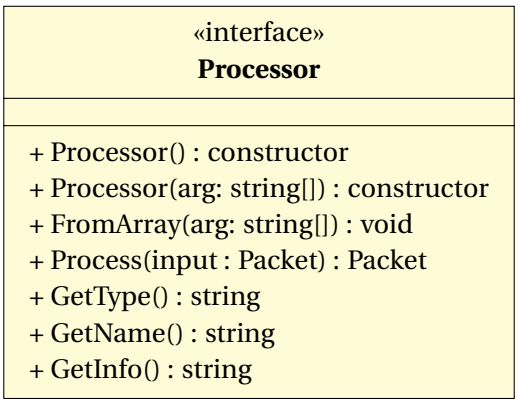
This class is used as an immediate data format between two Processors. Packet class encapsulates all data needed by Processors: voxel coordinates, edge values, etc... It also offers a way to pass named extra data between Processors.



Name	Returns	Visibility	Description
voxelCoordinates	double[][3]	Private	This member holds coordinates of voxels
edges	double[][]	Private	This member holds edge matrix
extras	Dictionary<string, Object>	Private	Named collection of extra datas
Packet()	«constructor»	Public	Dummy constructor
Packet(d: Packet)	«constructor»	Public	Copy constructor
SetExtra<T>(name: string, data: T)	T	Public	Sets an extra with the given name and content
GetExtra<T>(name: string, data: T)	bool	Public	Gets the extra with the given name or returns false
operator[](name: string)	Object	Public	Shorthand for getting and setting extras
GetEdges(n: int)	double[][]	Public	Gets the edge matrix
SetEdges(data: double[][])	double[][]	Public	Sets the edge matrix
GetCoords(n: int)	double[][3]	Public	Gets voxel coordinates
SetCoords(data: double[][3])	double[][3]	Public	Sets voxel coordinates

5.3.2 PROCESSOR INTERFACE

This interface defines outlines of Processors and how Processors should be implemented.



Name	Returns	Visibility	Description
Processor()	«constructor»	Public	Dummy constructor
Processor(arg: string[])	«constructor»	Public	This should be the constructor called from other places
FromArray(arg: string[])	void	Public	Sets properties of the Processor
Process(input: Packet)	Packet	Public	The real job is done here
GetType()	string	Public	Returns "sink", "process" or "input"
GetName()	string	Public	Returns internal name for the Processor
GetInfo()	string	Public	Returns a simple documentation

5.3.3 PIPELINE CLASS

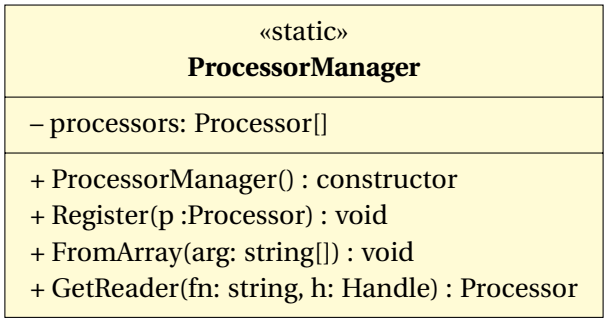
This class is responsible for chaining Processor operations. A pipeline is an object that the user can save to or load from it a file. Thus, it also enables the user to create his/her own presets. When a Processor is added to the Pipeline, Pipeline object checks whether it is the first Processor to be added, and if it is, is it an input type Processor.

Pipeline
– processors: Processor[]
+ Pipeline() : constructor + Pipeline(rhs: Pipeline) : constructor + FromArray(arg: string[]) : void + ToArray() : string[] + AddProcessor(pr: Processor) : bool + Run() : Packet

Name	Returns	Visibility	Description
processors	Processor[]	Private	This is the list of Processors this Pipeline consists of
Pipeline()	«constructor»	Public	Dummy constructor
Pipeline(rhs: Pipeline)	«constructor»	Public	Copy constructor
FromArray(arg: string[])	void	Public	Constructs "processors" with given information
ToArray()	string[]	Public	Saves "processors" list so that it can be loaded with FromArray
AddProcessor(pr: Processor)	bool	Public	Adds a Processor to the list. First Processor on the list should be input type
Run()	Packet	Public	Runs generated Processor sequence, returns output of last Processor

5.3.4 PROCESSORMANAGER CLASS

This class is responsible for managing Processor selection and generation. This is a static class and it's members are all static. Each Processor must register itself with the ProcessorManager. C++ doesn't allow static constructors, which would be used when registering. This is a problem the team is working on.

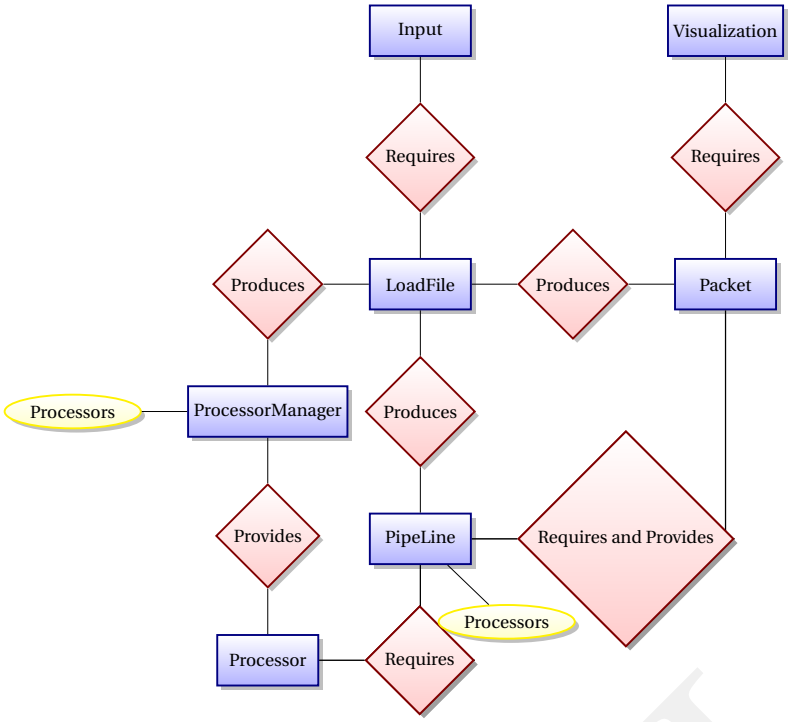


Name	Returns	Visibility	Description
processors	Processor[]	Private	This is the list of registered Processors
ProcessManager()	«constructor»	Public	Static constructor
Register(p: Processor)	void	Public	Registers p with ProcessManager
FromArray(arg: string[])	void	Public	Constructs a Processor with given information
GetReader(fn: string, h: Handle)	Processor	Public	Finds the input Processor that can read given source

5.4 DEPENDENCY VIEWPOINT

In this section of the design document, the relationships of interconnections and access among entities are specified. These relationships include information sharing, order of execution and parameterization of interfaces.

ER diagram below shows the entities and their relationships. They are also explained in the subsections of this section.



Dependency viewpoint provides an overall picture of the system entities and their relationships in order to assess the impact of requirements and design changes. This section helps maintainers in two ways: System failures or resource bottlenecks can be resolved by identifying the entities which causes them and development plan can be prepared by identifying which entities are needed by other entities and which should be developed first.

There are seven design entities which are Input, Box, Packet, Processor, ProcessorManager, LoadFile and Pipeline.

There are four design relationships, namely uses,requires, provides, produces.

- requires: In the main loadFile requires input from the user, Pipeline requires Packet of preprocessed input data and one or more Processors to process Packet.
- provides: Pipeline provides Packet at the end of its process and ProcessorManager generates processors and provides them for further use.
- produces: loadFile produces Packet, ProcessorManager and Pipeline due to the input, which includes user choices which effects attributes of these entities.

Short descriptions of attributes are given below but detailed information about attributes can be found at section 5.3 Logical Viewpoint.

- Packet
  - voxelCoordinates: Coordinates of the brain voxels.

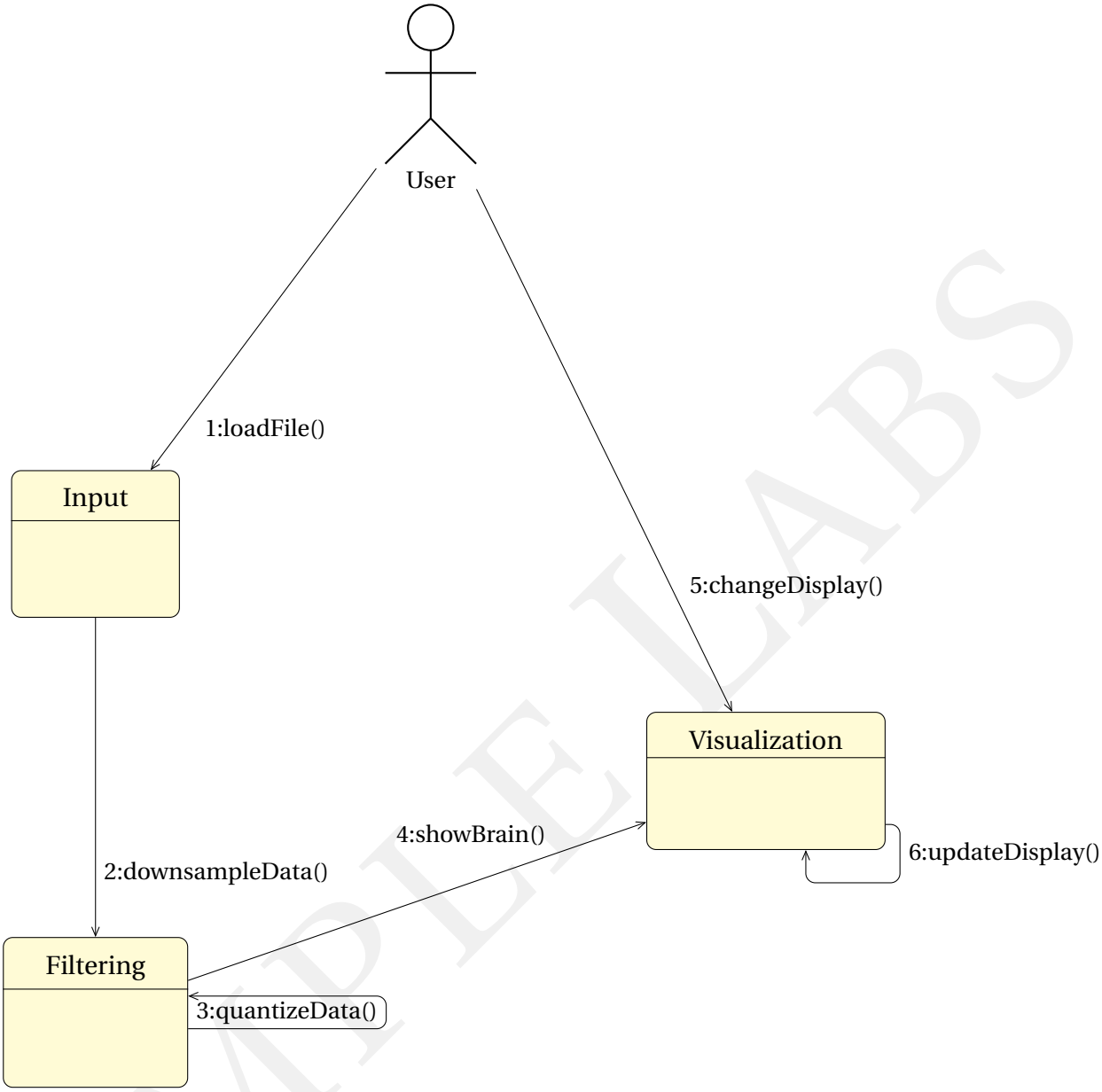
  - edges : Edge matrix.

- ProcessorManager
  - processors : list of processors.
- PipeLine
  - processors : list of processors.

5.5 PATTERNS USE VIEWPOINT

In this part of the design document, how subsystem will be connected and in which order their functions will be called is explained. The order of the function can be seen in Collaboration Diagram. First of all, duties of functions in the diagram will be explained.

Function Name	**Function Duty**
loadFile()	This function is called by user to load the data taken by fMRI machine. This function starts the Input subsystem.
downsampleData()	This function is called by Input subsytem to connect the Filtering subsystem to downsample the given data to minify it.
quantizeData()	This function is called by the Filtering subsystem after execution of downsampleData() to reduce the file size and ease the handling of data. This function does quantization on input.
showBrain()	This function is called by Filtering subsytem to connect the Visualization subsystem after execution of quantizeData() function. This function shows the processed data as a 3D image. This function uses built-in Unity3D functions and OpenGL libraries and function implemented by the Simple Labs.
changeDisplay()	This function is called if the user adjusts transparency, colors, rotation, zoom or layer depth or changes the display by clicking on "Show Side-by-Side" or "Four Regions" button. This function cannot be called before the showBrain() function.



Order of Function

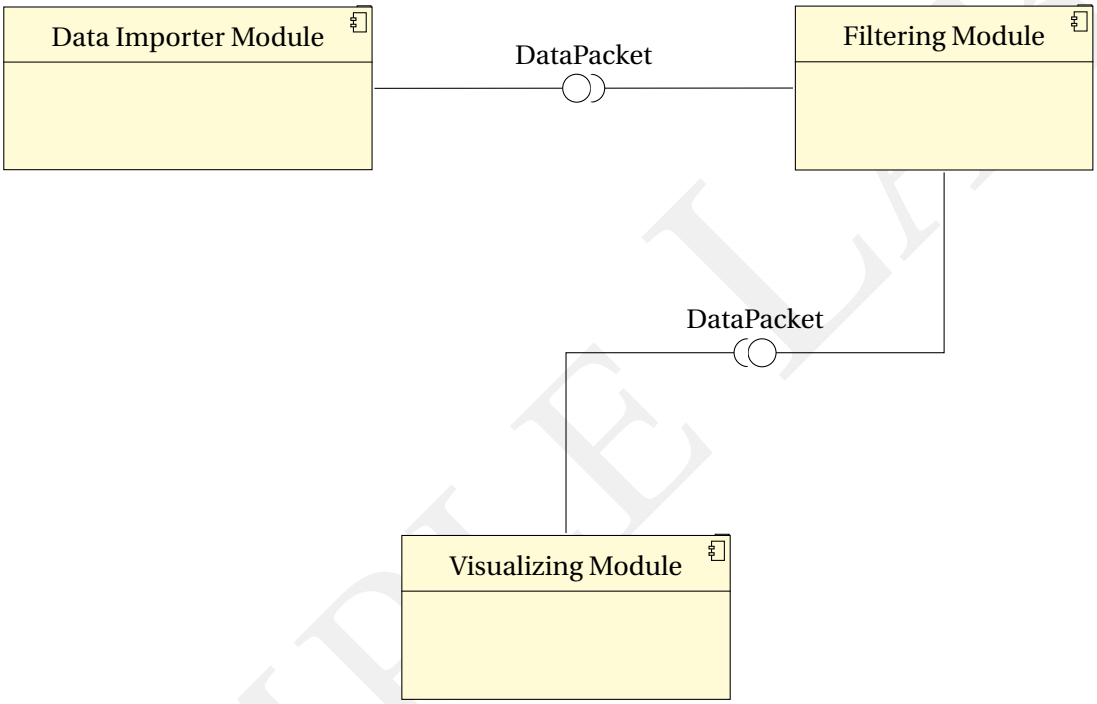
The function is called with respect to numbers stated in diagram.

Firstly, first function is called and input data is loaded to the system. Secondly, second and third functions are called and system is started. After that, fourth function is called to show the 3D image created with the processed data. Lastly, fifth and sixth functions are called repeatedly when there is a user interaction until the program is closed.

5.6 INTERFACE VIEWPOINT

Interface viewpoint can be decomposed into three major components.

First, the data importer module is responsible for importing voxel position values, voxel intensity values and edges (arclengths). The file format will be MATLAB file format; however, this module can be extended with ability to handle other file formats, namely CSV, raw text, etc. Second, the filtering module is responsible for preparing data to be shown on the screen smoothly. This includes down-sampling, quantization, edge boundling e.g. techniques. The output of this component will be ready-to-draw voxel position parameters, voxel intensity values and edges (archlengths). Lastly, visualization component will use 3D rendering engine and draw the image to the screen. This is visualized using UML component diagram below.



Planned user interface is depicted at Figures 5.1, 5.2, 5.3, 5.4 and 5.5. There will be only one main screen. Left pane is the image plane and user will be able to interact with this plane to rotate the 3D image. On the right pane, controls for rotating and zooming will be placed. Layer depth, transparency and voxel size will be customizable through a slider. Filter group lists the filters available (namely down-sampling, quantization, edge-bundling e.g.). Note that as the research continues new filters will be added. Filter options can be set up through edit menu -> Filters. Region can be selected using a dropdown menu. Available options will be whole brain, frontal lobe, parietal lobe, occipital lobe, temporal lobe and limbic lobe. Note that however, these region options are tentative. Lastly, a suitible view for colorblind people will be generated if the related option is enabled.

Our intention for the behaviour of these right pane options is as follows. As any option change occurs, the related action will be triggered instantaneously. However, as the

processing might take some time, an popup box indicating work done will be shown.

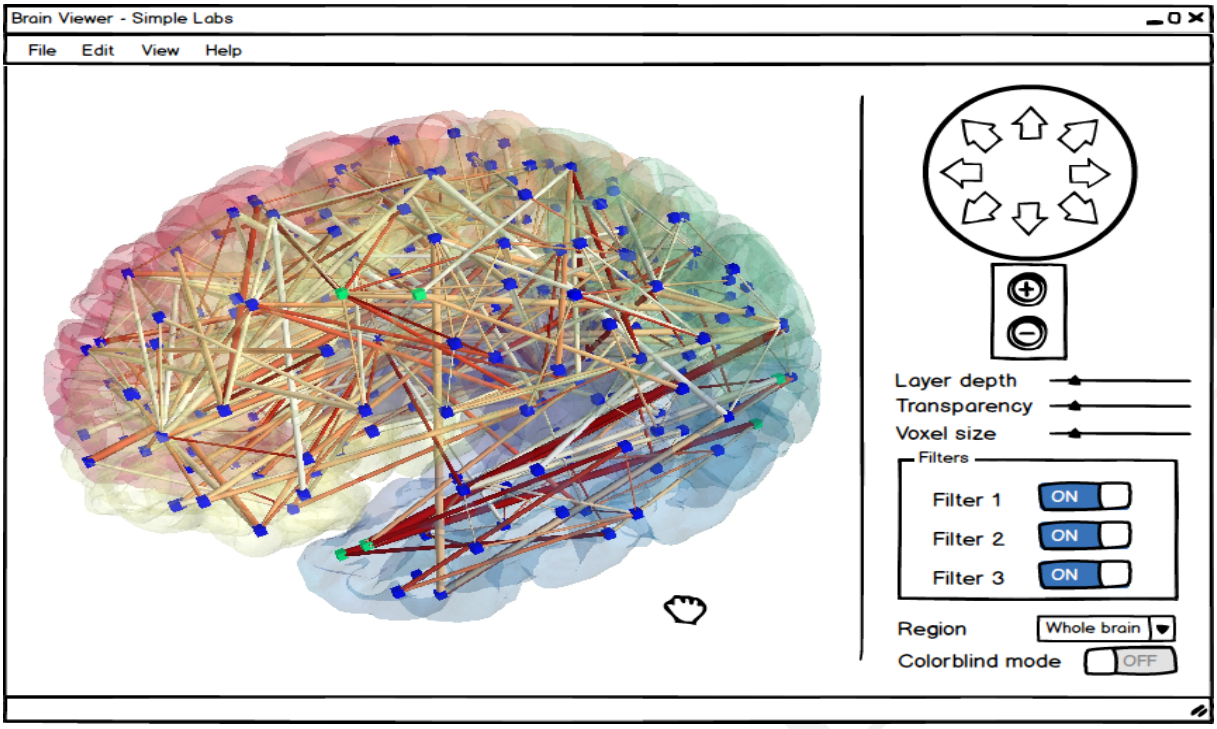


Figure 5.1: Main window

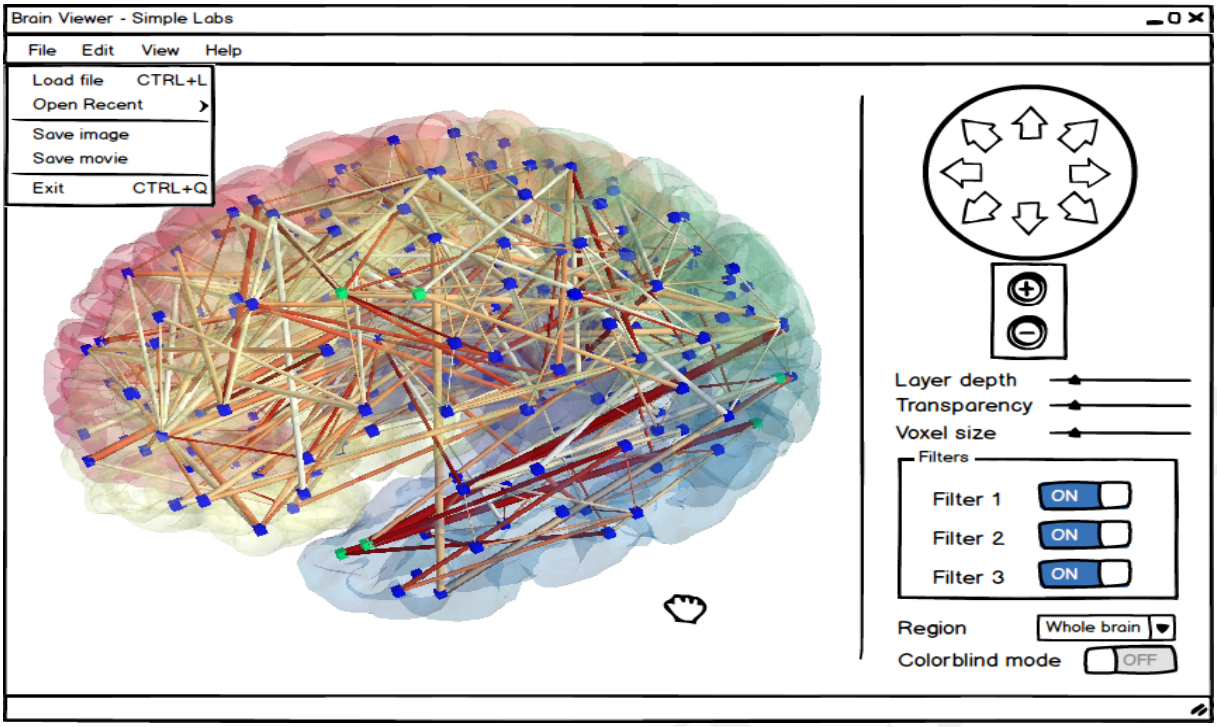


Figure 5.2: File menu

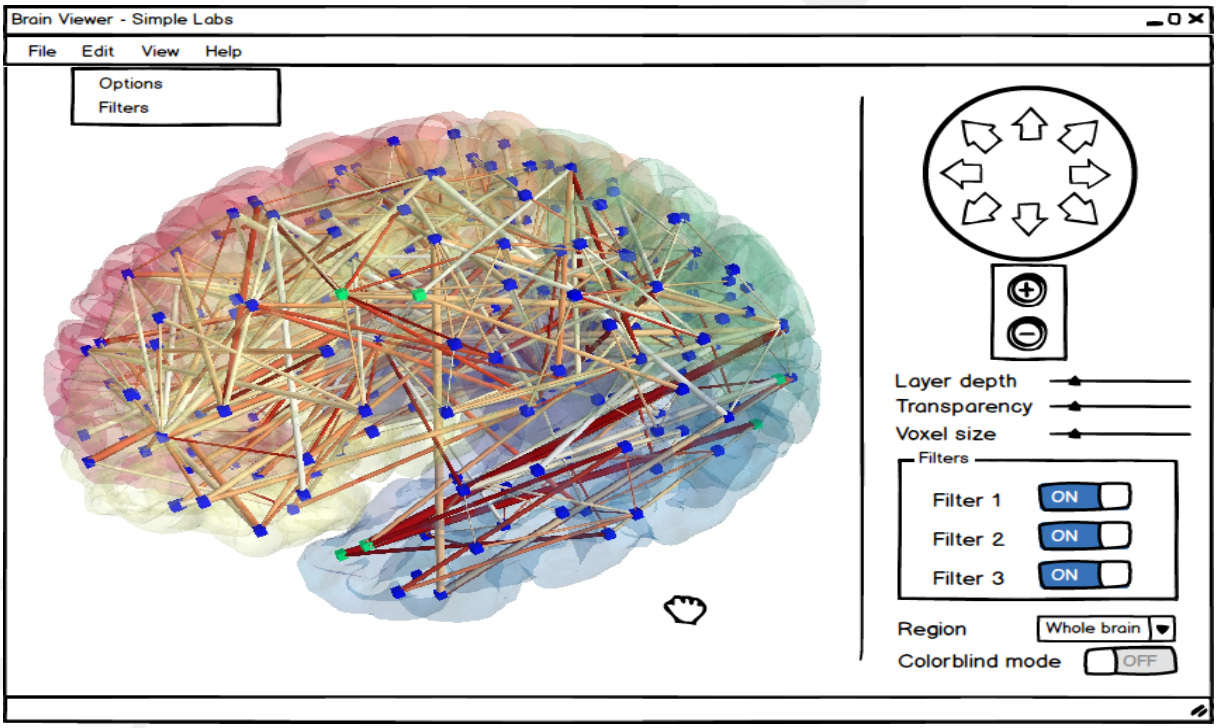


Figure 5.3: Edit menu

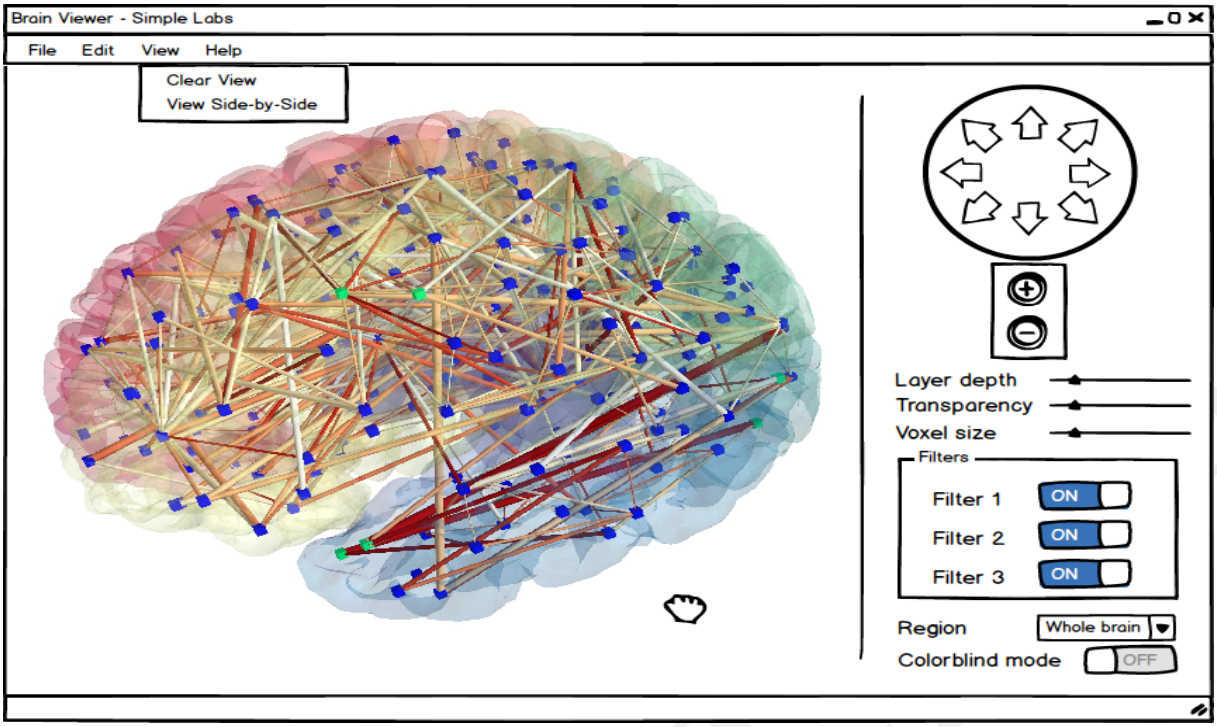


Figure 5.4: View menu

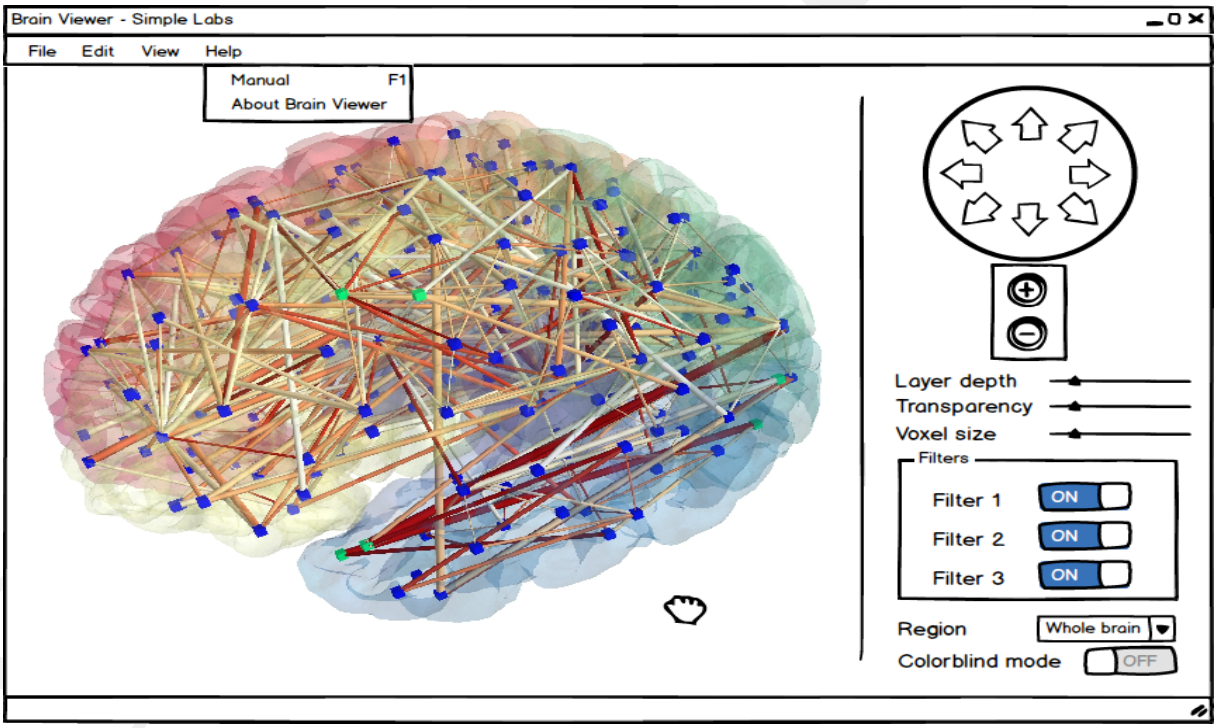


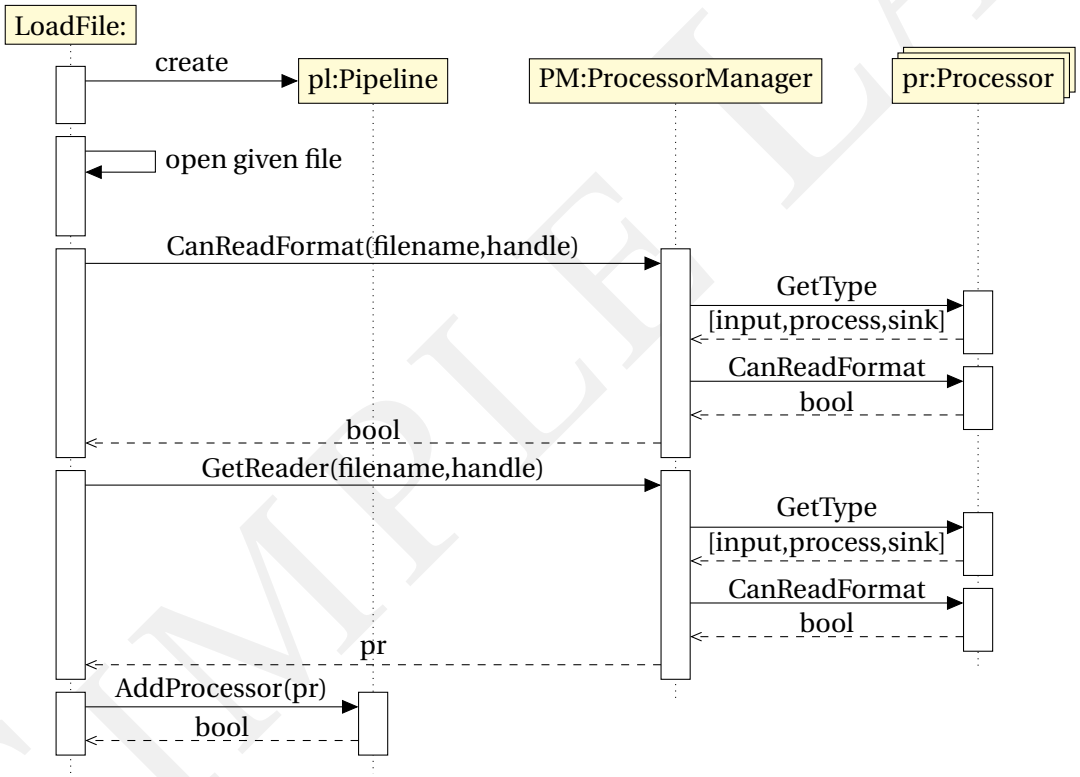
Figure 5.5: Help menu

5.7 INTERACTION VIEWPOINT

This section of the Software Design Description explains interactions between several objects of the project. Below, interactions happening on operations such as loading data, applying processors, creating pipelines can be seen. A simple written explanation is given with diagrams.

5.7.1 LOADING DATA

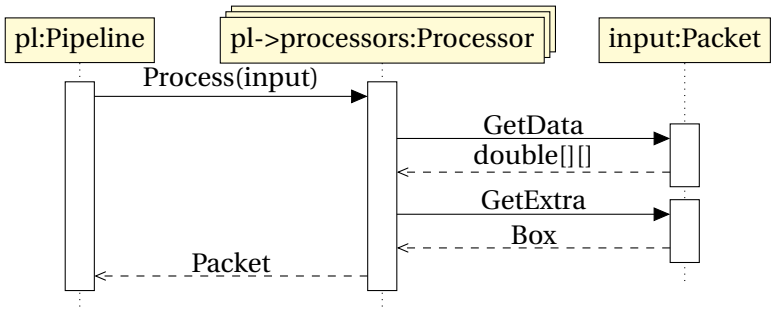
When LoadData process is initiated with a file name, first thing it does is to open given file. After that, it gives control to ProcessorManager via a CanReadFormat call. ProcessorManager forwards this call to registered "input" type Processors. First available input Processor is generated with the given filename and added to the Pipeline. This Pipeline object is then returned.



5.7.2 APPLYING PROCESSORS

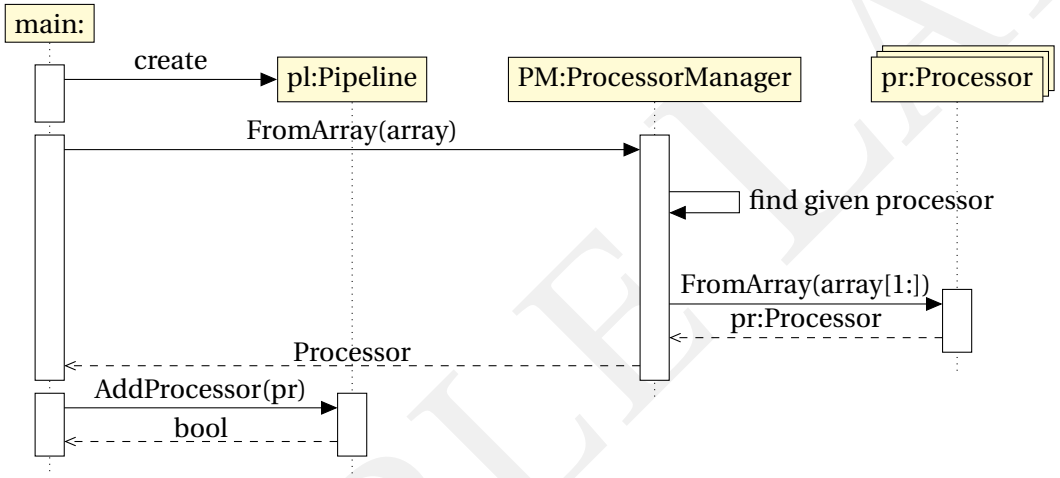
Processors are bound to Pipeline objects. However, they can be called without being bound. This flow explains how a Pipeline applies Processors. Pipeline object will generate a Packet object and follow Processor chain. A Processor is free to do whatever it wants on a

Packet.



5.7.3 CREATING PIPELINES

This diagram assumes named Processor creation from array. ProcessorManager finds wanted Processor and forwards call to it.



5.8 ALGORITHM VIEWPOINT

This section of the Software Design Description focuses on algorithmic aspects of the project. The project aims to minimize on-screen data, i.e. voxels and edges. However, this minimization process must be done delicately, so that only redundant information is purged. When this property is combined with projects big data requirements, even simple processes like sorting and finding N strongest edges become a computational problem.

In this version of the Software Design Description, there is no actual algorithm description. They will be added when figured out. Below is the types of minimization algorithms that will be used.

5.8.1 SPATIAL MINIMIZATION ALGORITHMS

This kind of algorithms try to minimize number of voxels. Sufficiently near voxels are combined. However, this type of algorithms change size of voxels, requiring a more complicated drawing method.

5.8.2 FUNCTIONAL MINIMIZATION ALGORITHMS

This kind of algorithms try to minimize number of edges drawn. 80000 voxels are not really much of a job given current capacity of graphics cards. However, routing and drawing 6.4 billion edges is computationally exhaustive. It also makes the output hard to understand. Because of these, this type of algorithms are essential to the project.

6 TIME PLANNING

The project is planned to be finished by June 2014. There will be 4 revisions. At each revision, the program will become more usable and bug-free.

At first revision, it is planned to have basic visualization functionalities. We will work with a toy dataset to understand data fields.

At second revision, processing pipeline will be implemented. This pipeline will enable us to do some basic operations on dataset. Hardware accelleration will not be used at this point.

Third and fourth revisions will be focused based on client feedback.