

Siftables Emulator
Singularity Software

January 27, 2012

Alex Mullans
Ethan Veatch
Eric Vernon
Kurtis Zimmerman

1 Domain Model

No updates were made to the domain model because it was initially based on the Sifteo API which has not (and will not) undergo changes over the course of the project.

2 Operation Contracts

No changes were made to the operations contracts because no modifications were made to the way in which the operations will be implemented nor the expectations we have for their effects on the system.

OC1: SelectFileAndClickOpen

Operation	SelectFileAndClickOpen(filePath : String)
Cross-references	UC1: Load program, UC2: Reload program
Preconditions	The OpenFileDialog is open.
Post-conditions	The file name was parsed. The emulator opened the file.

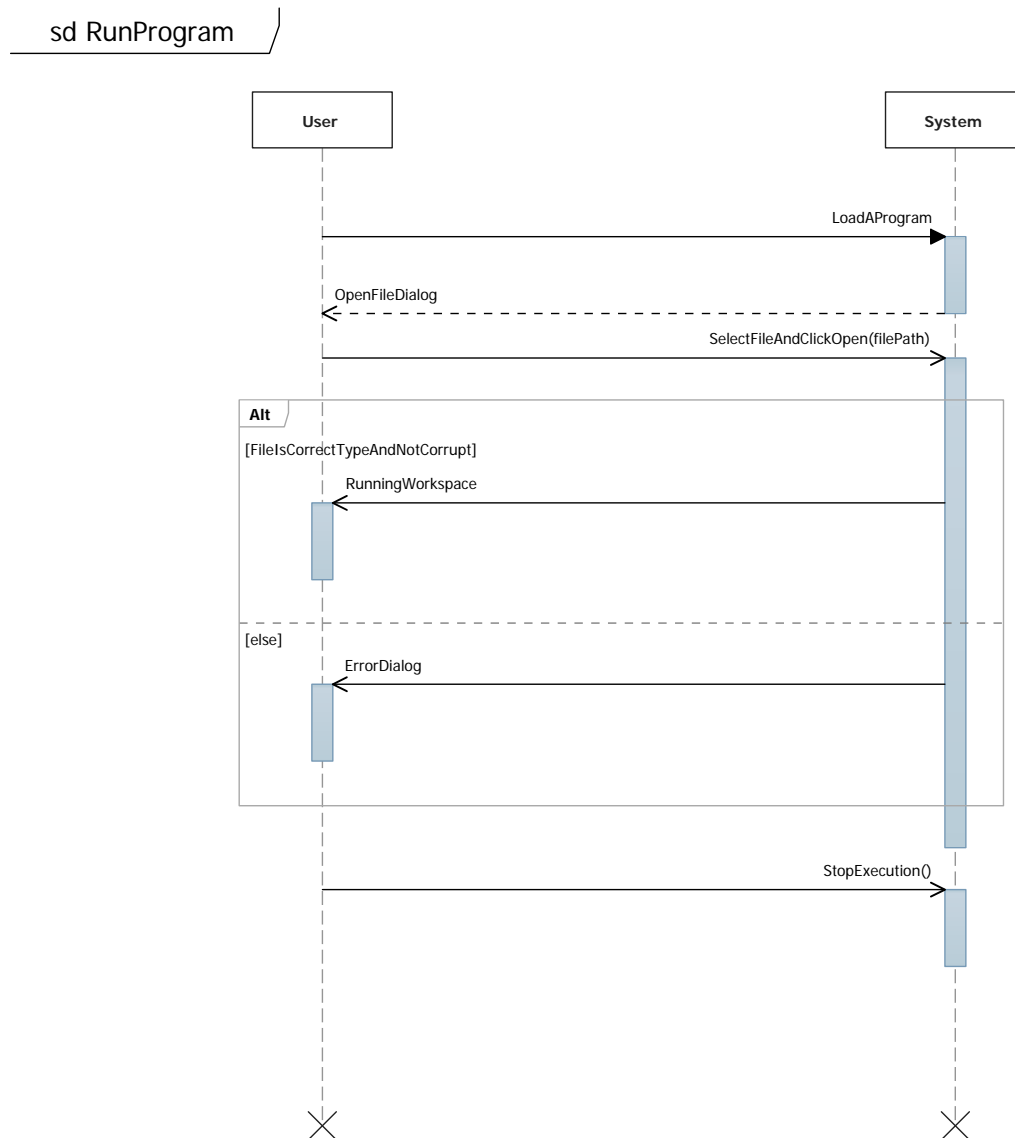
OC2: ZoomSliderChanged

Operation	ZoomSliderChanged(zoomLevel : int)
Cross-references	UC3: Zoom screen
Preconditions	There is an application running in the workspace.
Post-conditions	The workspace canvas has been magnified appropriately. The workspace zoomLevel attribute was updated.

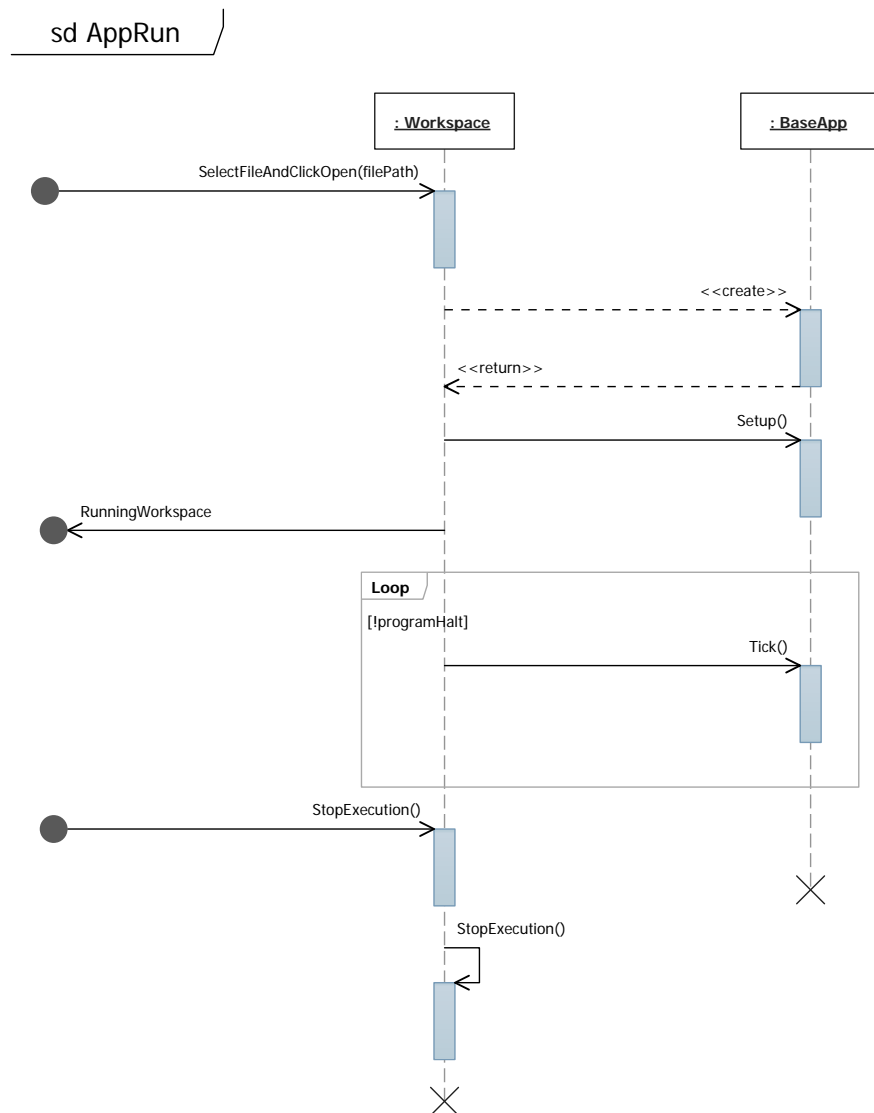
Additional operation contracts for the remaining use cases were not pursued because of their trivial nature. The basic format of this operation contract — user changes UI element, UI adjusts accordingly, and program updates relevant attributes — applies to the other use cases, which has not changed.

3 System Sequence Diagram

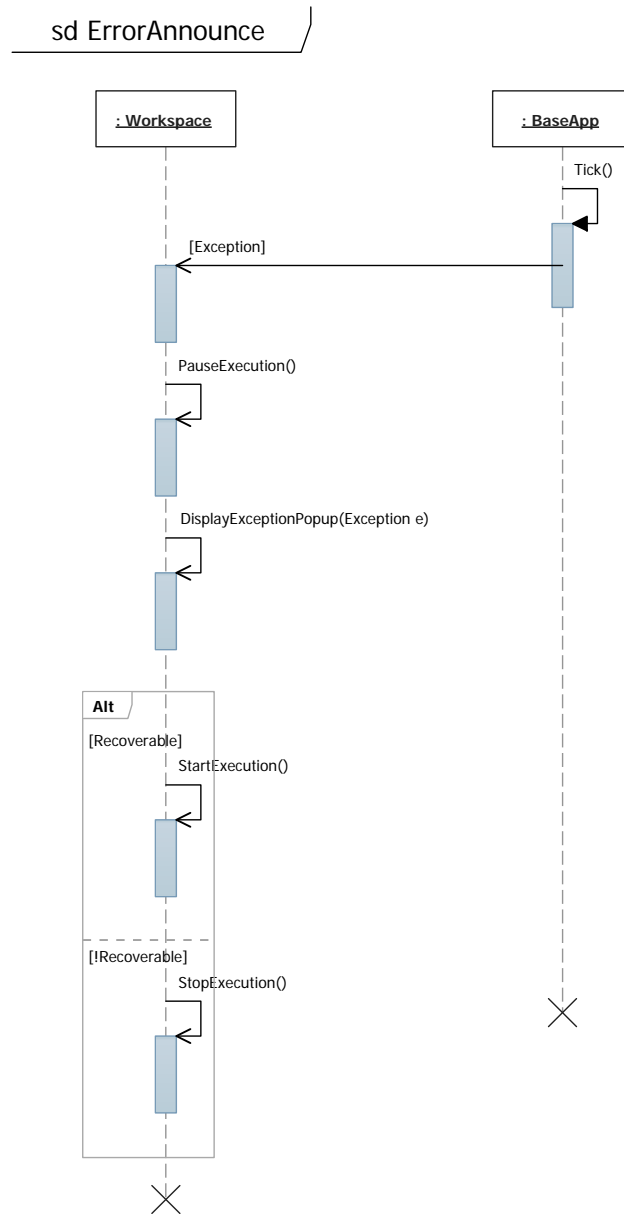
The initial RunProgram system sequence diagram had a duplicate step which was annihilated by replacing the *FileOpened* result with the *RunningWorkspace* result because the two were essentially duplicates.



4 Sequence Diagrams



The *Setup()* operation is necessary in addition to a constructor because this is the way in which initialization of an application is handled per Sifteo's API. No changes were made to this sequence diagram because it is consistent with the way in which the system behaves during this operation.



The *Tick()* operation was unintentionally omitted from the original diagram, and it has been added for clarity to indicate that the action is taking place during an actively running application.

5 Design Class Diagram

Any external relationships are shown beneath the class name to emphasize interaction with classes we did not create. For instance, each *Cube* is implemented as a *UserControl* object.

Additionally, every entity in our diagram is being implemented in our own original code, though many classes are inspired by the Sifteo API which we have to mirror. Those entities which do mirror the objects in the Sifteo domain will be separated into a distinct package to clarify the divide between those objects which carry out the emulation and those which the user primarily manipulates in their original applications.

Siftables

Class

→ Application

public

Siftables()

private

Application_Exit(object sender, EventArgs e) : void

Application_Startup(object sender, StartupEventArgs e) : void

Application_UnhandledException(object sender, ApplicationUnhandledExceptionEventArgs e) : void

ReportErrorToDOM(ApplicationUnhandledExceptionEventArgs e) : void

BaseApp

Class

public

BaseApp(List<Cube> cubes)

FrameRate : int

Setup() : void

Tick() : void

private

Cubes : List<Cube>

MainWindow

Class

→ UserControl

public

MainWindow()

private

cubes : CubeSet

loadAProgramButton_Click(object sender, RoutedEventArgs e) : void

MainWindow_Loaded(object sender, RoutedEventArgs e) : void

numberOfCubesUpDown_ValueChanged(object sender, RoutedProp...

snapToGridButton_Click(object sender, RoutedEventArgs e) : void

Cube

Class

→ UserControl

public

Cube()

dimension : int

FillRect(Color c, int x, int y, int w, int h) : void

FillScreen(Color c) : void

Image(string name, [int x = 0], [int y = 0], [int s...

SCREEN_HEIGHT : int

SCREEN_MAX_X : int

SCREEN_MAX_Y : int

SCREEN_MIN_X : int

SCREEN_MIN_Y : int

SCREEN_WIDTH : int

Nested Types

Side

Enum

Top

Right

Bottom

Left

None

CubeSet

Class

→ List<Cube>

public

AddCubes(int count) : void

CubeSet(Canvas workspace)

RemoveCubes(int count) : vo...

private

workspace : Canvas

Neighbors

Class

public

Bottom { get; set; } : Cube

Contains(Cube c) : bool

Count { get; } : int

CubeOnSide(Side s) : Cube

IsEmpty { get; } : bool

Left { get; set; } : Cube

Neighbors()

Right { get; set; } : Cube

SideOf(Cube c) : Side

Top { get; set; } : Cube

private

_neighbors : Cube[]

_numNeighbors : int

sideUtil(Side s, Cube value) : void

DragAndDropBehavior

Class

→ Behavior<UIElement>

protected

OnAttached() : void

OnDetaching() : void

private

_isDragging : bool

AssociatedObject_MouseLeftButtonDown(object sender, MouseButtonEventArgs e) : void

AssociatedObject_MouseLeftButtonUp(object sender, MouseButtonEventArgs e) : void

AssociatedObject_MouseMove(object sender, MouseEventArgs e) : void

DragAndDropBehavior_MouseEnter(object sender, MouseEventArgs e) : void

DragAndDropBehavior_MouseLeave(object sender, MouseEventArgs e) : void

mouseClickPosition : Point

parent : DependencyObject

This is a generic class that represents any application developed for the emulator or the actual cube platform.

Note: Because the class diagram was generated by Visual Studio, no associations were created (for instance, between the entity *Cube* which is associated with a unique *Neighbors* entity). We feel no clarity was lost by omitting these associations.

6 Applications of the GRASP Principles

6.1 Creator

MainWindow creates each of the cubes because it has the information (location in the space) necessary to construct them. Also, the *Siftables* entity is responsible for creating *MainWindow* because it acts as the starting point for the emulator.

6.2 Information Expert

Because each *Cube* object is the sole owner of the information (in this case, a collection of graphical objects) which are contained on its screen, it is responsible for conveying this in the workspace onto the graphical representation of its screen.

6.3 Controller

There is no concrete example of a controller in our design, but *BaseApp* acts like a use case controller because it is in charge of handling interactions between domain-level objects and the user interface (on the *Cube* screens). Though we are not sure which entity will have this responsibility, there will also be a controller (or dispatcher) to begin the emulation of an application and handle the calls to *Tick()* which will subsequently cause changes to the interface.

6.4 Low Coupling

BaseApp only connects to the UI layer simply through the *Cube* entities because it is responsible for controlling events related to the cubes and must be responsible for carrying out the steps specified in each application. This reduces coupling between the two logical layers and allows the domain layer to be implemented more independently from the interface layer.

6.5 Pure Fabrication

The *Neighbors* entity does not exist in the physical domain, but it is useful in the implementation to separate the responsibility of keeping track of a cube's neighbors and their interactions from the graphical representation of the cube. The *CubeSet* class represents the set of cubes which exist in the current workspace, which is maintained as a separate entity to allow events to be defined on the entire set of cubes. For instance, an event can be triggered when a cube is added or removed from the set, which could not easily be handled if a set of cubes was implemented simply as a data structure.

6.6 High Cohesion

The *Cube*, *Neighbors*, and *CubeSet* class all emphasize high cohesion by abstracting out specific, distinct responsibilities to separate entities. The *Cube* acts primarily as a graphical interface to the cube objects, while *Neighbors* has the responsibility of maintaining neighborhood relationships between *Cube* objects. The *CubeSet* responsibilities are mentioned above.

6.7 Protected Variations

While our system does not directly interact with third-party interfaces, and therefore does not directly use the Protected Variations pattern, the project as a whole is itself a Protected Variation that implements the Sifteo API. An overriding goal of our design going forward will be to remain fully compatible with that API so that applications written for the actual Sifteo Cube platform can also be run on our emulator.

Note: Because our system does not interact with third-party interfaces, there are not clear examples of the **Indirection** pattern.