SOFTWARE DESIGN DOCUMENT

(v2.0)

PROJECT:
Visualization of the Human Cognition Using Brain

Group: Kernel Panic

Irmak Doğan	1819242
Esen Aytan	1819036
Zeynep Büşra Çınar	1819804
Kamyar Ghasemlou	1786896

PREFACE

This document contains the system design information about “Visualization of the Human Cognition Using Brain Data” project. This 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 Project on both client and server side applications. The first section of this document includes Project Identification, Stakeholders Identification and requirements, Composition of the developers’ team. The following sections include document purpose and design viewpoints of the system.

Revision History

Version	**Date**	**Changed**	**A/D/M**	**Brief Description**
1.0	04/01/2015	-	-	Initial Version
2.0	01/03/2015	Correction of some points		Second Version

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

Contents

1 Overview

1.1 Scope

The software to be produced is a new version of CEREBRA produced by simple Labs team as senior term project during 2013-2014 academic year. In this project it is intended to improve the existing CEREBRA application and improve its usability and features. At the end of the project, an improved version of CEREBRA will be produced with containing new filters and features.

Unity3D is the game engine in upon which CEREBRA has already been implemented, thus current project will continue on Unity3D. The new version is planned to have important improvements such as animation capability for time series data, special filter to increase comprehensibility, improved data parsing, improvement of design patterns, etc.

1.2 Purpose

This document describes how CEREBRA will be improved to satisfy the requirements and structured to implement features identified in the Software Requirements Specification document prepared by Kernel Panic.

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 precisely.

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

/toc.html

2 Definitions

3

3D	3 dimensional
CSV	“Comma Separated Value” file format
fMRI	Functional Magnetic Resonance Imaging
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
MATLAB	Matrix Laboratory
METU	Middle East Technical University
MNI	Montreal Neurological Institute
RAM	Random Access Memory
SRS	Software Requirements Specification
SDD	Software Design Document
UML	Unified Modeling Languages
Unity3D	Unity is a cross-platform game creation system developed by Unity Technologies, including a game engine and integrated development environment (IDE).

4 Conceptual Model For Software Design Description

4.1 Software Design In Context

The aim of this project is to improve CEREBRA project which visualizes fMRI data as a 3D graph of nodes. The fMRI data includes the brain response of a human in response to some particular circumstances (e.g. Picture of a red apple or when pinched).

The major capabilities of the final visual will be:

- Generation animations for a small (6 step) time series of the fMRI.
- Various filters to suppress voxels or slice the 3D graph for better understandability.
- MNI conversion and representation inside a real-life brain model.
- Different coloring for different brain regions.

Since the fMRI data is very large and complex, time and space will be main constraints. It is planned to apply various computer science optimization concepts to handle such constraints.

The target audience of this project is mostly academicians and medical institutes. Cognitive state representation and visualization of human brain is fundamentally important in neuroanatomy, neurodevelopment, cognitive neuroscience and neuropsychology. And with a tool to visualize and animate it, it is believed to contribute to the efficiency of those activities.

As mentioned above, project will be implemented in Unity3D Game Engine.

C # will be used as the programming language.

4.2 Software Design Descriptions Within The Life Cycle

4.2.1 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.

4.2.2 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.

5 Design Description Information Content

5.1 Introduction

This is an SDD document for Visualization of the Human Cognition Using Brain Data project. Detailed information about our Project design cases is given with UML diagrams. All along to document, document identification, diagrams, user views and user viewpoints are provided.

The contents are also going to be explained in this section are as follows:

- Identification of the SDD,
- Identified design stakeholders,
- Identified design concerns,
- Selected design viewpoints, each with type definitions of its allowed design elements and design languages,
- Design views,
- Design overlays,
- Design rationale

5.2 SDD Identification

This is the Software Design Document which is written on the request of Ceng 491 instructors to be able to guide the development process of our project. All sections in this document is written by all of the KernalPanic group members. The supervisors of this project are Dr. Fatoş Yarman Vural and Asst. Emre Aksan. This is not the final design document for the project and it can be subject to change in future. The date of issue of the initial version of this document is January 01, 2015. For more information, “1.Overview” can be looked at.

5.3 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 see user friendly interface because they want to have easy to use project.
- They want good performance for the project so that project can work continuously.
- They want modularity for the project so that it will be easy to insert new functionality in future.
- They want user manual for the project.
- They want the project to be completed in time.
- They want to be kept informed about the process.

5.4 Design Views

Our project has emerged from a need of an efficient, simple and smooth animation of brain data and applying some operations to this brain data to make it more meaningful. In other words some filtering techniques are used to make brain meaningful.

Design view shows estimated cost, staffing, documenting and scheduling. Relationships of the classes are easily perceived. A context view is about determining the services required, a logical view is about drawing the relations between basic entities, a dependency view and a patterns use view is about defining the relation between subsystems, an interface view is about giving insight about how the end product will be, an interaction view is about depicting the flow of information and an algorithm view is about focusing on the algorithms used is required.

5.5 Design Viewpoints

In this document, the context viewpoint is about the role of the user in the system. Then, a logical viewpoint defines the classes and the relationships between them. Dependency viewpoint is about the relationships of interconnections and dependencies inside the system. Pattern use viewpoint is about connection of subsystem into the project. Interface viewpoint is about relations of the UI modules and a mockup visualization. Interaction viewpoint is concerned with interactions between several objects. Finally, algorithm viewpoint explains required algorisms in the project.

5.6 Design Elements

Design choices are made in a way such that it can easily upgrade the project according to the needs of the stakeholders and users. Each component of the implementation like functions, variables and classes will be commented such that, for a further modification on software it will be very easy to understand the code and improve it.

The main design elements are entities (such as users, process selection, nodes and edges), attributes (includes name, type, purpose and author of attributes), relations (association or correspondence among two or more design

entities) and design constraints. These main design elements are defined inside the related viewpoints in detail in section 5.

5.7 Design Overlays

Design Overlays usually used to add information to the existing views. The interface viewpoint includes the user interface which enables user to select the process for brain data. This concept will be explained clearly when necessary in the design viewpoints section.

5.8 Design Rationale

Design choices are made to improve reusability, sustainability and provide extensibility. It should be modified according to demands from stakeholders and users. It should also be prone to change for possible improvements and requirements changes.

Moreover, Data set characteristics is a the challenging part of this project. The algorithm and software frameworks must support efficient data handling as the data sets may consist of +80K voxels and the project is intended to be used on personal computers with medium computing power. Therefore all of design decisions should be made by taking this fact into consideration.

Below are some design decisions made to comply with mentioned constraint and requirements:

5.8.1 Modularity

Processes should be modular and easy to modify/add/remove, for such an end, processes are defined as independent and in a data oriented fashion and registered to processesManager. This approach provides safe process management, when user needs to apply a process to the data, list of registered processes is obtained from process manager. This ensures that process are dynamic and easy to update.

5.8.2 Automatic File Detection

With the non-technical user in mind and to increase user-friendliness, project is designed in such a way to detect file automatically and not require the user to select data files one by one, Matlab .mat files provide variable names and a pre-defined config file would provide enough information for CEREBRA to detect relevant files and load them.

5.8.3 Configurability

Data sets are provided with different characteristics, we have decided to require a config file for each data set to increase the ease of use for different data set structures. For example voxel data comes with different variable names(e.g. XYZ, XYZ_data, XYZ_mat etc.) config file should include the variable name for voxel data so that project would be able to detect the associated file.

5.9 Design Languages

UML use case diagrams, UML component diagrams, UML class diagrams, UML sequence diagrams and ER diagrams are used in this documentation.

6 Design viewpoints

6.1 Introduction

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

- Context Viewpoint
- Logical Viewpoint
- Dependency Viewpoint
- Patterns Use Viewpoint
- Interface Viewpoint
- Interaction Viewpoint
- State Dynamics Viewpoint

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

6.2 Context viewpoint

This section of Software Design Description focuses on the services provided. The context is defined by reference to actors. Context Diagram is provided in section 5.5 Composition Viewpoint, Diagram 10.

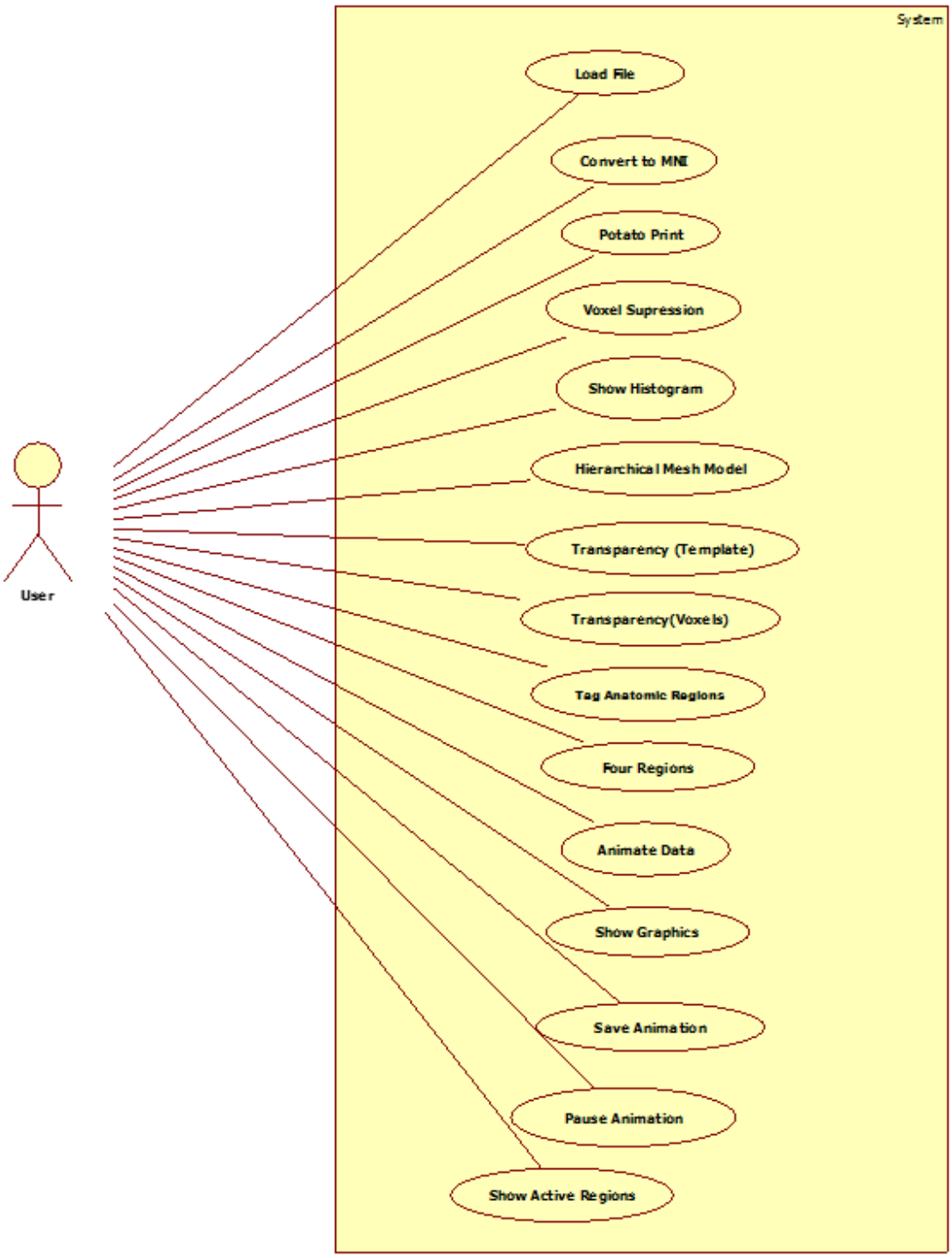

Diagram 1: System Environment

6.2.1 Design concerns

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

6.2.2 Design elements

Actors

- User: The user that uses the program.

Services

- Load File: The user loads the data taken from fMRI machine.
- Convert To MNI: Loaded voxels coordinates are converted to MNI coordinates.
- Potato Print: According to taken axis, brain slices are shown separately in different regions of the screen.
- Voxel Suppression: According to the taken two intensity values, the voxels within the specified range are shown.
- Show Histogram: During the voxel suppression operation, histogram of voxel intensity values are shown to the user.
- Hierarchical Mesh Model: Brain meshes are shown according to hierarchical level.
- Transparency (Template): Transparency of the brain template can be adjusted by using specified slider.
- Transparency (Voxels): To be able to see inner voxels, transparency of the outer voxels can be adjusted.
- Four Regions: The user can do selection between four brain lobes. Only the selected lobes will be visible.
- Tag Anatomic Regions: Showing brain' anatomic regions in different colors and naming on tooltip screen.
- Animate Data: If there is suitable data for the animation, animation is started by using specified GUI element.
- Show Graphics: After the animation, graphics with respect to the brain regions are shown.
- Save Animation: Before the animation is started, the user select the specified GUI elements to save animation. Then, the program save the process as a video file.
- Pause Animation: During the animation process, the user can pause the animation.
- Show Active Regions: During the animation process, only active regions or voxels are shown.

6.2.3 Example languages

In this section, UML Use Case Diagrams are used.

6.3 Logical Viewpoint

In this viewpoint, the classes that will be used in the project are explained with their attributes and methods. For each class, there will be a diagram to overview the class and then tables that visibility of the class diagram are shown in. Also,

definition of each class element is provided. After all classes are explained, the class diagram that shows relationships between the classes will be drawn. In this way, classes' methods and interactions will be explained in detail.

Because of the fact that this project will be used in a highly active research area, it is important that novel ideas be implemented easily. It will be achieved with a highly algorithm and data agnostic approach by project members.

6.3.1 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.

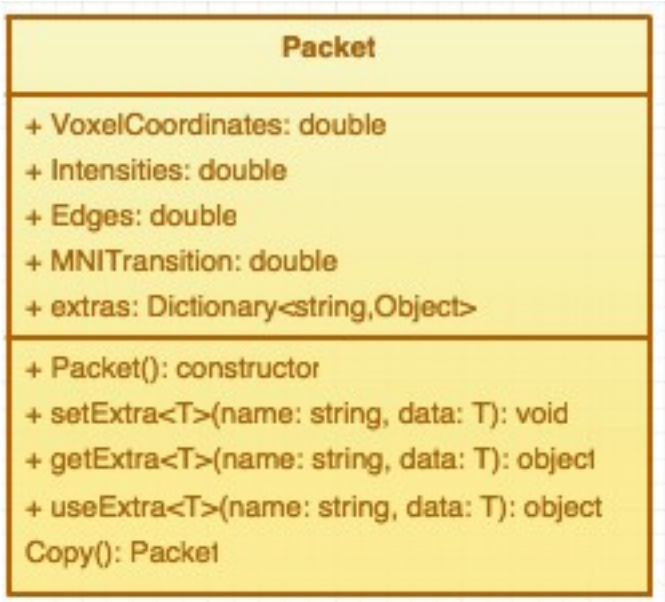

Diagram 2: Packet Class

Name	Type/Return Type	Visibility	Description
VoelCoordinates	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 data

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

6.3.2 Processor Interface

This interface gives definition about in general forms of Processors and how they could be implemented. Class of Processor is shown below.

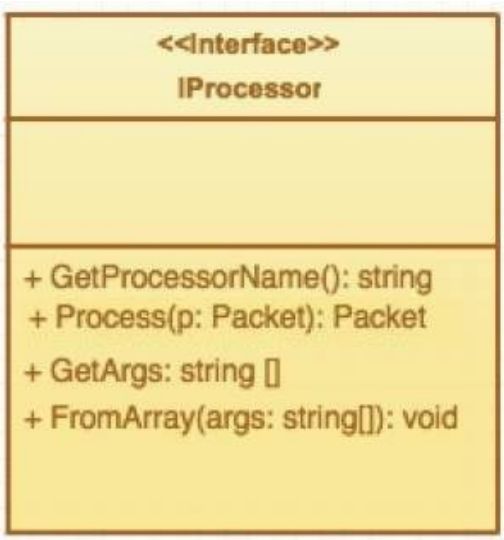

Diagram 3: Processor Interface

Name	Type/Return Type	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	Private	Sets properties of the Processor
Process(input: Packet)	Packet	Public	The real job is done here
GetProcessorName()	string	Public	Returns internal name for the Processor
GetInfo()	String	Public	Returns a simple documentation

Processor Interfaces generates ten different processes. Such as:

- ***Convert to MNI***
- ***Potato Print***
- ***Voxel Suppression***
- ***Show Histogram***
- ***Hierarchical Mesh Model***
- ***Animate Data***
- ***Pause Animation***
- ***Show Graphics***
- ***Show Active Regions***
- ***Tag Anatomic Regions***
- ***Show Brain Regions(Four Regions)***

6.3.3 Pipeline Class

Chaining Processor operations are happened here. A pipeline is an object that responsible for controlling the processors. The user can add or remove processes. Thus, the user can create his/her presets. When a Processor is added to the Pipeline, Pipeline object checks whether it is the first Processor to be added. And then, processes will be run orderly to visualize the brain data in a desired way.

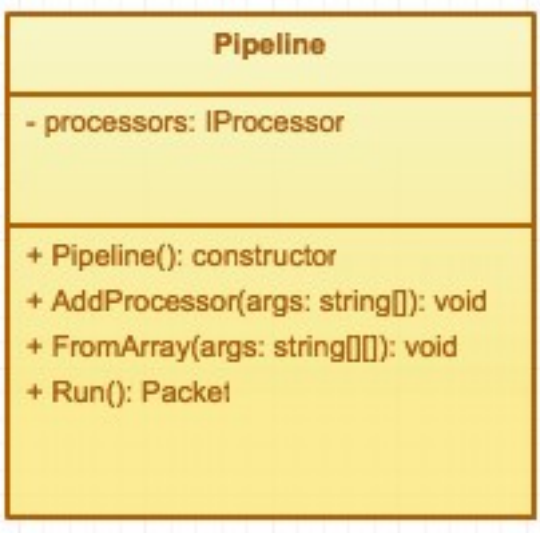

Diagram 4: Pipeline Class

Name	Type/Return Type	Visibility	Description
processors	Processor[]	Private	This is the list of Processors this Pipeline consist 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

6.3.4 ProcessorManager Class

Processor selection and generation are managed in this class. This is a static class and its’ members are all static. Each Processor must be registered with the ProcessorManager. This class is generated once and used during lifetime of the system. That’s why, singleton pattern should be applied. We have only one object from ProcessorManager class which is responsible for process management.

Diagram 5: ProcessorManager Class

Name	Type/Return Type	Visibility	Description
processors	Processor[]	Private	This is the list of Processors
ProcessManager()	<<constructor>>	Public	Static constructor
Register(p: Processor)	Void	Public	Register p with ProcessManager
FromArray(arg: string[])	Void	Public	Constructs “processors” with given information
GetReader(fn: string)	Processor	Public	Finds the input Processor that can read given source.

6.3.5 InitBehaviour Class

This class is responsible for creating a packet to render by applying all specified filters to the packet. The user loads the data thanks to this class and also, GUI operation are handled via this class.

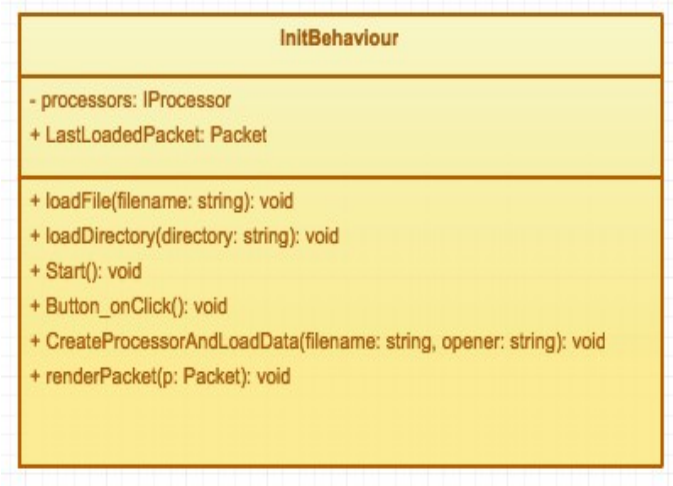

Diagram 6: InitBehaviour Class

Name	Type/Return Type	Visibility	Description
processors	Processor[]	Private	This is the list of Processors
LastLoadedPacket	Packet	Public	Returns Packet which is loaded lastly.
loadFile(filename: string)	Void	Public	Load file to system with giving input as a filename.
Start()	Void	Public	To begin initBehavior
Button_onClick()	Void	Public	Analyze or filter the data according to the specified work for that button.
CreateProcessorAndLoadData(filename: string, opener: string)	Void	Public	After clicked “+” button, create processor and load data concern with the process. It gives as input filename and opener.

6.3.6 OptimizedPacketRenderer Class

This class is responsible for render the data which was taken from initBehaviour. During rendering process, the class checks the packetToRenderer if this packet is for image or animation by looking at the data columns. If there are more than one column for one voxel, it is an animation packet, then its intensity values should be changed accordingly.

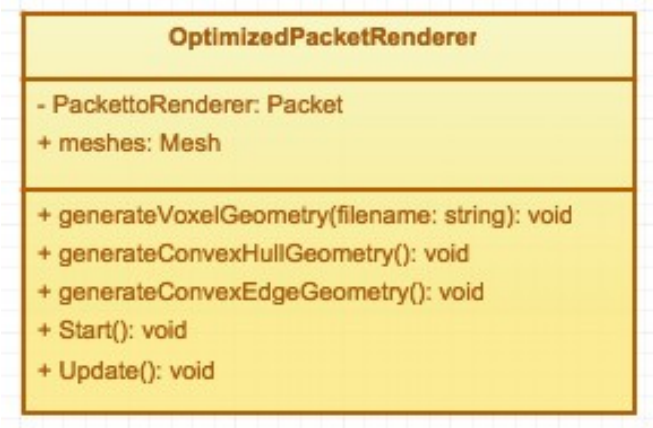

Diagram 7: OptimizedPacketRenderer Class

Name	Type/Return Type	Visibility	Description
PackettoRenderer	Packet	Private	The packet which will be rendered by Unity.
Meshes	Mesh	Private	The Unity object which consist of vertices, triangles, normal, intensities data.
generateVoxelGeometry(filename: string)	void	Public	Create voxel in desired geometry
generateConvexHullGeometry()	Void	Public	Create convex hull in desired geometry
generateConvexEdgeGeometry()	Void	Public	Create edges in desired geometry
Start()	void	Public	Start rendering process of the system
Update()	void	Public	Update the scene with respect to the delta time.

6.3.7 Relationships between Classes

All these relationship are shown on Class Diagram in the below figure.

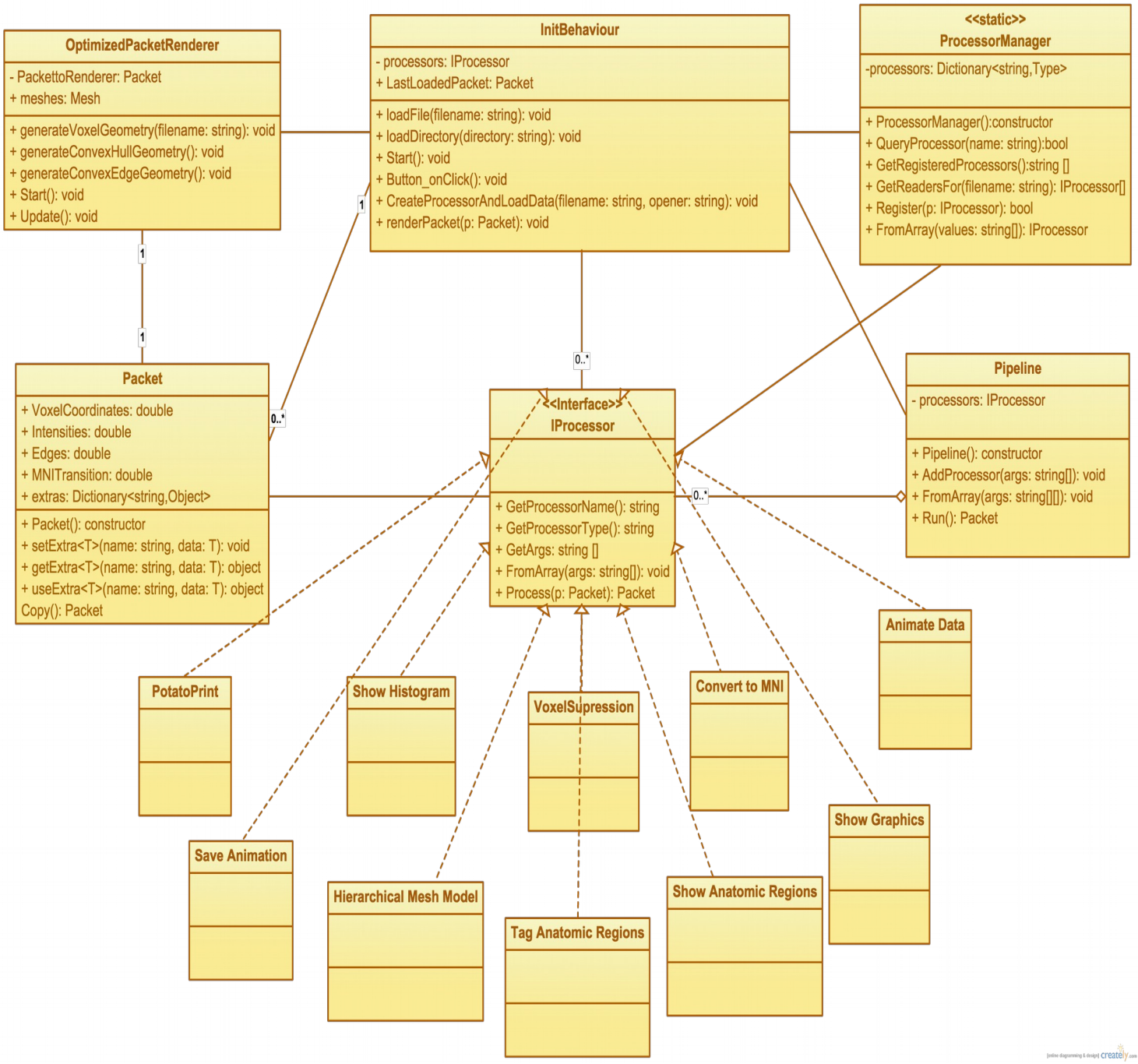

Diagram 8: Class Relationship

6.4 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 the section.

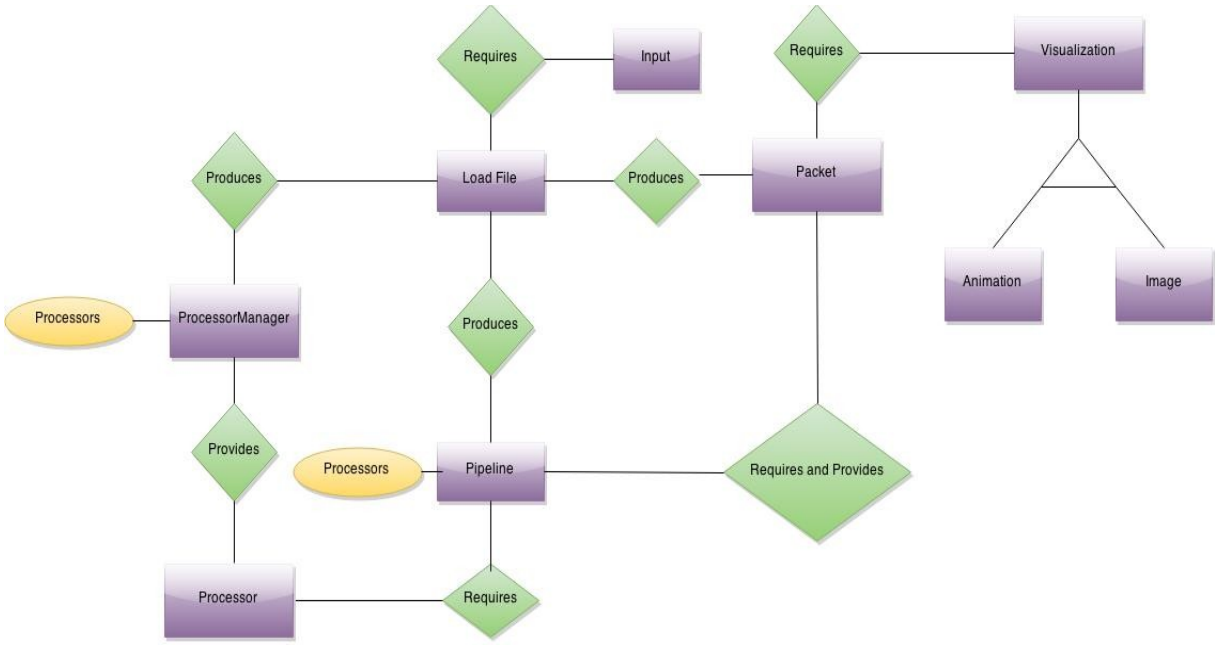

Diagram 9: ER Diagram

Dependency viewpoint will list the subsystems and explain the interconnections among them in detail.

It 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, Pipeline, and Visualization. Also, visualization is separated in two parts called animation and image.

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.

6.5 Composition 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()	After taken the data by fMRI machine, the function is called by user to load the data. Then, Input subsystem is started.
downSampleData()	This function is called by Input subsystem 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()	Filtering subsystem calls this function to connect the Visualization subsystem after execution of quantizeData() function. The processed data is shown as 3D Image by this function. This function uses built-in Unity3D functions and OpenGL libraries and implemented functions.
changeDisplay()	This function is called if the user adjusts transparency, MNI coordinates, colors, rotation, potato print, voxel suppression or zoom or changes the display by clicking on "Show Side-by-Side", “Hierarchical Mesh Model” or "Four Regions" buttons. This function cannot be called before the showBrain() function.
updateDisplay()	This function is called by Visualization subsystem to update the display with respect to specified changes by the user.
startAnimation()	This function is called if there is suitable data for animation. Visualization subsystem calls this function to connect Animation subsystem. The processed data is shown as 3D Animation by this function. This function uses built-in Unity3D functions and OpenGL libraries and implemented functions.
changeAnimation()	This function is called if the user wants to change

	animating data by adjusting transparency, MNI coordinates, colors, rotation, voxel suppression or zoom or changes the display by clicking on, “Show Active Regions”, "Show Side-by-Side", “Hierarchical Mesh Model” or "Four Regions" buttons. This function cannot be called before the startAnimation() function.
updateAnimation()	This function is called by Visualization subsystem to update the display with respect to specified changes by the user.

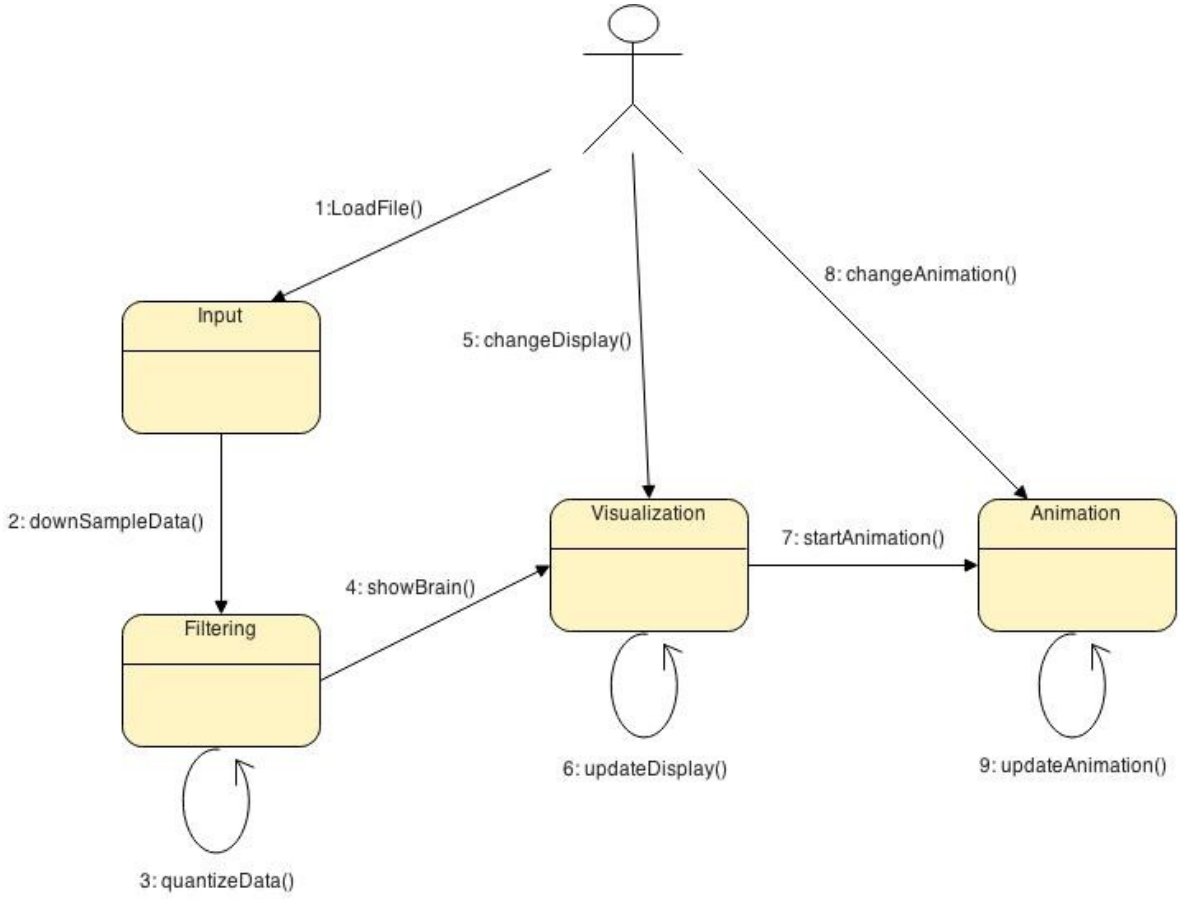

Diagram 10: Context Diagram

The function is called with respect to the 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. If the data is suitable to animate, that is, the data is a time series data, animation is started with seventh function. Then, animation process continues with user interactions until the program is closed.

6.6 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 should 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 e.g. techniques. The output of this component will be ready-to-draw voxel position parameters, voxel intensity values and edges (arclengths).

Thirdly, visualization component will use 3D rendering engine and draw the image to the screen.

Lastly, animation component will use GPU and shaders to render 3D animation to the screen.

This is visualized using UML component diagram below.

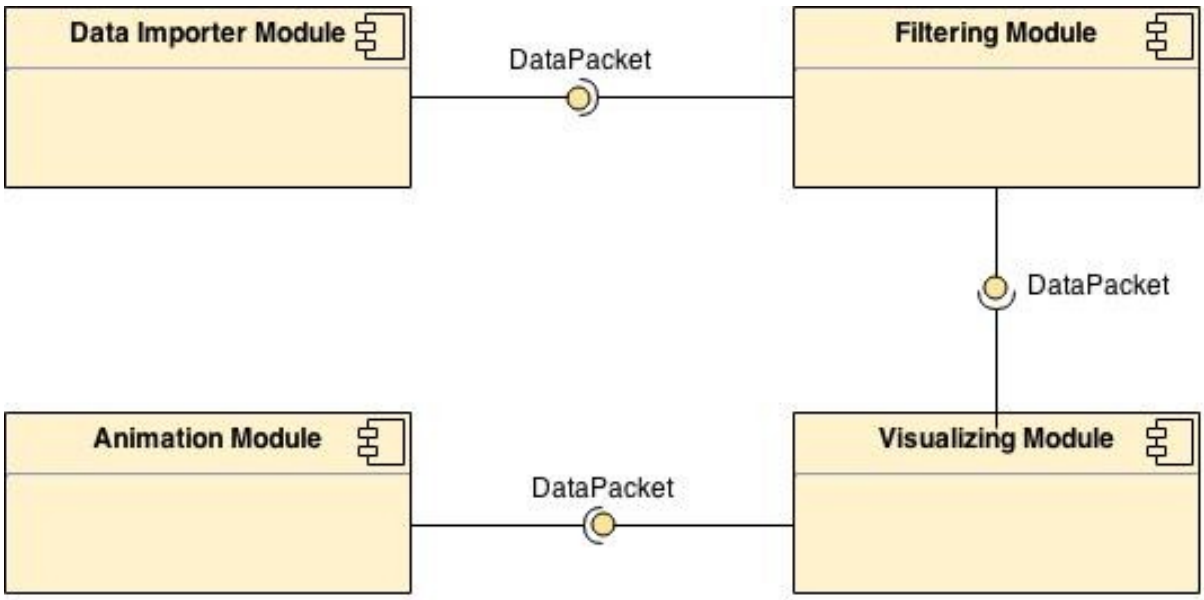

Diagram 11: UML component diagram

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. Transparency for voxels and transparency for

brain template 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 applied via “+” and “-” buttons by adding the listview. 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.

Our intention for the behavior 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.

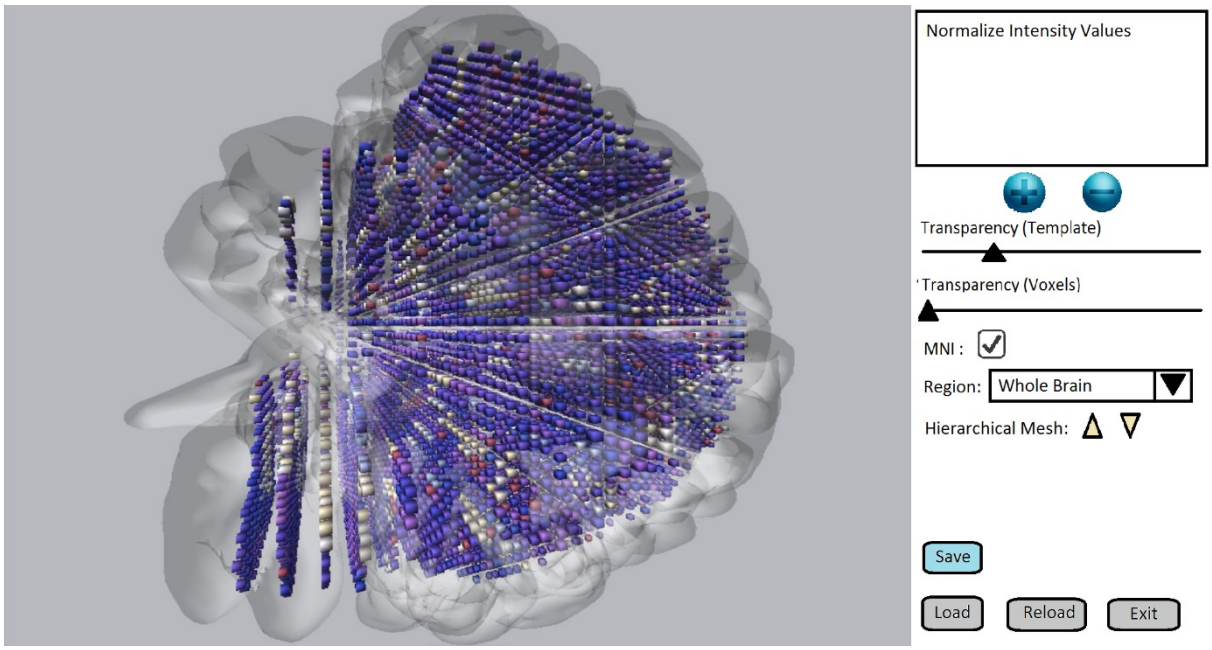

Figure 1: MNI Conversion

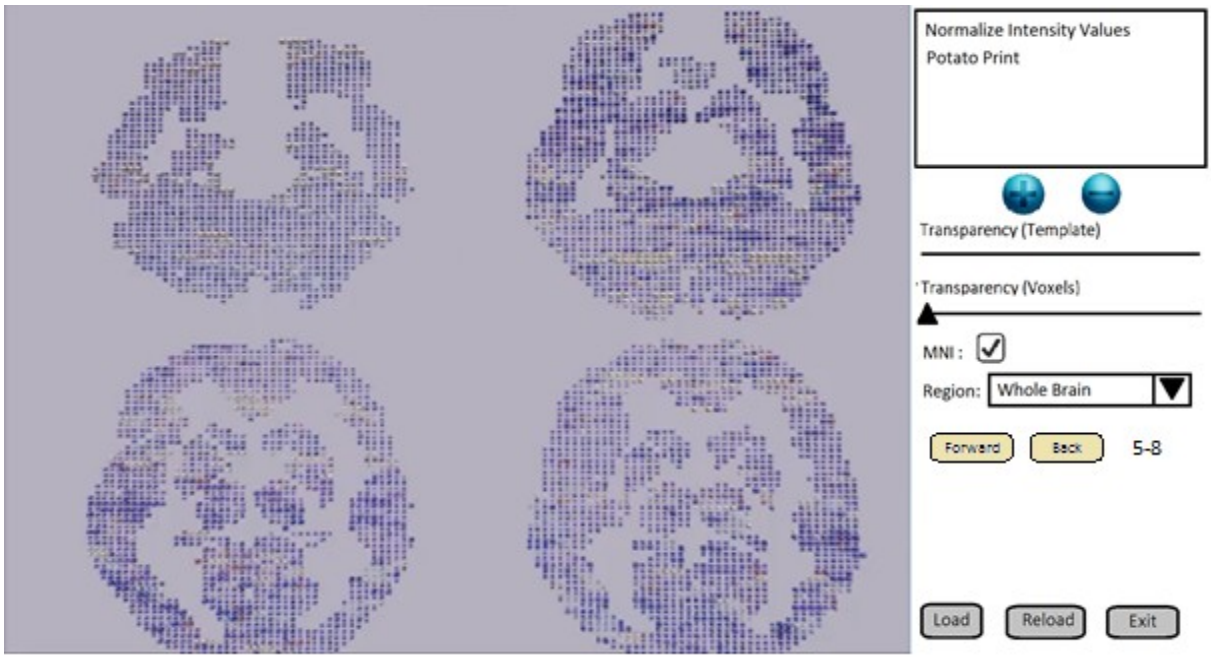

Figure 2: Potato Print

After inserting the Potato Print process Brain Template and Hierarchical Mesh Model UI is removed from the GUI. “Forward” and “Back” buttons are inserted to the GUI to display forward and backward slices as quads in the screen. When a slice is double clicked, the slice will pop up.

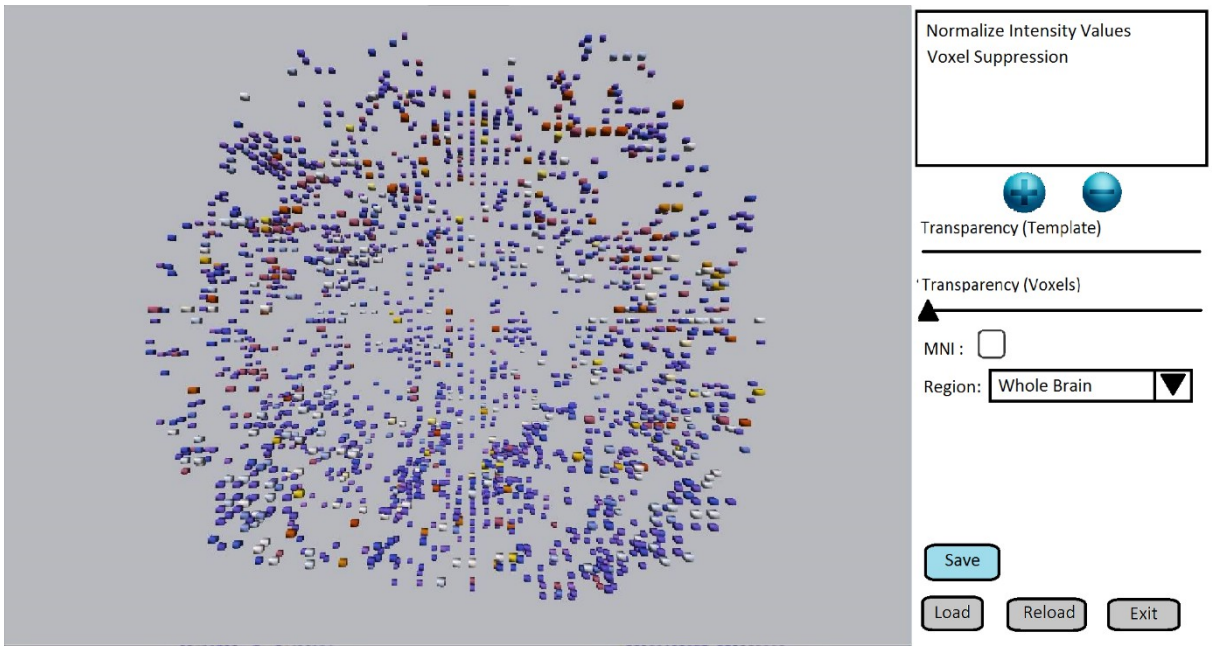

Figure 3: Voxel Suppression

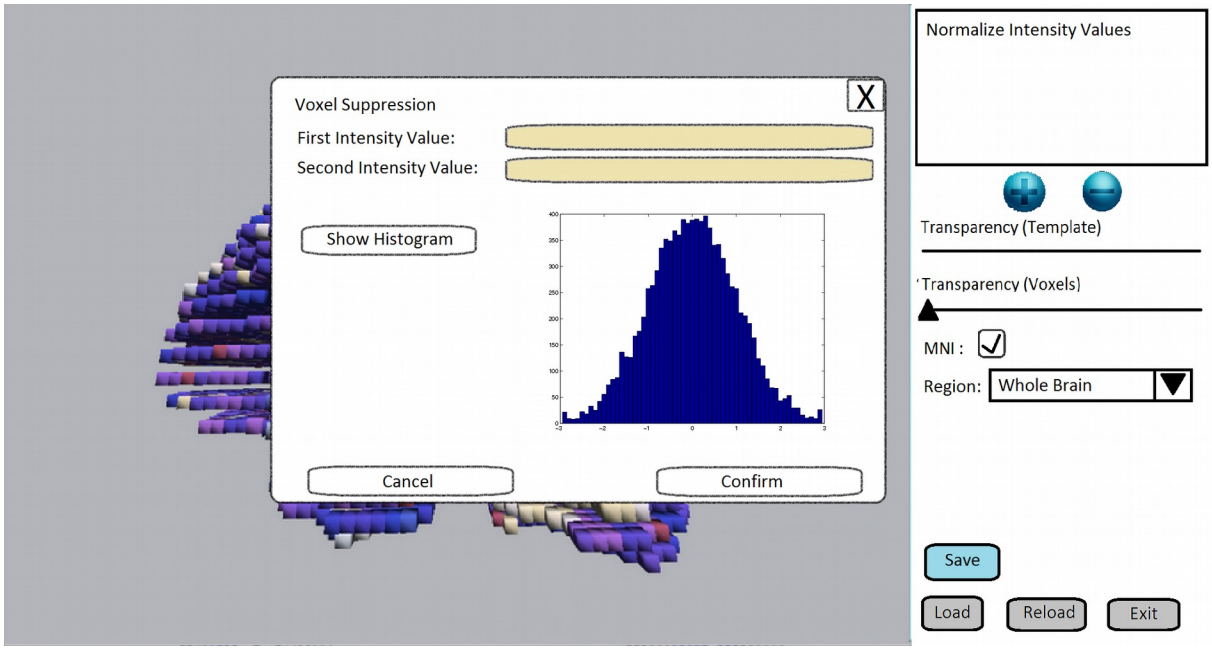

Figure 4: Histogram of Voxel Suppression

While inserting the voxel suppression process, the user can see a MATLAB histogram of voxel intensity values by clicking the “Show Histogram” button. After inserting this process, hierarchical mess model UI is removed.

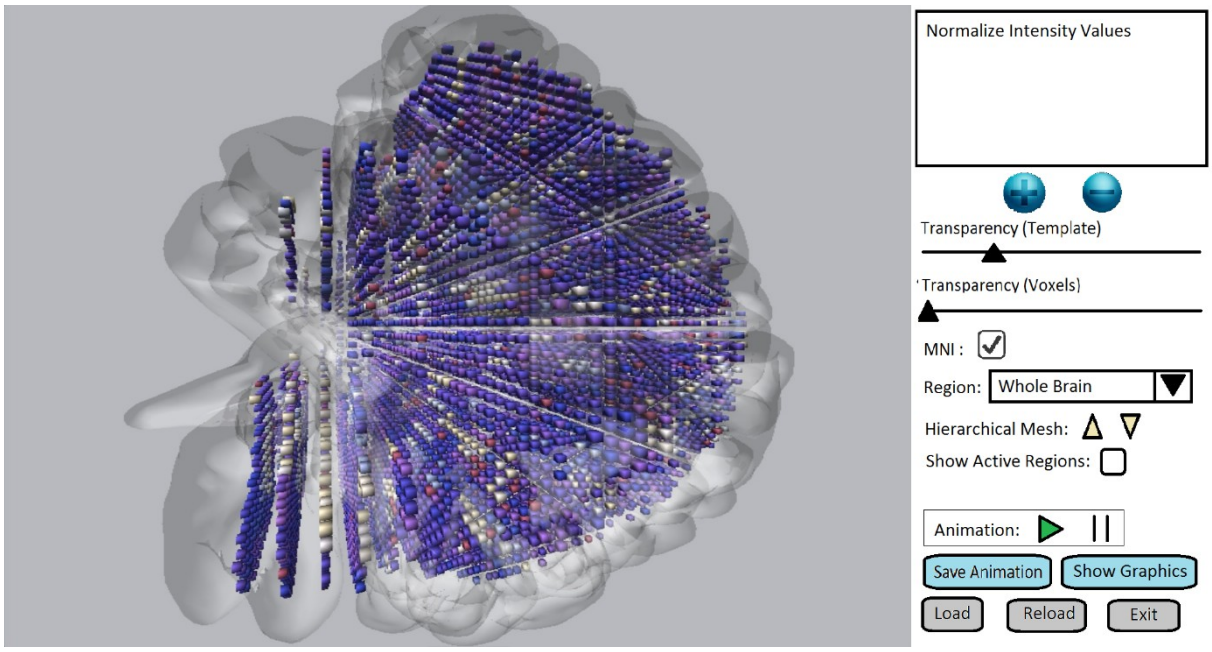

Figure 5: Start Animation

6.7 Interaction Viewpoint

In this viewpoint, the interaction and the connection between user interface elements will be explained screen by screen in detail. Thus, the interactions between these objects and how they communicate should be explained. The interaction viewpoint is chosen for this purpose.

In below, interactions happening on operations such as loading data, applying processors, creating pipelines can be seen. A simple written explanation is given with diagrams.

6.7.1 Loading Data

When LoadData function 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.

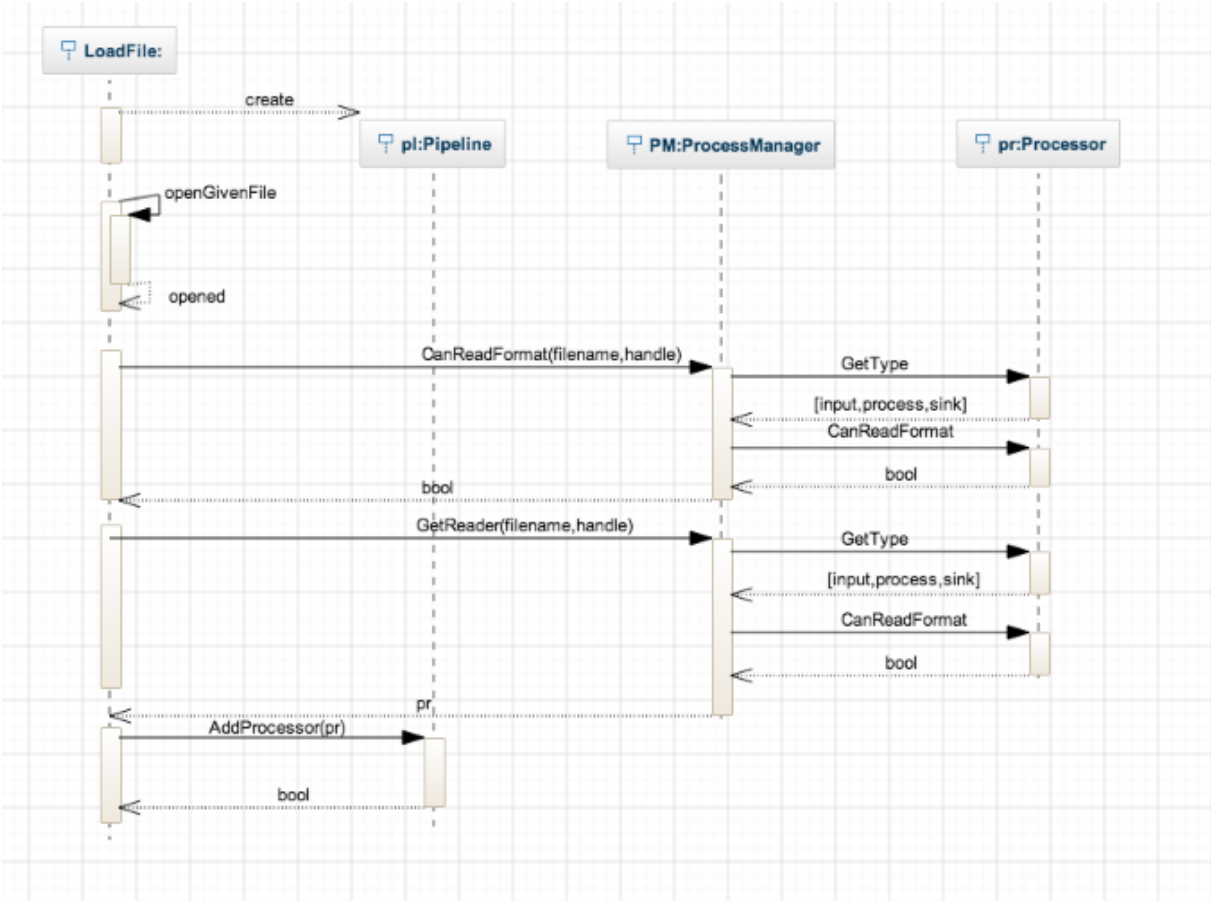

Diagram 12: Loading Data Sequence Diagram

6.7.2 Applying Processors

All the processors are bound to Pipeline objects. But, 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 can do whatever it wants on a Packet.

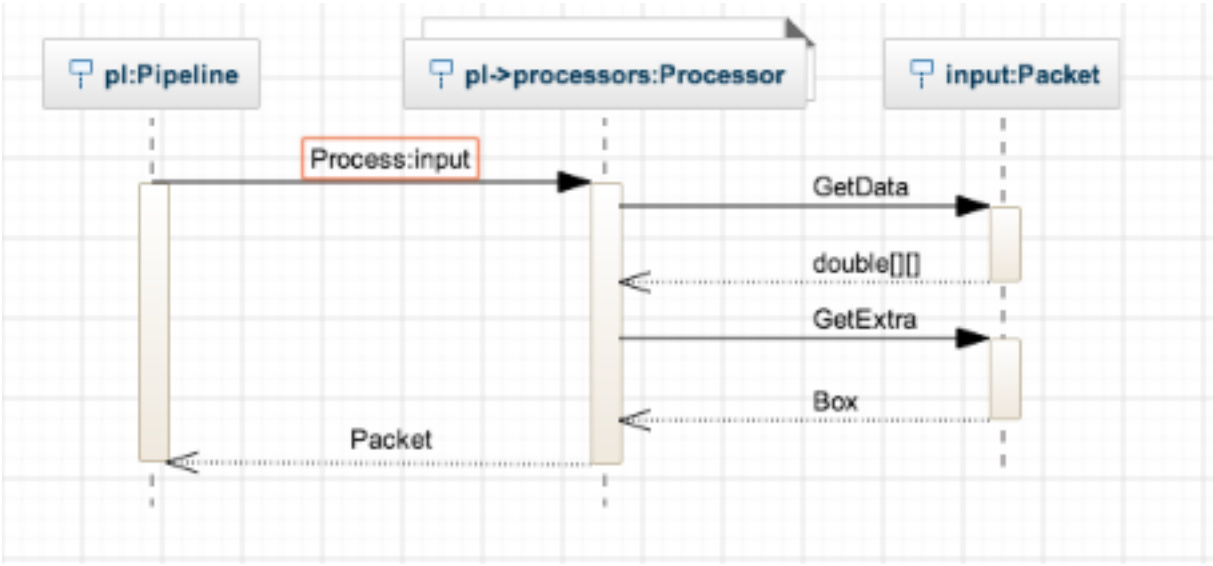

Diagram 13: Applying Processors Sequence Diagram

6.7.3 Creating Pipelines

This diagram assumes named Processor creation from array. ProcessorManager finds wanted Processor and forwards call to it. Purpose of this class is explained in the 5.3.3 Pipeline Class part of the Logical Viewpoint section.

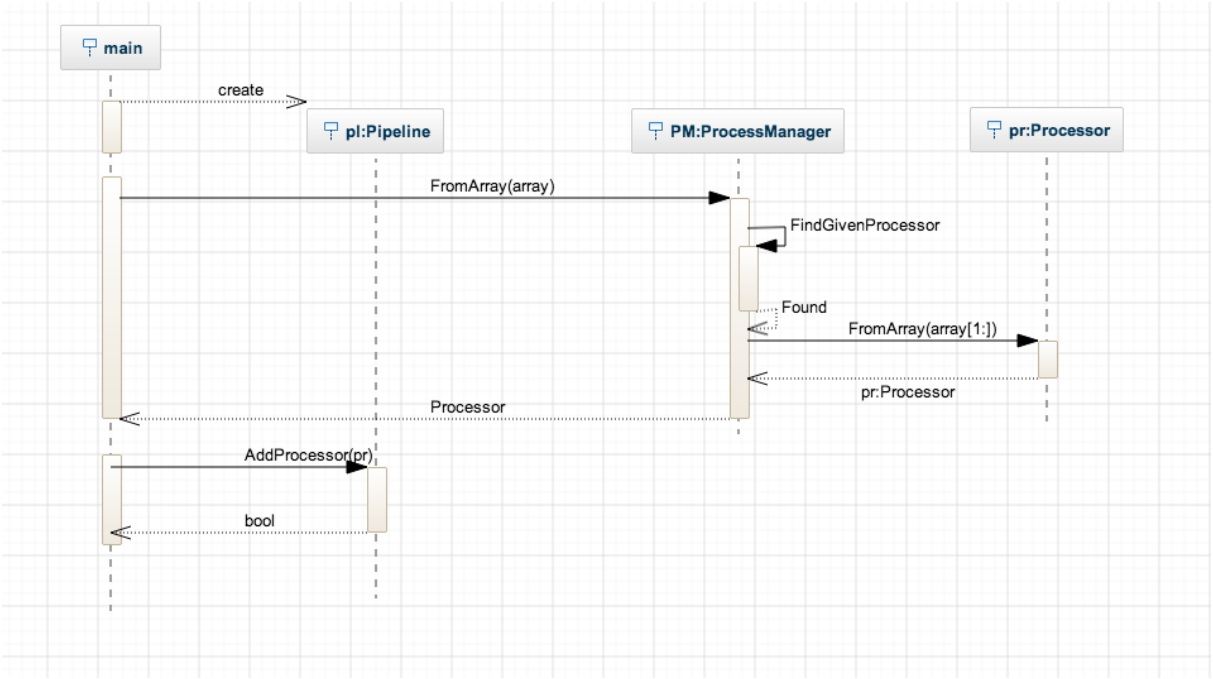

Diagram 14: Creating Pipelines Sequence Diagram

6.7.4 Packet Rendering

In this diagram, we will show that initBehavior sends the packet to optimized packet renderer. Then, Optimized packet renderer renders the Packet by using Unity3d rendering engine.

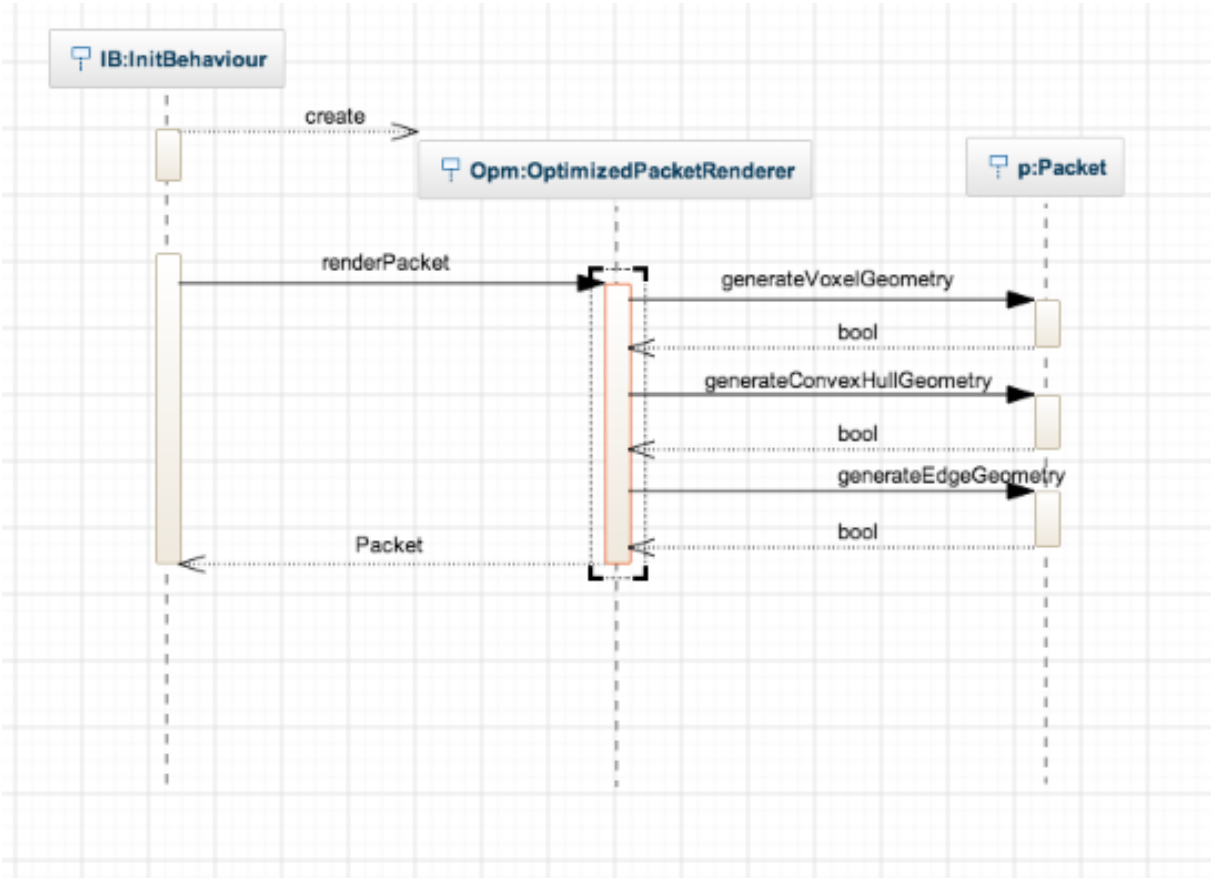

Diagram 15: Packet Rendering Sequence Diagram

6.7.5 Animation Pipeline

In this diagram, we will show animation process. Optimized packet renderer controls the packet. If voxel intensity values include time series data, packet renderer will animate it. On the other hand, if it includes instance time data that is there is only one column for intensity values, only 3D image belongs to that instance time will be shown on screen.

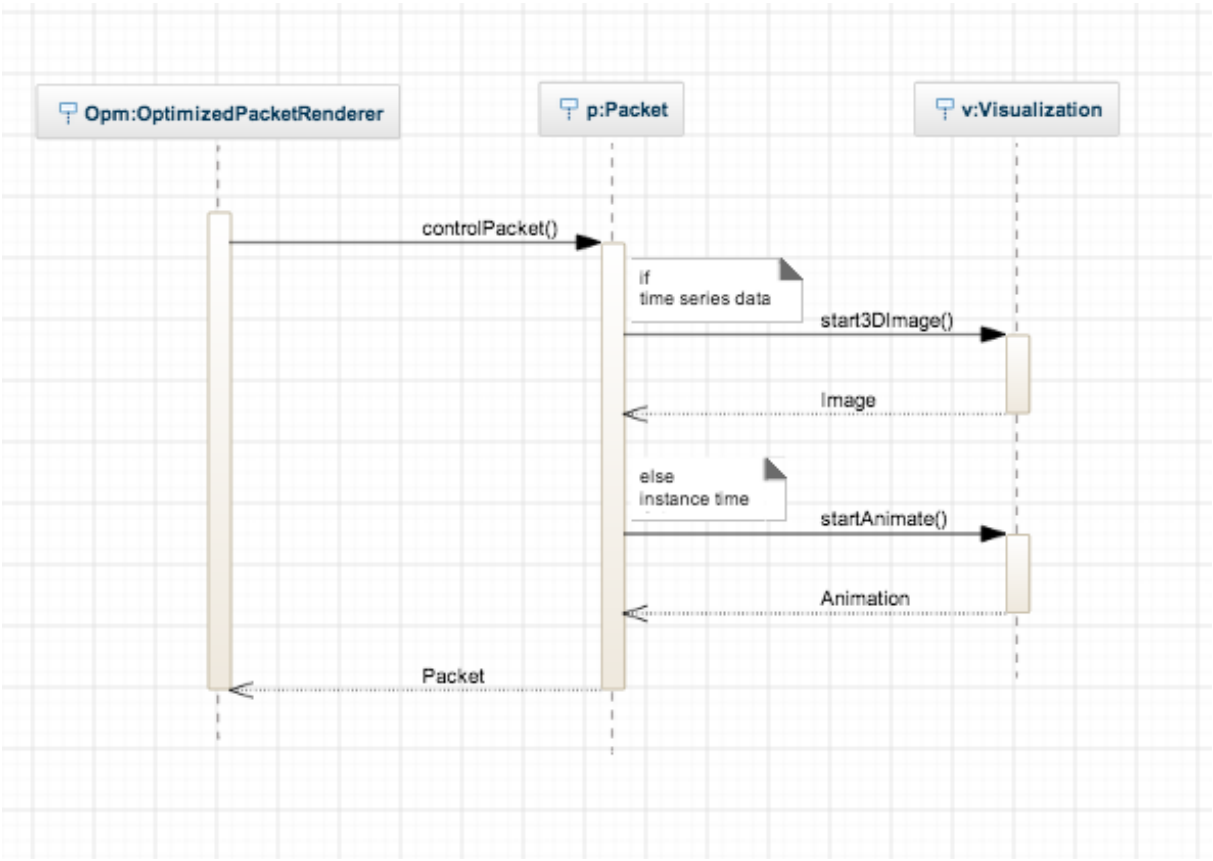

Diagram 16: Animation Pipeline Sequence Diagram

6.8 State Dynamics Viewpoint

As shown in the diagram below, the application starts with a worker thread and waits an input brain data which is fMRI data. It analyzes the data and bring it in compliance with visualization. During the visualization step, user can do same changes on the data such as normalizing, zooming, converting, rotating, adding new processes etc. or if the data is a time series data animation process can be started by the user by clicking the “Animate” button. During the animation state, the user can pause the animation by clicking “Pause” button. After that, if the user clicks “Animate” button, the animation resumes. Also, during the animation state, user can do some changes on the data such as showing active regions, showing graphics in addition to all filters applied to 3D image. The user can exit the program in somewhere of the application.

1 Pattern Use Viewpoint

In this project, factory design pattern is used. The factory method pattern is a creational pattern which uses factory methods to deal with the problem of creating objects without specifying the exact class of object that will be created.

The reason of using that, we use different processes for visualizing brain differently.

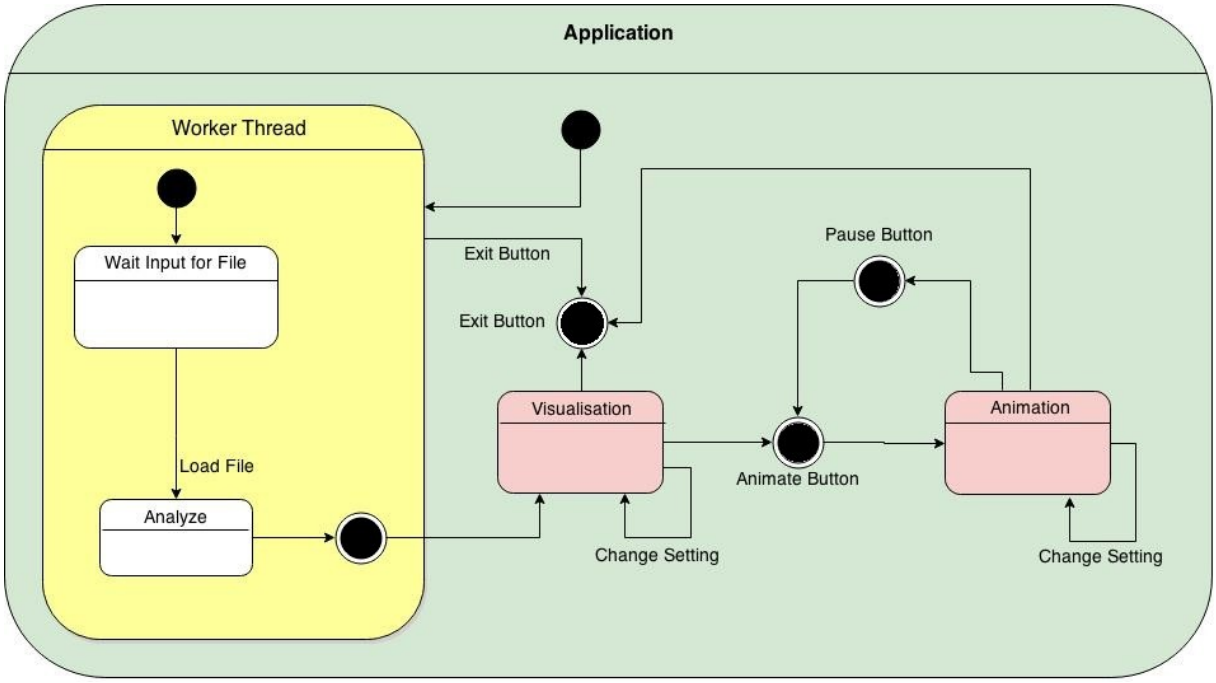

Diagram 17: State Transition Diagram

7 Traceability Matrix

	UC0 (Section 2.2.0 In SRS)	UC1 (Section 2.2.1 In SRS)	UC2 (Section 2.2.2 In SRS)	UC3 (Section 2.2.3 In SRS)	UC4 (Section 2.2.4 In SRS)	UC5 (Section 2.2.5 In SRS)	UC6 (Section 2.2.6 In SRS)	UC7 (Section 2.2.7 In SRS)	UC8 (Section 2.2.8 In SRS)	UC9 (Section 2.2.9 In SRS)	UC10 (Section 2.2.10 In SRS)	UC11 (Section 2.2.11 In SRS)	UC12 (Section 2.2.12 In SRS)	UC13 (Section 2.2.13 In SRS)	UC14 (Section 2.2.14 In SRS)	UC15 (Section 2.2.15 In SRS)
Diagram 1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Diagram 2	X	X	X	X	X	X	X	X	X		X	X			X	X
Diagram 3	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Diagram 4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Diagram 5	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Diagram 6	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Diagram 7	X	X	X	X	X	X	X	X	X		X	X			X	X
Diagram 8	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Diagram 9	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Diagram 10	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Diagram 11	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Figure 1	X								X							
Figure 2	X	X														
Figure 3	X		X													
Figure 4	X											X				
Figure 5	X					X									X	X
Diagram 12	X	X	X	X	X	X	X	X	X		X	X			X	X

Diagram 13	X	X	X	X	X	X	X	X	X		X	X				
Diagram 14	X	X	X	X	X	X	X	X	X		X	X				
Diagram 15	X	X	X	X	X	X	X	X	X		X	X			X	X
Diagram 16	X														X	X
Diagram 17	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

8 Conclusion

In this design document, we provide details about the software architecture and initial design of the project “Visualization of Human Brain Data” and also implementation details of the project with respect to viewpoints, modules and data design. This document is intended to be used as a reference to the both stakeholders defined before in this document and also to the team Kernel Panic.