DESIGN SPECIFICATION

SKETCHUP BLOCKS - ANDREW SMITH - CSIR

Version 1.4

13/10/2013

11081092 - ET, van Zyl

11211548 - WH, Oldewage

11183153 - JL, Coetzee

CONTENTS

CHANGE LOG

Version 0.1	23/07/2013	Document created.
Version 0.2	24/07/2013	Added Introduction and System Description.
Version 0.22	24/07/2013	Added Glossary.
Version 0.24	24/07/2013	Added process diagrams.
Version 0.5	26/07/2013	Inserted GUI designs and class diagrams.
Version 0.9	26/07/2013	Finalized document for demo.
Version 1.0	28/07/2013	Added use cases and associated diagrams.
Version 1.2	25/08/2013	Added detailed description of calculations.
Version 1.3	14/09/2013	Updated GUI design.
Version 1.4	13/10/2013	Revised cover page

1. INTRODUCTION

1.1 PURPOSE

The purpose of this document is to give a high-level abstraction of the architectural strategies/tactics that will be used in the Sketchup Blocks project. This document aims to provide clear motivation for the chosen architectural design as well as an explanation of how it will serve to implement the specifications required by the Vision and Scope document for the Sketchup Blocks project.

1.2 PROJECT SCOPE

Sketchup Blocks aims to provide the client with a system that will convert physical 3D models into digital form. The system will include software and hardware components.

Hardware components:

- The physical blocks with fiducial markers attached (called 'smart blocks').
- The construction floor on which the smart block constructions are built.
- Approximately five web cameras.

The scope of the software components will broaden from release to release. The client only requires the first release to consider the project successful. The releases following the first release will address additional concerns and implement extra features mentioned by the client.

Release 1:

The initial system will be able to recognize a small set of geometrically simple objects without any time constraints. After the blocks have been transformed into a digital model, the vertices of the model will be output to an object file for import into Google Sketchup.

Release 2:

This release aims to limit the time taken to recognize a fiducial and recreate the model it represents. The amount of constructed and recognizable blocks will also be increased. The setup of the system will be simplified to improve mobility and allow for fast setup. The removal of blocks will be detected without extra input from the user.

Release 3

This release aims to add a viewing window that will display the model as it being created thereby providing visualization before exporting to a model. The detection of the fiducials should be possible in suboptimal lighting conditions.

Optional release

Custom blocks may be imported to the smart block database. Users can view the construction of other users over the network.

1.3 RELATED DOCUMENTS

Project Vision and Scope (Sketchup Blocks project, by CICO).

Architectural Requirements (Sketchup Blocks project, by CICO).

Technical Specification (Sketchup Blocks project, by CICO).

Project Management (Sketchup Blocks project, by CICO).

Test Document (Sketchup Blocks project, by CICO).

Sketchup Blocks Developer Blog (http://sketchupblocks.wordpress.com/).

2. SYSTEM DESCRIPTION

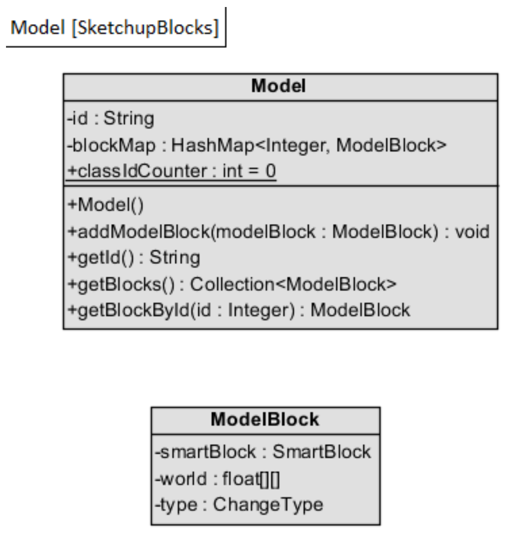
The project subscribes to the concept of the Internet of Things. The Internet of Things is centred around the idea of ‘smart objects’ which are physical objects and devices that are given a virtual representation and linked to information networks. These smart objects will ideally be able to capture, react to and communicate about data captured from their physical environment autonomously.

This project aims to create a system that can convert physical 3D models into digital models that can be viewed and edited in Google Sketchup. The physical model will be built with blocks that have fiducial markers attached. The locations of these blocks will be picked up on web cameras by Reactivision software and the resulting data will be used to create digital 3D models that can be viewed and exported to Google Sketchup.

3. DESIGNS

3.1 SYSTEM DESIGN

The Sketchup Blocks system is centred about the Model, which stores the digital representation of the physical model. Particularly, it stores the positions and vertices of all the blocks on the construction floor.



Model [SketchupBlocks]
Model
-id : String
-blockMap : HashMap<Integer, ModelBlock>
+classIdCounter : int = 0
+Model()
+addModelBlock(modelBlock : ModelBlock) : void
+getId() : String
+getBlocks() : Collection<ModelBlock>
+getBlockById(id : Integer) : ModelBlock
ModelBlock
-smartBlock : SmartBlock
-world : float[][]
-type : ChangeType

Fig 3.1.1

Other classes' access to the Model is strictly via a Lobby. The Lobby provides a uniform interface through which the model's internal state can be retrieved and modified. This means of communication will ease the implementation of the project's networking capabilities.

The construction of this Model is very nearly a pipeline process, in which many different classes are involved in processing the camera input to cause a change in the Model's structure. This process will be described in detail below.

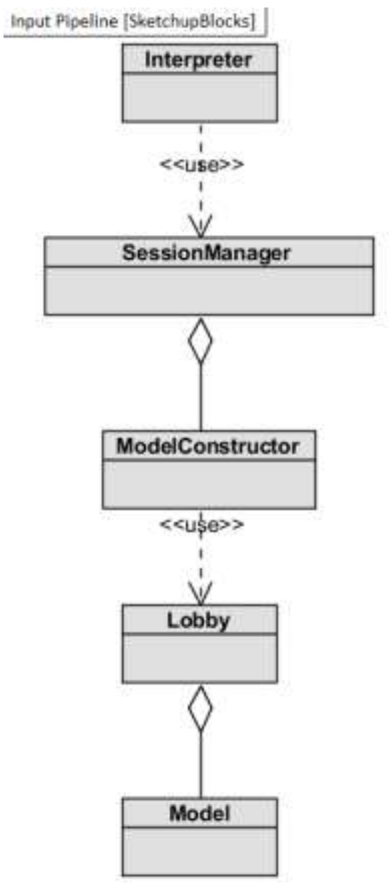


Fig 3.1.2

The lowest level of work is done by the Interpreter class. The Interpreter class receives data from ReacTIVision regarding the location and orientation of the fiducial markers recognized by the camera. It then passes this information along the pipeline for other classes to use.

There will be an instance of the Interpreter class running for every camera connected to the system. The Interpreter implements the TuioProcessing interface to allow it to intercept the ReacTIVision data. The Interpreter responds to TUIO Object events by wrapping relevant data in CameraEvent objects and passing them along to the SessionManager.

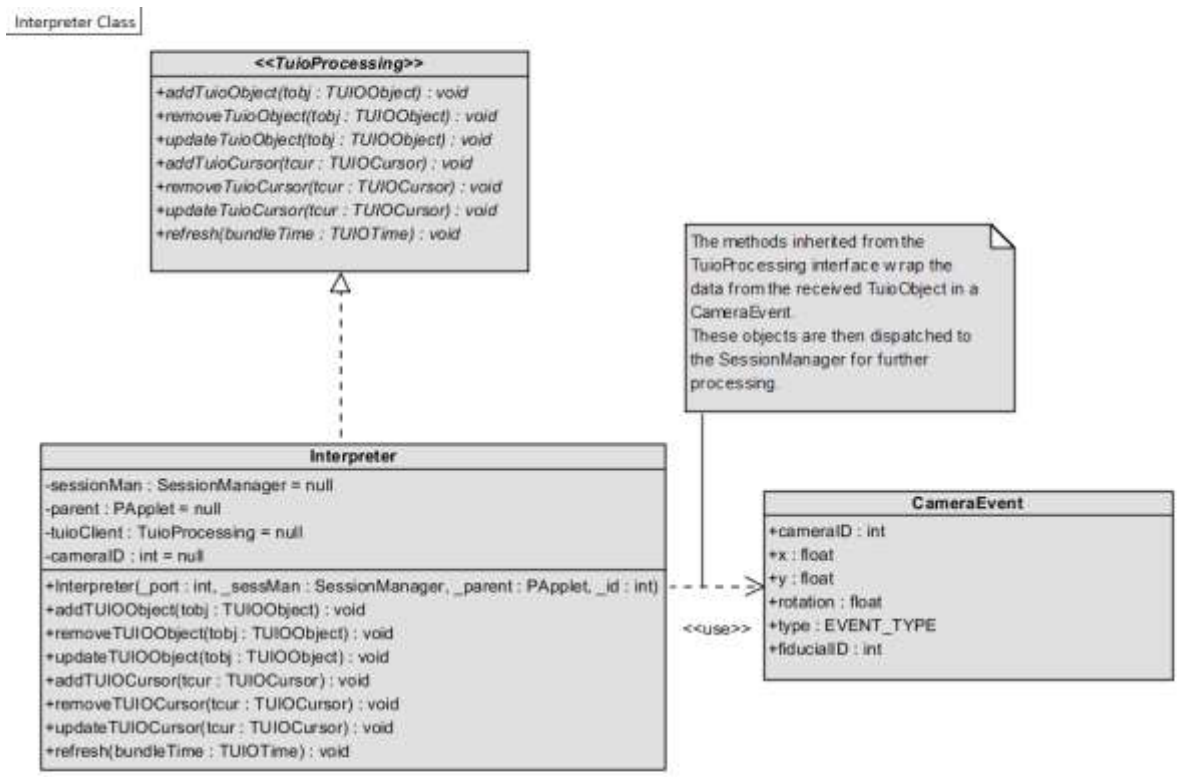

Interpreter Class
<<TuioProcessing>>
+addTuioObject(tobj : TUIOObject) : void
+removeTuioObject(tobj : TUIOObject) : void
+updateTuioObject(tobj : TUIOObject) : void
+addTuioCursor(tcur : TUIOCursor) : void
+removeTuioCursor(tcur : TUIOCursor) : void
+updateTuioCursor(tcur : TUIOCursor) : void
+refresh(bundleTime : TUIOTime) : void
The methods inherited from the TuioProcessing interface wrap the data from the received TuioObject in a CameraEvent.
These objects are then dispatched to the SessionManager for further processing.
Interpreter
-sessionMan : SessionManager = null
-parent : PApplet = null
-tuioClient : TuioProcessing = null
-cameraID : int = null
+Interpreter(_port : int, _sessMan : SessionManager, _parent : PApplet, _id : int)
+addTUIOObject(tobj : TUIOObject) : void
+removeTUIOObject(tobj : TUIOObject) : void
+updateTUIOObject(tobj : TUIOObject) : void
+addTUIOCursor(tcur : TUIOCursor) : void
+removeTUIOCursor(tcur : TUIOCursor) : void
+updateTUIOCursor(tcur : TUIOCursor) : void
+refresh(bundleTime : TUIOTime) : void
<<use>>
CameraEvent
+cameraID : int
+x : float
+y : float
+rotation : float
+type : EVENT_TYPE
+fiducialID : int

Fig 3.1.3

In addition to other duties which will be discussed later, the Session Manager examines the Camera Events that it receives from Interpreters.
It then queries a database of all the blocks to determine the block to which a particular fiducial belongs and then to retrieve all the information saved about that block (such as the other fiducials associated with it, their position on the block, the block size and shape, etc). This information, along with the Camera Event is wrapped in an Input Block and is then passed to the Model Constructor.

The Model Constructor uses the data it receives to infer the positions of the physical building blocks in 3D space. The details of these calculations can be found in the Technical Specifications.

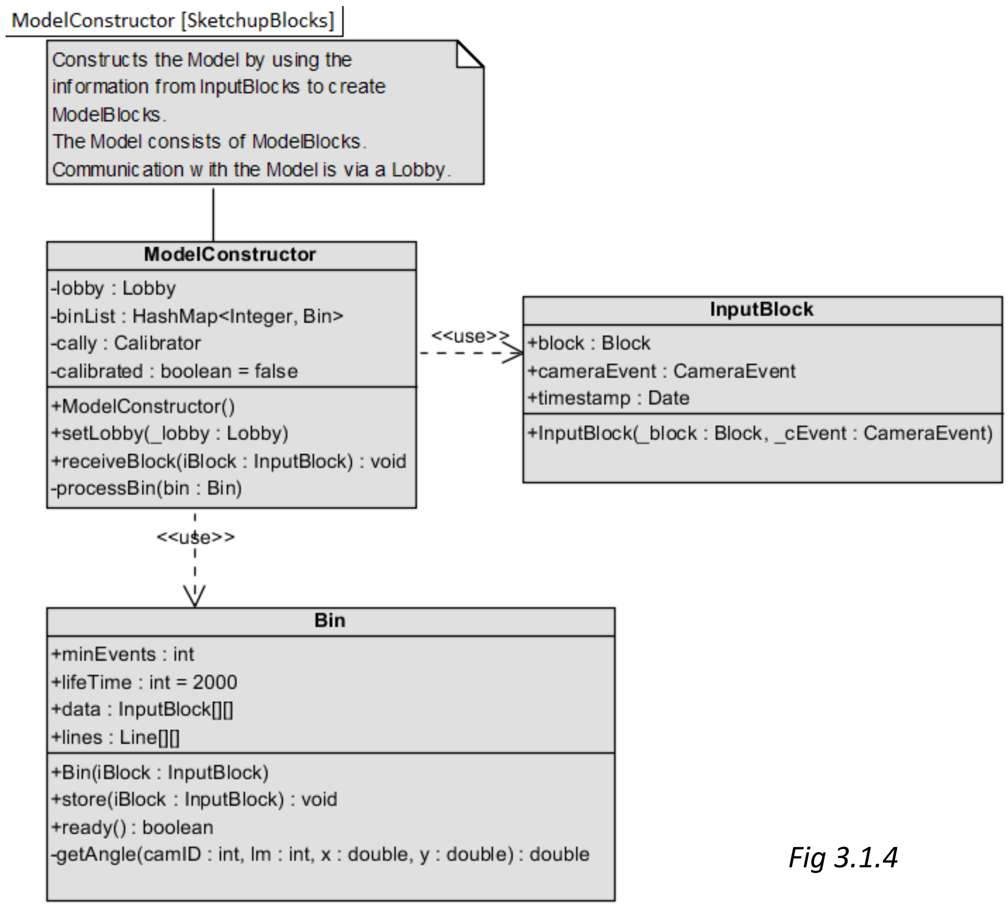

ModelConstructor [SketchupBlocks]
Constructs the Model by using the information from InputBlocks to create ModelBlocks. The Model consists of ModelBlocks. Communication with the Model is via a Lobby.
ModelConstructor
-lobby : Lobby
-binList : HashMap<Integer, Bin>
-cally : Calibrator
-calibrated : boolean = false
+ModelConstructor()
+setLobby(_lobby : Lobby)
+receiveBlock(iBlock : InputBlock) : void
-processBin(bin : Bin)
<<use>>
InputBlock
+block : Block
+cameraEvent : CameraEvent
+timestamp : Date
+InputBlock(_block : Block, _cEvent : CameraEvent)
<<use>>
Bin
+minEvents : int
+lifeTime : int = 2000
+data : InputBlock[][]
+lines : Line[][]
+Bin(iBlock : InputBlock)
+store(iBlock : InputBlock) : void
+ready() : boolean
-getAngle(camID : int, lm : int, x : double, y : double) : double

Fig 3.1.4

The construction pipeline as well as the data structures that it uses to communicate are shown below for clarity.

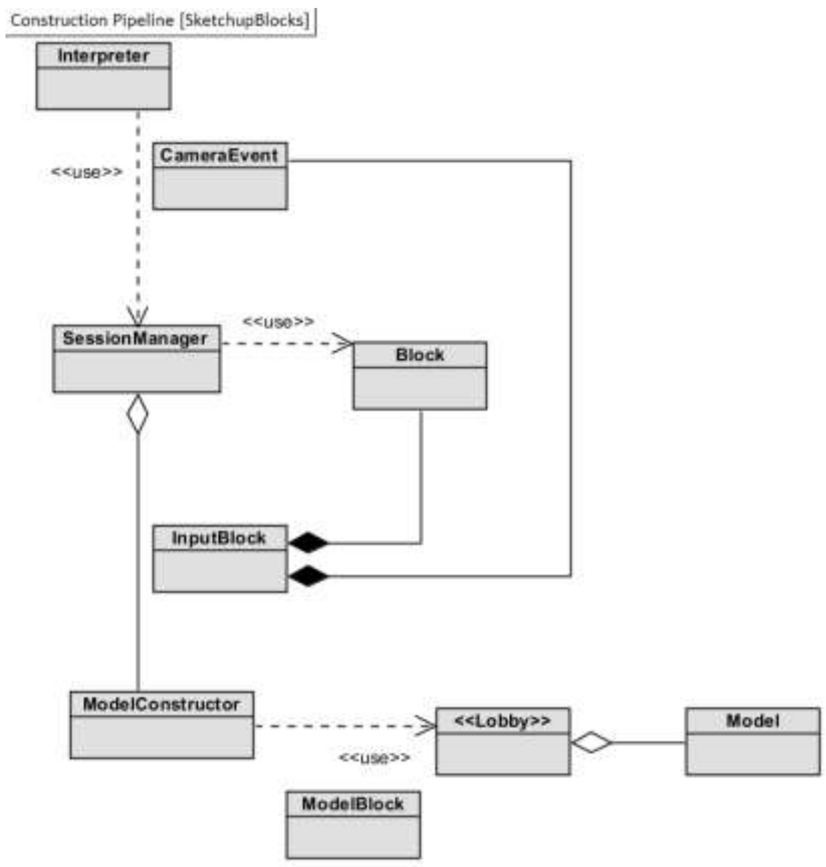


Fig 3.1.5

Note that the cameras require calibration, a process during which they determine their own positions in 3D space. The calibration process is done through a combination of Geometric Sphere Intersection methods and Particle Swarm Optimization, the details of which are also provided in the Technical Specifications.

As has been mentioned in previous documents, the blocks used by Sketchup Blocks can be divided into three categories:

- Smart blocks - used for construction
- Command blocks –used for menu interaction and camera calibration
- User blocks – used for collaboration

This heirarchy and the database in which they are stored can be seen below.

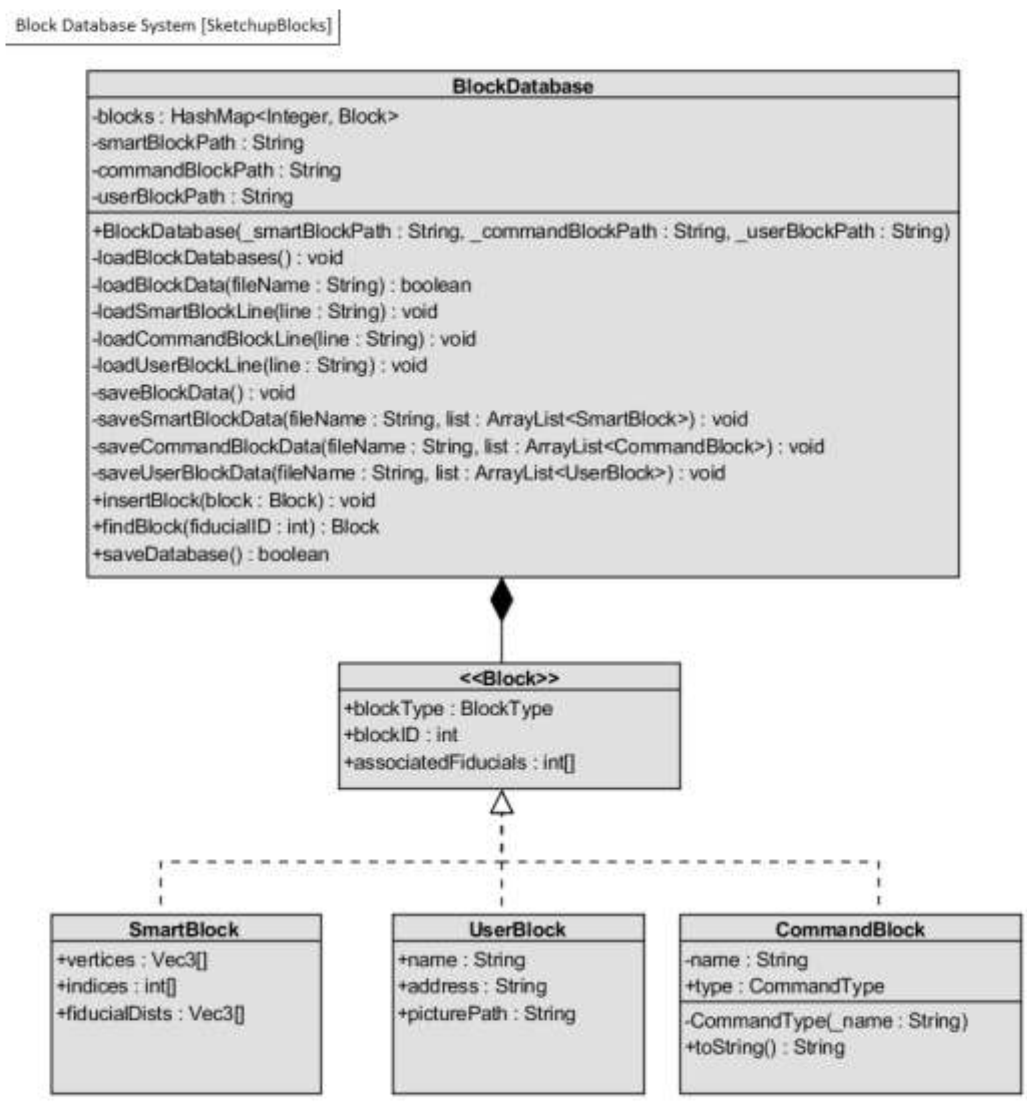
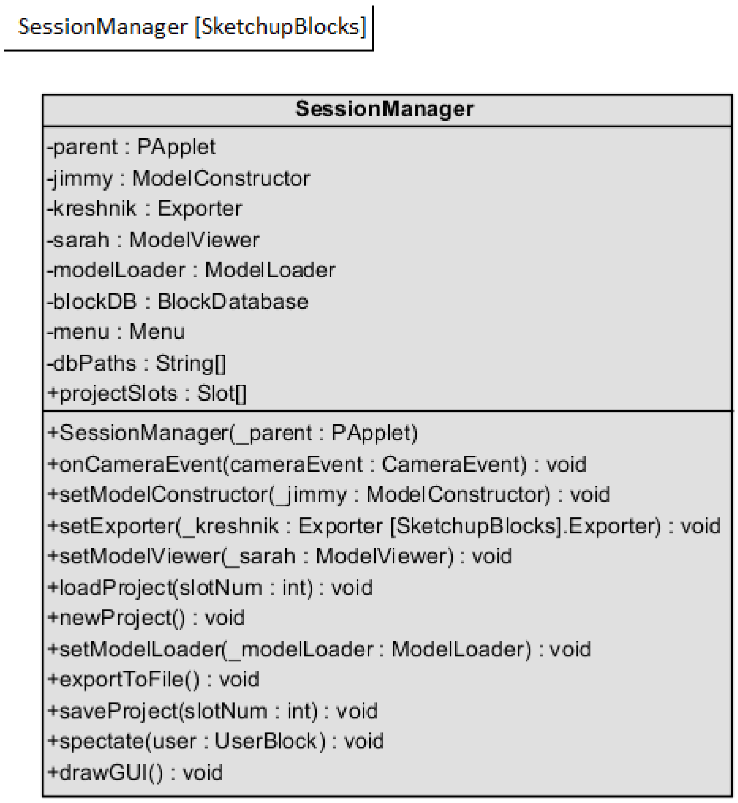

Block Database System [SketchupBlocks]
BlockDatabase
-blocks : HashMap<Integer, Block>
-smartBlockPath : String
-commandBlockPath : String
-userBlockPath : String
+BlockDatabase(_smartBlockPath : String, _commandBlockPath : String, _userBlockPath : String)
-loadBlockDatabases() : void
-loadBlockData(fileName : String) : boolean
-loadSmartBlockLine(line : String) : void
-loadCommandBlockLine(line : String) : void
-loadUserBlockLine(line : String) : void
-saveBlockData() : void
-saveSmartBlockData(fileName : String, list : ArrayList<SmartBlock>) : void
-saveCommandBlockData(fileName : String, list : ArrayList<CommandBlock>) : void
-saveUserBlockData(fileName : String, list : ArrayList<UserBlock>) : void
+insertBlock(block : Block) : void
+findBlock(fiducialID : int) : Block
+saveDatabase() : boolean
<<Block>>
+blockType : BlockType
+blockID : int
+associatedFiducials : int[]
SmartBlock
+vertices : Vec3[]
+indices : int[]
+fiducialDists : Vec3[]
UserBlock
+name : String
+address : String
+picturePath : String
CommandBlock
-name : String
+type : CommandType
-CommandType(_name : String)
+toString() : String

Fig 3.1.9

The role of the Session Manager will now be examined in greater detail.



```
SessionManager [SketchupBlocks]

SessionManager
-parent : PApplet
-jimmy : ModelConstructor
-kreshnik : Exporter
-sarah : ModelViewer
-modelLoader : ModelLoader
-blockDB : BlockDatabase
-menu : Menu
-dbPaths : String[]
+projectSlots : Slot[]
+SessionManager(_parent : PApplet)
+onCameraEvent(cameraEvent : CameraEvent) : void
+setModelConstructor(_jimmy : ModelConstructor) : void
+setExporter(_kreshnik : Exporter [SketchupBlocks].Exporter) : void
+setModelViewer(_sarah : ModelViewer) : void
+loadProject(slotNum : int) : void
+newProject() : void
+setModelLoader(_modelLoader : ModelLoader) : void
+exportToFile() : void
+saveProject(slotNum : int) : void
+spectate(user : UserBlock) : void
+drawGUI() : void
```

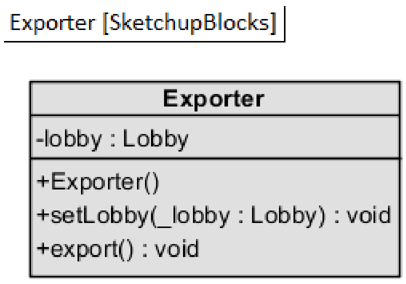
Fig 3.1.10

In addition to the construction of the Model, the Sketchup Blocks system must also perform other tasks such as exporting a model, viewing a model, project saving, project loading, etc.

The SessionManager keeps track of the classes that perform these functions and manages communication among them.

A brief discussion of these classes now follows.

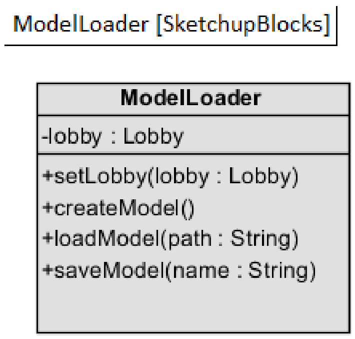
The Exporter class converts the Model to a Collada file, which can be viewed and edited in Google Sketchup.



```
Exporter [SketchupBlocks]

Exporter
-lobby : Lobby
+Exporter()
+setLobby(_lobby : Lobby) : void
+export() : void
```

Fig 3.1.11



```
ModelLoader [SketchupBlocks]

ModelLoader
-lobby : Lobby
+setLobby(lobby : Lobby)
+createModel()
+loadModel(path : String)
+saveModel(name : String)
```

Fig 3.1.12

The Model Loader allows the user to save and load projects via serialization. It assumes that the cameras and physical blocks have not been moved between saving and loading.

The Model Viewer displays a 3D view of the digital Model as it is constructed. Since it displays the Model in real-time, it must be notified of any changes that are made to the Model. Consequently, it implements the Model Change Listener Interface.

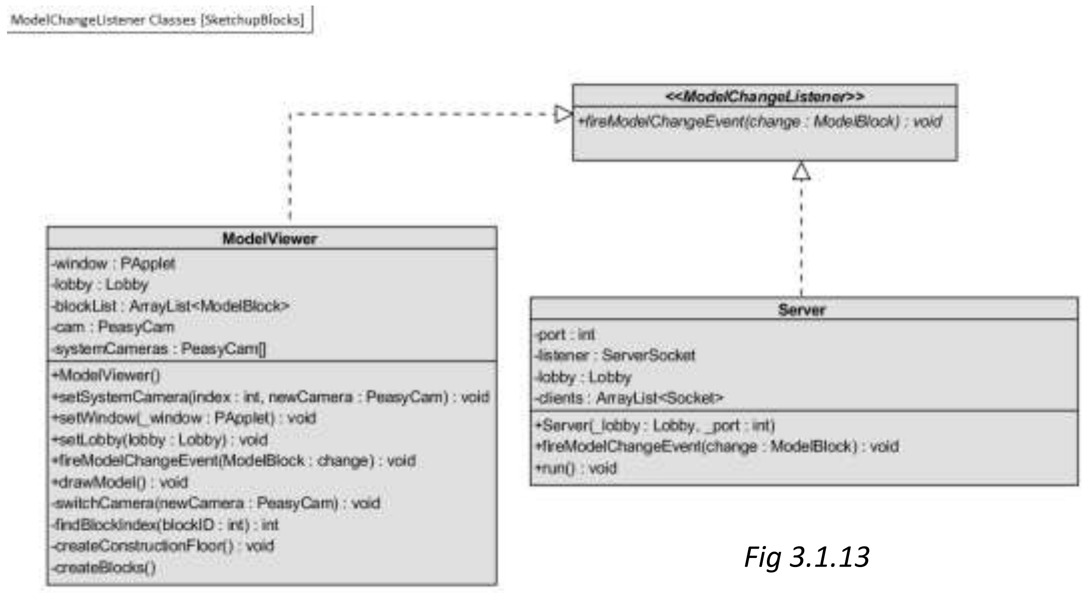
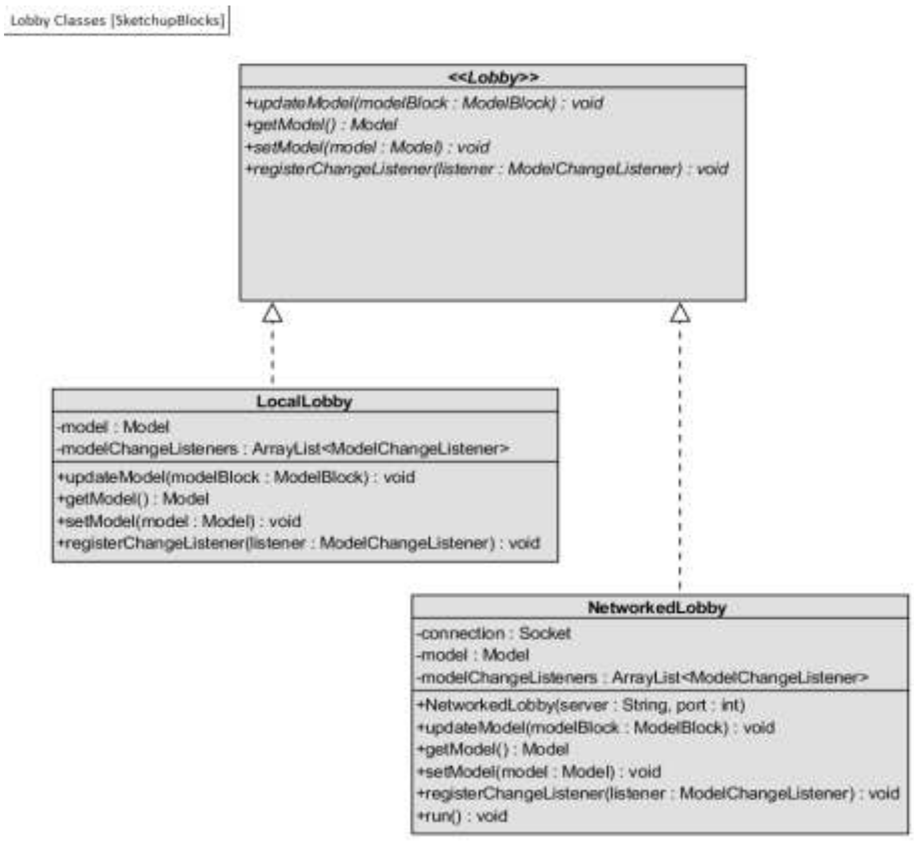

ModelChangeListener Classes [SketchupBlocks]
<<ModelChangeListener>>
+fireModelChangeEvent(change : ModelBlock) : void
ModelViewer
-window : PApplet
-lobby : Lobby
-blockList : ArrayList<ModelBlock>
-cam : PeasyCam
-systemCameras : PeasyCam[]
+ModelViewer()
+setSystemCamera(index : int, newCamera : PeasyCam) : void
+setWindow(_window : PApplet) : void
+setLobby(lobby : Lobby) : void
+fireModelChangeEvent(ModelBlock : change) : void
+drawModel() : void
-switchCamera(newCamera : PeasyCam) : void
-findBlockIndex(blockID : int) : int
-createConstructionFloor() : void
-createBlocks()
Server
-port : int
-listener : ServerSocket
-lobby : Lobby
-clients : ArrayList<Socket>
+Server(_lobby : Lobby, _port : int)
+fireModelChangeEvent(change : ModelBlock) : void
+run() : void

Fig 3.1.13

The optional release of the Sketchup Blocks project allows users to spectate other users' projects. This will require a specialized Lobby (namely a networking lobby) and a Server, which will broadcast changes of the Model to any users that are spectating the project.


Lobby Classes [SketchupBlocks]
<<Lobby>>
+updateModel(modelBlock : ModelBlock) : void
+getModel() : Model
+setModel(model : Model) : void
+registerChangeListener(listener : ModelChangeListener) : void
LocalLobby
-model : Model
-modelChangeListeners : ArrayList<ModelChangeListener>
+updateModel(modelBlock : ModelBlock) : void
+getModel() : Model
+setModel(model : Model) : void
+registerChangeListener(listener : ModelChangeListener) : void
NetworkedLobby
-connection : Socket
-model : Model
-modelChangeListeners : ArrayList<ModelChangeListener>
+NetworkedLobby(server : String, port : int)
+updateModel(modelBlock : ModelBlock) : void
+getModel() : Model
+setModel(model : Model) : void
+registerChangeListener(listener : ModelChangeListener) : void
+run() : void

3.2 PROCESS SPECIFICATION

The use cases for the Sketchup Blocks system are provided below. More use cases will be added as the system grows.

3.2.1 NEW PROJECT

SERVICES CONTRACT

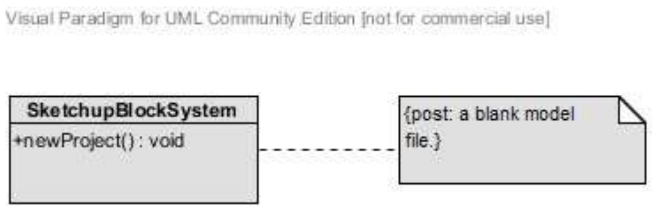

Visual Paradigm for UML Community Edition [not for commercial use]
SketchupBlockSystem
+newProject() : void
{post: a blank model file.}

Fig 3.2.1

FUNCTIONAL REQUIREMENTS

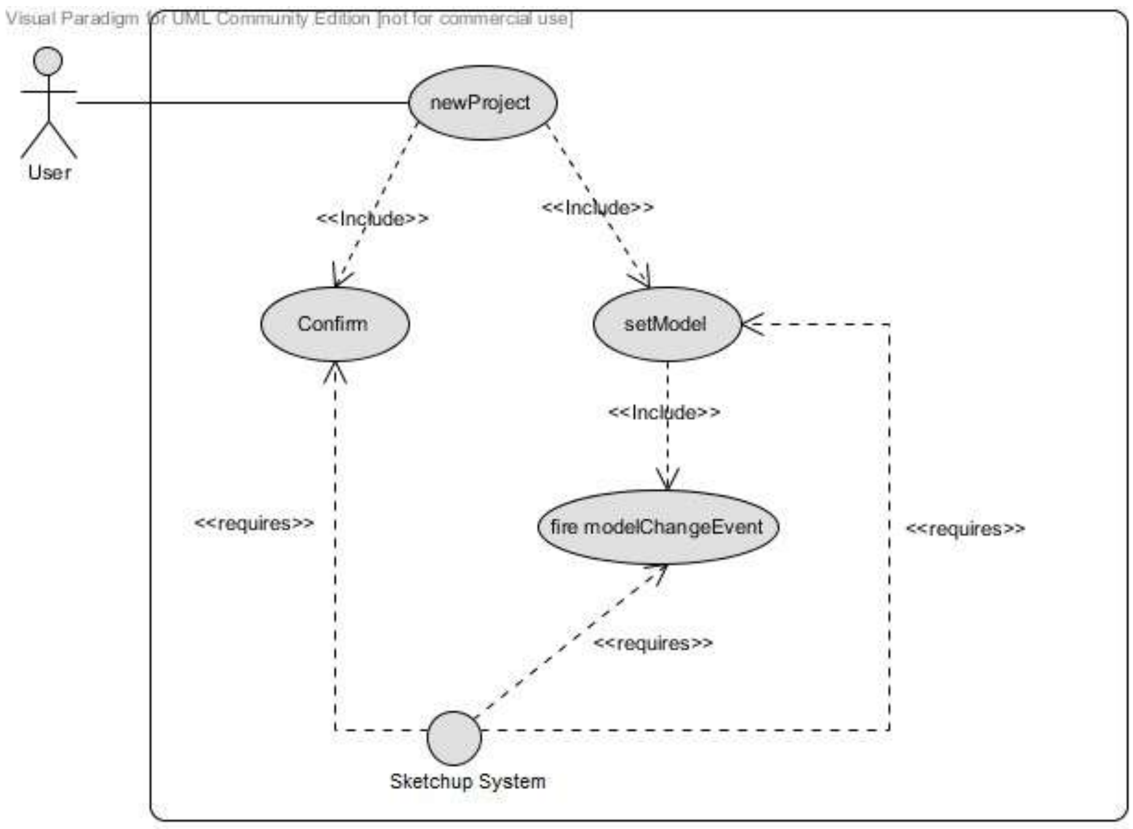


Fig 3.2.2

ACTIVITY DIAGRAM

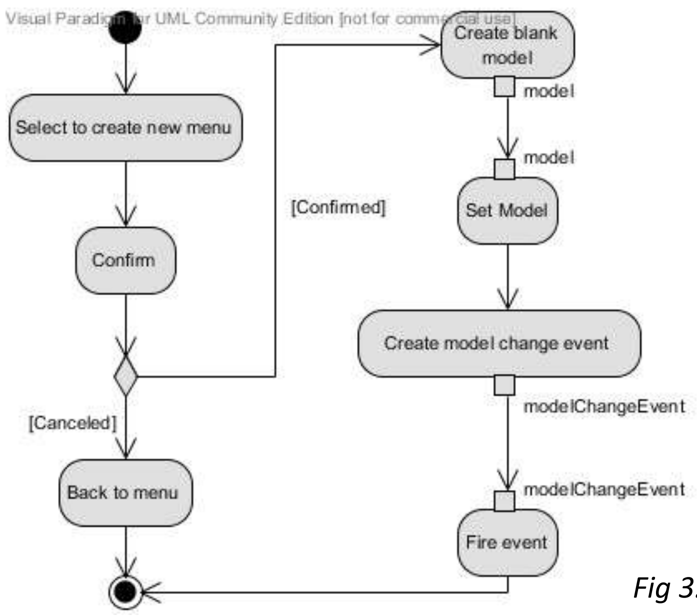
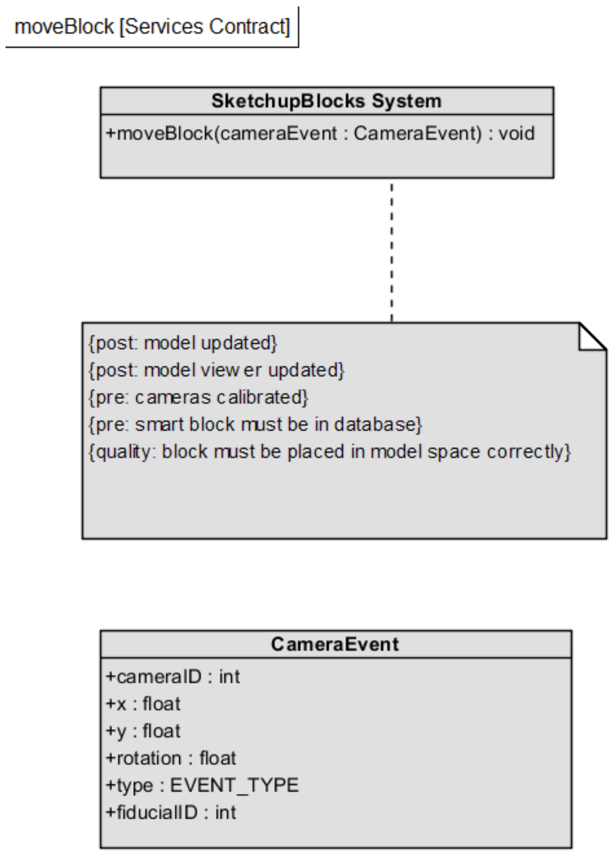


Fig 3.2.3

3.2.2 MOVE BLOCK

SERVICES CONTRACT



moveBlock [Services Contract]
SketchupBlocks System
+moveBlock(cameraEvent : CameraEvent) : void
{post: model updated}
{post: model viewer updated}
{pre: cameras calibrated}
{pre: smart block must be in database}
{quality: block must be placed in model space correctly}
CameraEvent
+cameraID : int
+x : float
+y : float
+rotation : float
+type : EVENT_TYPE
+fiducialID : int

Fig 3.2.4

FUNCTIONAL REQUIREMENTS

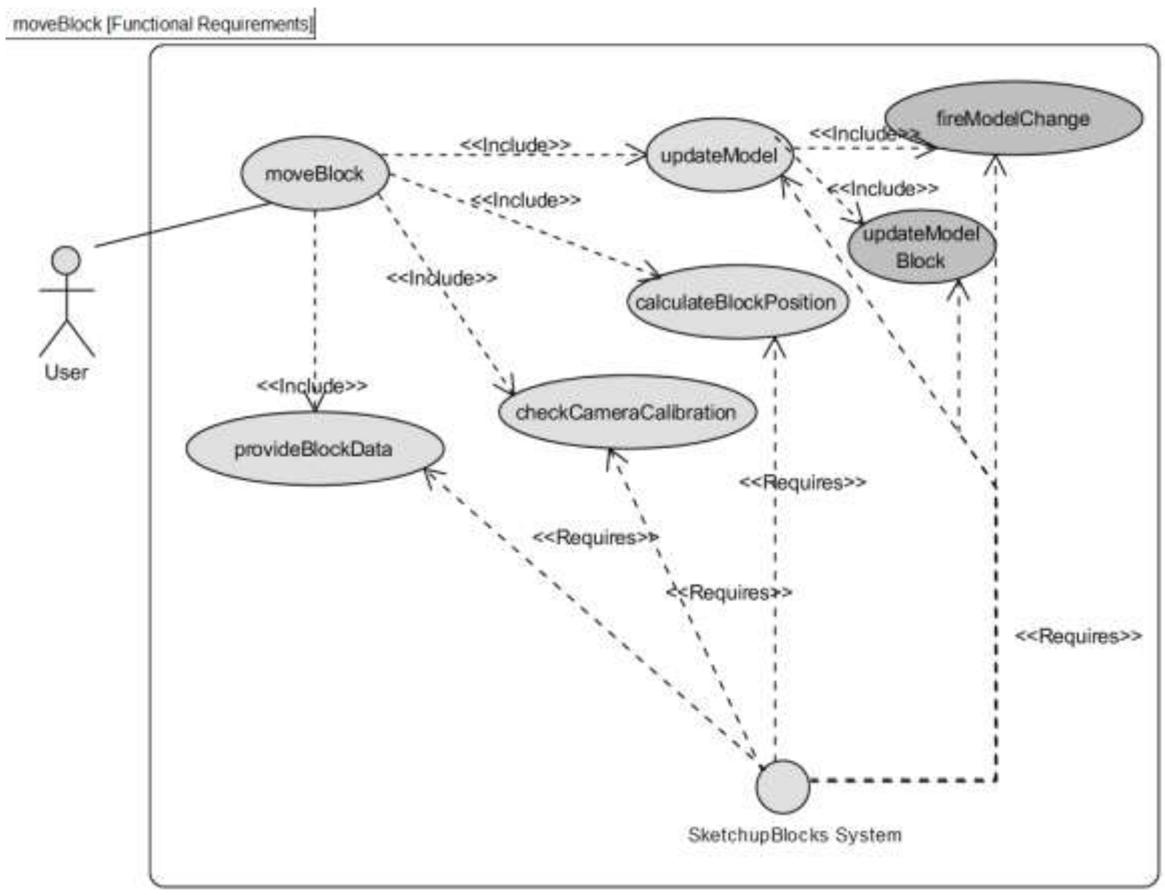


Fig 3.2.5

ACTIVITY DIAGRAM

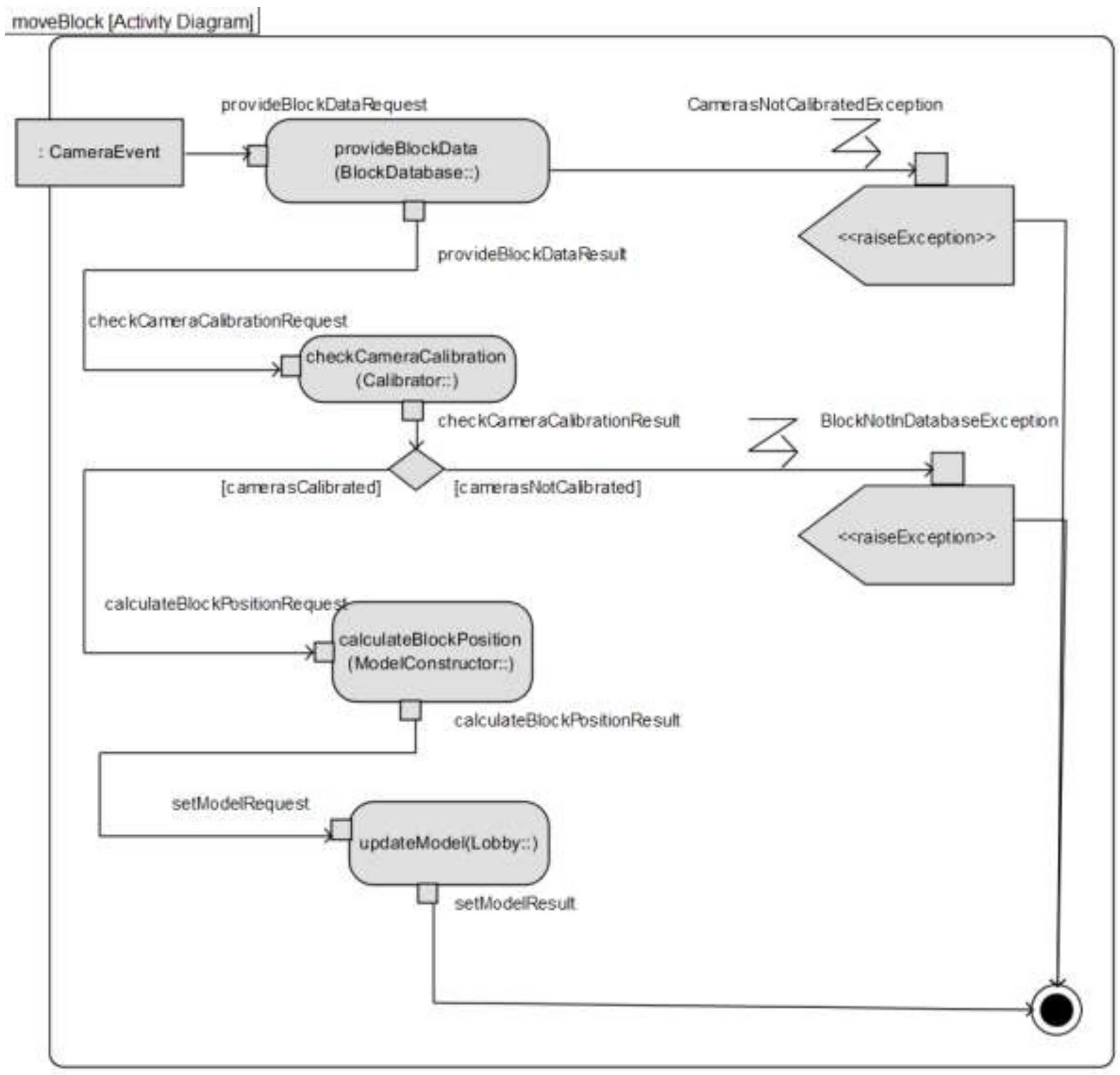

moveBlock [Activity Diagram]
: CameraEvent
provideBlockDataRequest
provideBlockData (BlockDatabase::)
CamerasNotCalibratedException
<<raiseException>>
provideBlockDataResult
checkCameraCalibrationRequest
checkCameraCalibration (Calibrator::)
checkCameraCalibrationResult
BlockNotInDatabaseException
[camerasCalibrated]
[camerasNotCalibrated]
<<raiseException>>
calculateBlockPositionRequest
calculateBlockPosition (ModelConstructor::)
calculateBlockPositionResult
setModelRequest
updateModel(Lobby::)
setModelResult

Fig 3.2.6

3.2.3 EXPORT MODEL

SERVICES CONTRACT

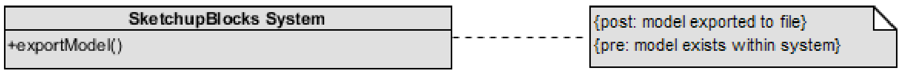

SketchupBlocks System
+exportModel()
{post: model exported to file}
{pre: model exists within system}

Fig 3.2.7

FUNCTIONAL REQUIREMENTS

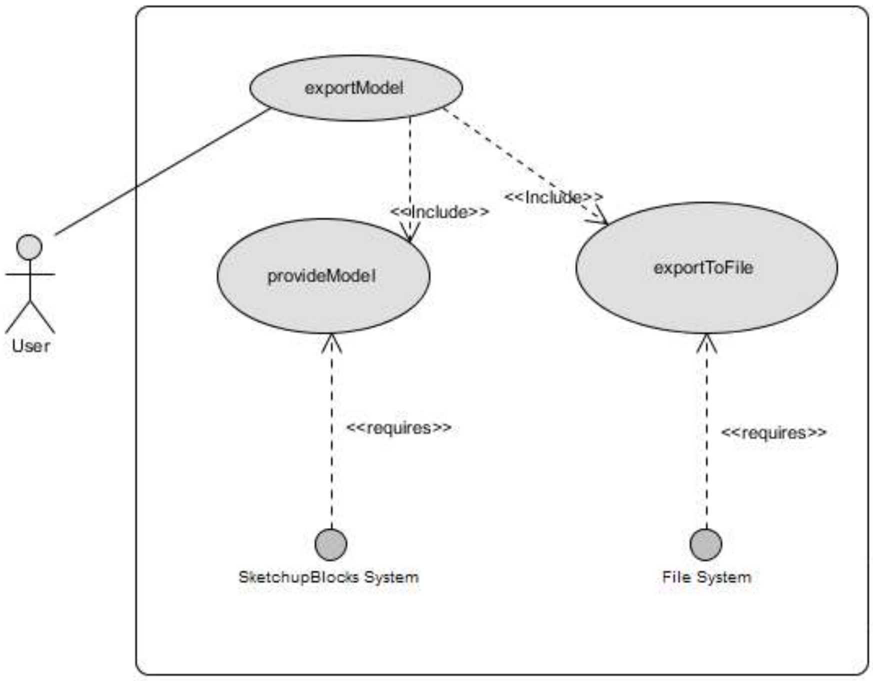


Fig 3.2.8

ACTIVITY DIAGRAM

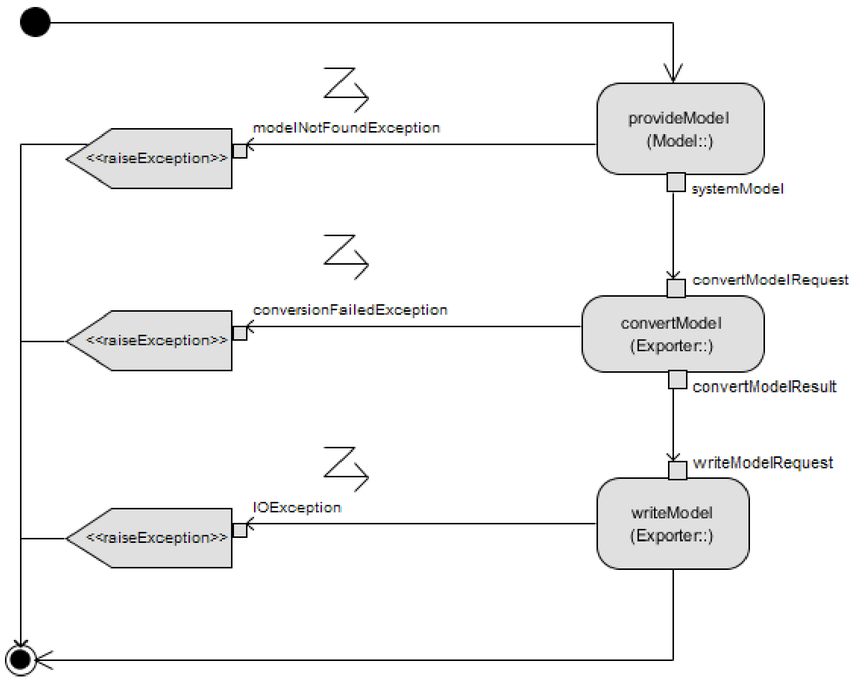


Fig 3.2.9

3.2.4 PROCESS CALIBRATION BLOCK

SERVICES CONTRACT

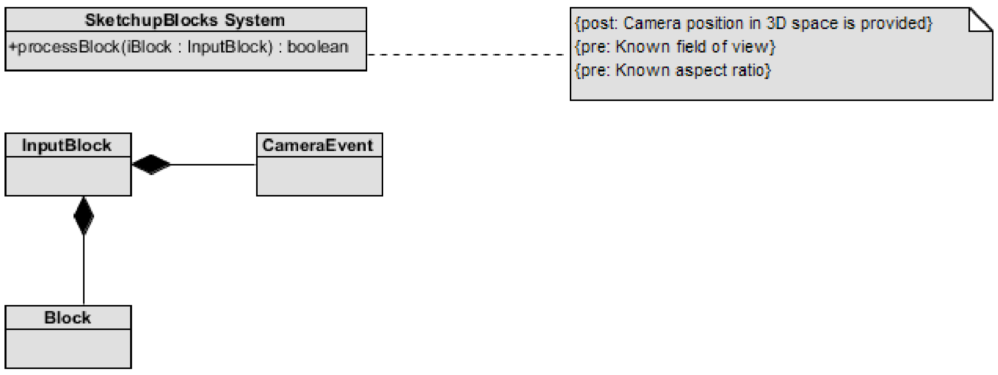

SketchupBlocks System
+processBlock(iBlock : InputBlock) : boolean
{post: Camera position in 3D space is provided}
{pre: Known field of view}
{pre: Known aspect ratio}
InputBlock
CameraEvent
Block

Fig 3.2.10

FUNCTIONAL REQUIREMENTS

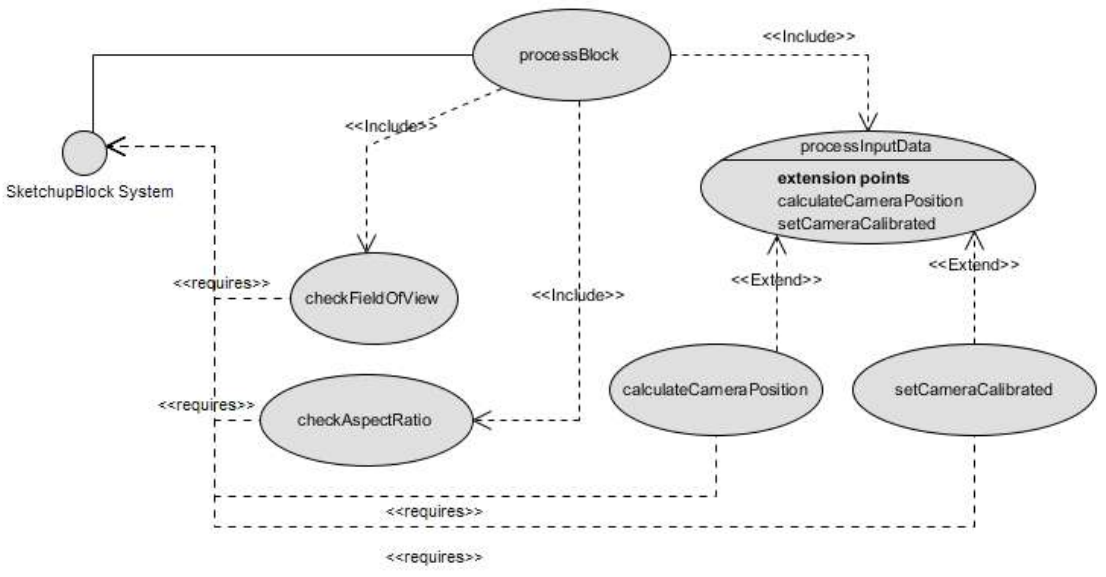


Fig 3.2.11

ACTIVITY DIAGRAM

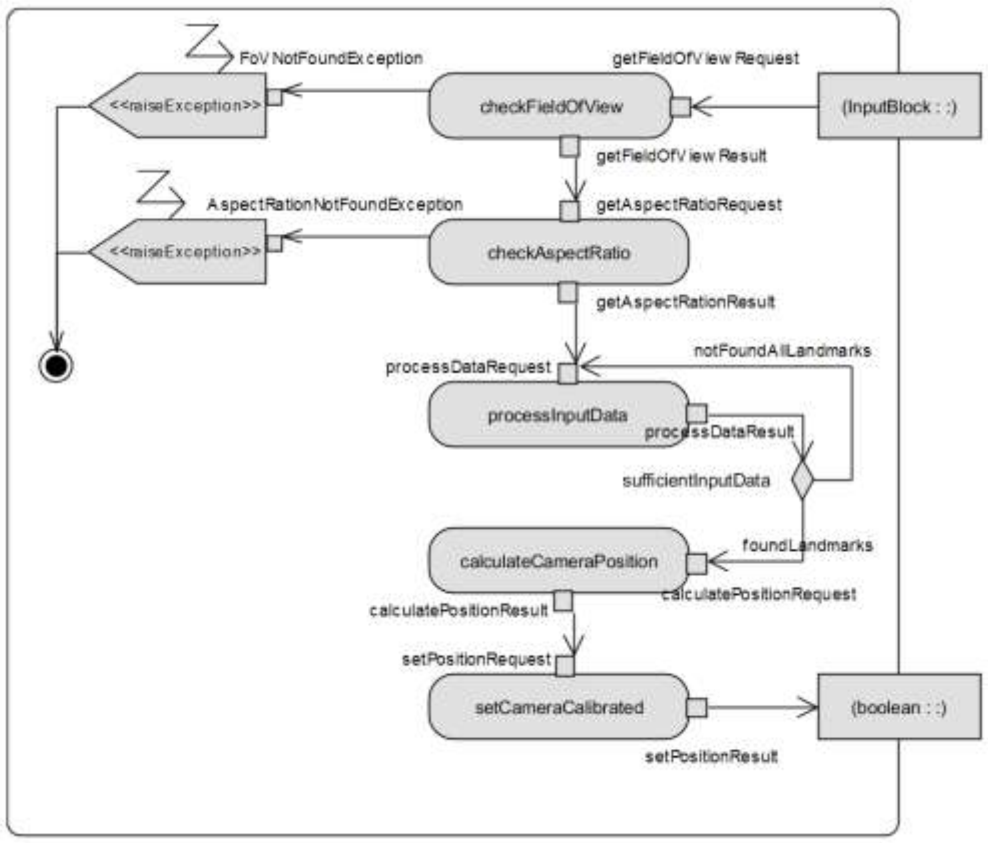


Fig 3.2.12

3.3 USER INTERFACE DESIGN

The current up to date user interface designs are shown below. This interface will be shown as an overlay over the 3D view. This interface will accept keyboard input as well as command block input.

3.3.1 GUI – CALIBRATING IN PROGRESS

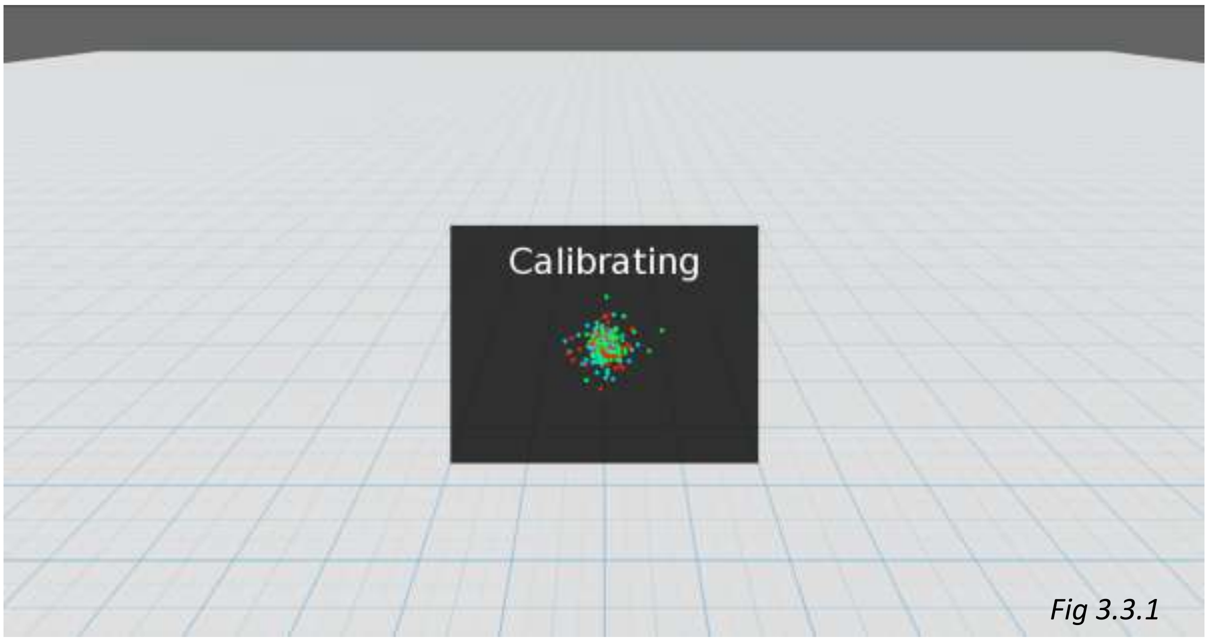


Fig 3.3.1

3.3.2 GUI– CALIBRATION DONE

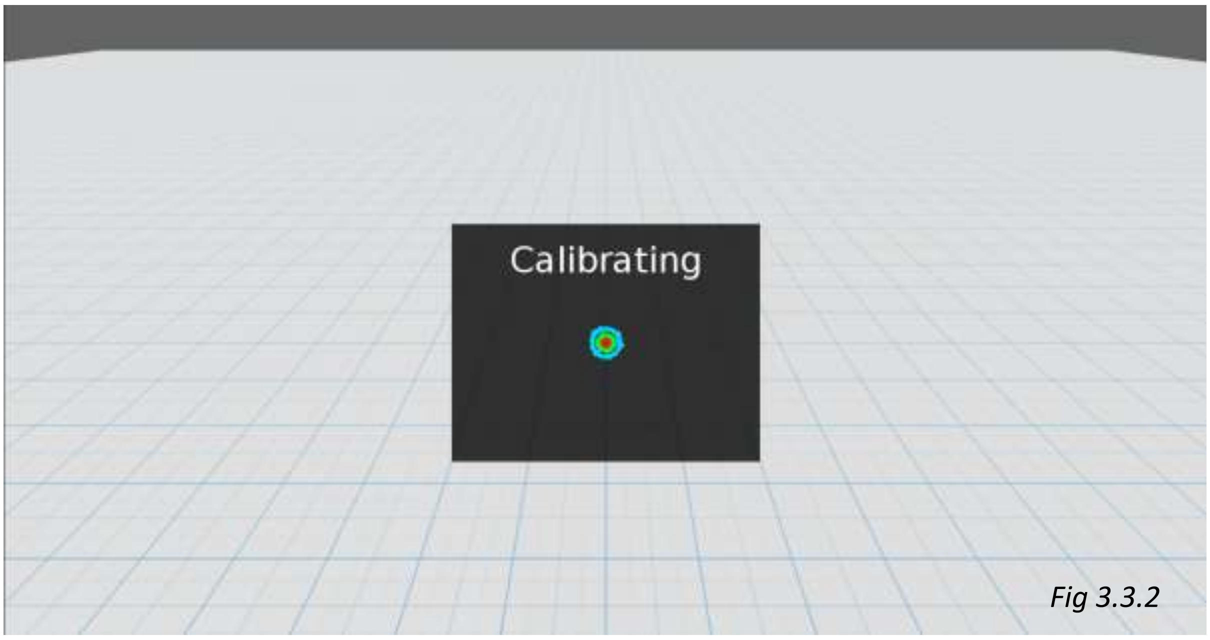


Fig 3.3.2

3.3.3 GUI - CONNECTING TO PEER

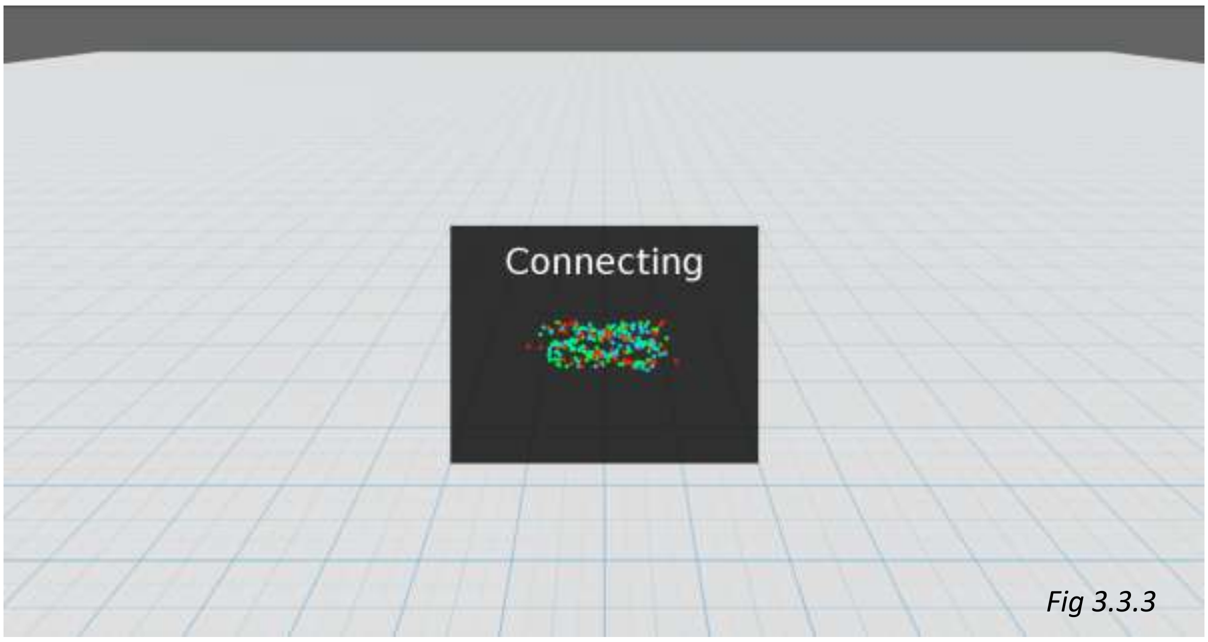


Fig 3.3.3

3.3.4 GUI - EXPORTING TO COLLADA FILE

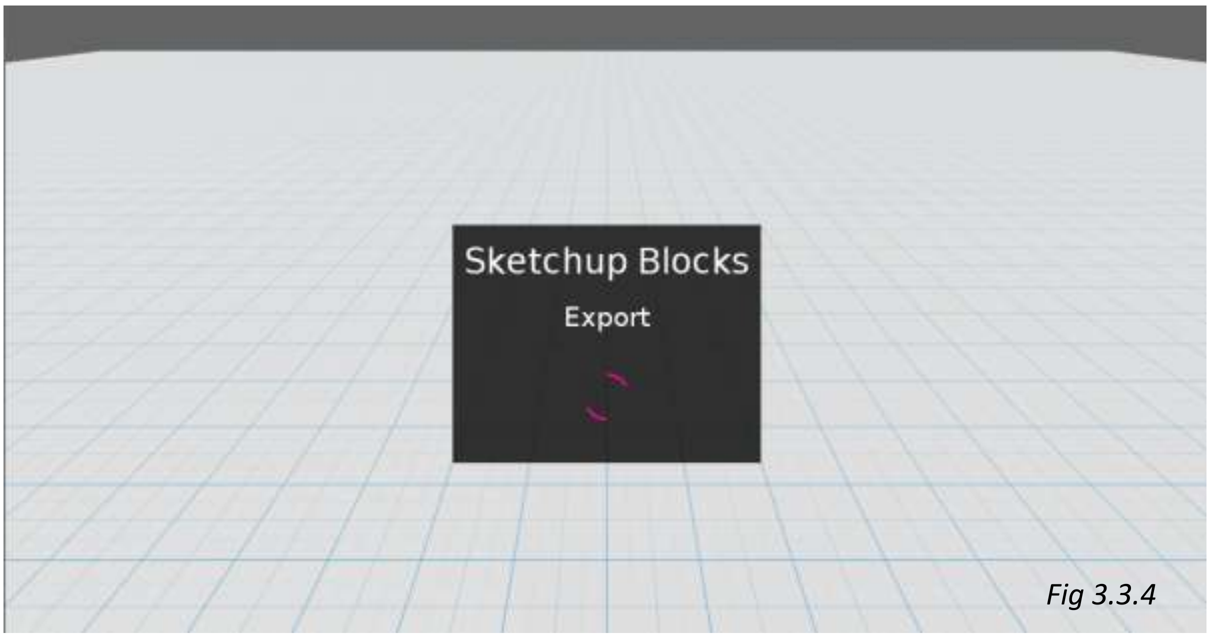


Fig 3.3.4

4. QUALITY ISSUES

The pursuit for high quality software requires adherence to the architectural strategies proposed as well as realization of the non-functional requirements of the project.

Architectural Strategies

- Caching
- First-In/First-Out

Non-Functional Requirements

- Throughput
- Portability
- Resilience
- Low Production Cost

5. CODING GUIDELINES

The coding guidelines to be followed closely resemble those provided by Oracle's for Java. Listed below are the more important points that will be adhered to as well as the important differences between the standards that will be used for Sketchup Blocks and those specified by Oracle.

Wrapping lines

- If an expression cannot fit on a single line break, the split should be made after a comma or before an operator.
- The new line should be aligned with the start of the expression.

Block comments

- Precede block comments with a blank line.
  Example:

```
/*
* Comments
* Comments
*/
```

Single line comments

- Precede comments with a blank line if on a line by itself.
- If comment is a trailing comment align with other comments to separate from the code.
  Example:

```
if(condition)
{

/* Comment */
      Statement;              /* This is a statement */
      AnotherStatement;       /*This is yet another statement*/
}
```

Declarations

- One declaration per line, thereby allowing for comments to follow.
- Declare variables at the beginning of a block with one line following declaration to separate from code (unless they are loop variables).

Method declarations

- Leave no space between the name and the parenthesis starting parameter list.
- Start the code block on a new line

Statements

- One statement per line.
- Return statements should not use parenthesis unless it improves understanding.

Statements of the form: if, if-else,if-else-if

- Always follow an if statement with braces.
  Examples:

```
if(condition)
{
	Statements;
}


if(condition)
{
	statements;
}
else
{
	moreStatements;
}


if(condition)
{
	statements;
}
else if(otherCondition)
{
	statements;
}
```

Indentation

- Indent each new scope block with one tab.

Scope blocks

- The starting brace of a block should be a new line.
- The closing brace should be aligned with the opening brace.

6. REQUIREMENT MATRICES

6.1 REQUIREMENT IDENTIFIERS

FUNCTIONAL REQUIREMENTS

REQUIREMENT	DESCRIPTION	SOURCE	PRIORITY
FR1	Cubic/Simple shapes	Andrew Smith	High
FR2	3D Model	Andrew Smith	High
FR3	Sketchup Integration	Andrew Smith	High
FR4	Auto-detected block addition	Andrew Smith	High
FR5	Auto-detected block removal	Andrew Smith	High
FR6	More complex shapes	Andrew Smith	Medium
FR7	Real-Time Display	Andrew Smith	Medium
FR8	Network Interaction	Andrew Smith	Low
FR9	Custom Shapes	Andrew Smith	Low

NONFUNCTIONAL REQUIREMENTS

REQUIREMENT	DESCRIPTION	SOURCE	PRIORITY
NFR1	Throughput	Andrew Smith	High
NFR2	Portability	Andrew Smith	Medium
NFR3	Resilience	Andrew Smith	Medium
NFR4	Low production cost	Andrew Smith	Low

6.2 REQUIREMENTS TRACEABILITY MATRIX

TRACEABILITY MATRIX – FUNCTIONAL REQUIREMENTS

	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9
Vision & Scope	X	X	X	X	X	X	X	X	X
Architectural Requirements	X	X	X			X	X	X	X
BlockDatabase Test	X	X				X			X
Calibration Test		X					X		
Evaluator Test		X							
Exporter Test			X						
Interpreter Test				X	X				
LinearSolver Test		X							
Local Lobby Test		X					X	X	
Matrix Test		X							
Menu Test	X						X	X	
Model Viewer Test							X		
Networked Lobby Test								X	
Particle Creator Test		X							
Server Test								X	
Vec3 Test		X					X		
Vec4 Test		X					X		
Done				X	X				
Acceptance Test 1									
Accepted									

TRACEABILITY MATRIX – NONFUNCTIONAL REQUIREMENTS

	NFR1	NFR2	NFR3	NFR4
Vision & Scope	X	X	X	X
Architectural Requirements	X	X	X	X
BlockDatabase Test				
Calibration Test	X	X	X	
Evaluator Test	X		X	
Exporter Test				
Interpreter Test			X	
LinearSolver Test	X	X	X	
Local Lobby Test				
Matrix Test				
Menu Test				
Model Viewer Test				
Networked Lobby Test				
Particle Creator Test	X	X	X	
Server Test				
Vec3 Test				
Vec4 Test				
Done	X			X
Acceptance Test 1				
Accepted				

7. GLOSSARY

Collada	Interchange file format for interactive 3D applications that is supported by Google Sketchup.
Command block	A smart block used to control the system (as opposed to model construction).
Construction floor	The flat surface on which the physical model is built and around which the web cameras are erected.
Fiducial markers	Small markers that will be attached to the physical blocks and used to identify them.
Google Sketchup	3D modelling program.
IoT	Internet of Things
MVC	Model-View-Controller, an architectural pattern
Object file	The Collada file that contains all the data of a particular model
ReacTIVision	Open source, cross-platform computer vision framework for tracking fiducial markers that are attached to physical objects.
Smart block	A physical block with an attached fiducial marker.
TUIO	A protocol that allows the encoding and transmission of a tangible interface component abstraction, such as tokens, pointers and geometries. This is the protocol by which ReacTIVision will communicate with the Sketchup Blocks system.