Software Design Document

for

Satellite Attitude and Orbit Visualization

Version 1.2 approved

Prepared by Rolf Castro, Miguel Cayetano, Alan Daniel, Victor Orozco, and Allen Ma

Boeing / Sen Yao

March 7, 2019

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Alan Daniel	11/08/2018	Initial documentation	0.0
Rolf Castro	11/08/2018	Reformatting, adding DFDs, and Resources	0.1
Alan Daniel	11/09/2018	Introduction subsections	0.2
Rolf Castro	11/11/2018	Adding Architectural Strategies	0.3
Alan Daniel	11/12/2018	Drafted Policies and Tactics	0.3
Rolf Castro	02/04/2019	Insertion of Subroutines to DDM, TSM, and SCM	0.4
Rolf Castro	03/01/2019	Completing of System and Lower Level Component Design	0.5
Alan Daniel	03/01/2019	Finishing Database Design and and User Interface sections	0.6
Alan Daniel	03/02/2019	Completing User Interface section, Requirements Validation and Verification, Glossary, and References sections	1.0
Rolf Castro	03/04/2019	Modification of User Interface to hide upload button when no file is in the dropbox, reformat document, and update Table of Contents	1.1
Rolf Castro	03/07/2019	Reformatting document, finishing up subroutines in TSM and SCM	1.2

1. Introduction

1.1 Purpose

The purpose of this project is to deliver a software that illustrates up to three satellite trajectories with respect to the Earth. The software will also include client input to upload two-line element files via ‘drag and drop’ and browse functionalities. The end result will serve as a web application for Boeing to track the current position of their satellites in real-time.

1.2 Document Convention

The convention of this document starts from the high-level requirements and explains more into detail of each system and lower level component design. This will serve as a manual as to how the Satellite Attitude and Orbit Visualization (SAOV) works. Since the high-level requirements will be discussed in the beginning of the document and the lower-level requirements are at the bottom of the document, the reader will require to do some jumping around to understand how a specific component interacts/exports/etc. with another component. This document is not a user guide, it does not describe how to use the SAOV when ran locally (On the client’s computer) or on a server. That is in the “SAOV User Guide” documentation.

1.3 Intended Audience and Reading Suggestions

This document is intended for the Boeing developers that will receive our final product, advisers to review our documentation, and the Computer Science Department at California State University, Los Angeles. The rest of this document will contain architectural details, designs, and goals for our project within the next six months.

1.4 System Overview

The software system is a web application that illustrates the position of any satellite at a given time. Controls within the CesiumJS API allow the client to zoom in and out of the display, speedup the orbit, add multiple satellites, and overwrite older satellite two-line element files. The design approach behind the software system includes web programming elements such as websockets in order to communicate to and from the server the site is hosted on. The system is written in JavaScript.

2. Design Considerations

2.1 Assumptions and Dependencies

- This software will assume the user has a machine with WebGL, which is capable of rendering interactive 2D and 3D imagery. Most machines have this.

- Regardless of WebGL, the web application will assume that the user has hardware capable of running the web application for extended periods of time.
- The software will assume the client has a stable internet connection, the software cannot run offline as it is powered by Bing Maps.
- The software will assume the client has the necessary files (two-line element files) to upload, which is necessary for satellite visualization.

2.2 General Constraints

- After testing the number of visual orbits that can be displayed on the development team's computers, it has been determined that 3 satellite orbits would be the optimal view amount as the web application is resource-heavy. Although if a client's computer is powerful enough they may view more than 3 orbits at a time.
- Currently works only on Safari version 12.0.1, Google Chrome version 70.0.3538.77 Internet Browsers, and Microsoft Edge version version 17.17134.
- Once a client has exited or refreshed their web application page they create a new instance and reset their view of the web application.
- After 5 TLE uploads the clients' view of the application becomes sluggish due to the large CZMLs being sent the to client from the server.
- In order to have a proper view of the web application and successful uploads of TLE files the clients must have a stable Internet connection.
- Due to the Scrum methodology that our team utilized for this project, issues arose as quickly as implementations were added into the application. Much of our verification and validation came from simply printing out errors, code tracing to the last working line and/or blocks of code, and referencing APIs and documentations.
- End-users are allowed to interact only with the web application's view by fast-forward/rewind/pause/play time, uploading TLE files, choosing a view reference, and choosing which satellite's orbit to visualize.
- When in use with a server or ran locally, Node.js and the dependencies contained within the package.json are required for the web application to run as intended.
- No client may interact with another client's unique view of the web application.
- This web application's minimum hardware requirement are as follows:
    - Intel(R) Core™ i7 CPU 2.00 GHz
    - At least 4.00 GB of RAM

2.3 Goals and Guidelines

- The main priority of this product is to fulfill the requests of the liaison on a week-to-week basis.
- The main goal of this project is to accurately visualize multiple satellites in space with minimal delay.
- The goal is to have the product finished ahead of schedule to make room for debugging.
- Efficiency will serve as a key principle in the finished product as it has shown signs of delay.

2.4 Development Methods

The development of this software resembles Scrum Development. Our requirements were provided on a weekly basis shortly after finishing the previous week's requirements. Scrum Development includes methodologies similar to Iterative Development, in which development cycles are done between one to two weeks. In this software's development, the requirements were given, reviewed, sprinted and appended to within the span of a week.

3. Architectural Strategies

The team used JavaScript as the programming language to develop the liaison's desired web application. Alongside JavaScript, the team also implemented Node.js, Cesium.js, Vue.js, and socket.io for generating unique client instances which can update one another's TLE list. JavaScript was the language of choice since the team has knowledge of JavaScript from taking the same class topic. Java was considered but was immediately denied due to the use of Cesium.js.

The Iterative Development approach was used in response to frequent weekly/biweekly meetings. The Waterfall methodology, which the team was familiar with, was replaced with the Scrum methodology. As such, roles were split and each member was responsible for bits and pieces of the application and documents and different demos were generated at times. With different versions of the same application, merge conflicts arose. The team then adapted by performing tests on smaller parts of the application to narrow down bugs. The team also considered a Test-Driven Development approach but the pressure of weekly progress made it difficult to extensively test each part of the application.

As more requirements surfaced, modules were assigned to team members. A necessary Level 0 DFD were to be drafted in order to maintain communication and development management throughout the whole web application. The Scrum methodology prevented the team in sticking to a planned Level 0 DFD and it was revised many times. The DFDs in general were as dynamic as the development of the application.

In tandem with the software development process, Software Requirements and Software Design Documents were being drafted. Alongside the growing list of features, the documents were dynamically changing with each iteration of the web application development process. This was to record the changes/updates/removal of features in the application. Due to the need of producing results and updates quickly, research of existing software components were the ideal approach as opposed to developing the team's own.

As stated earlier in this document, compile errors were easily caught by our team with error printing and code tracing. The runtime errors were often found by external parties like our liaison and advisors. The process of debugging runtime errors mainly stemmed from reenacting the causations of the external parties' bugs.

After a weekly/biweekly meeting with the team's liaison and advisors, we congregate into planning the next sprint of each member for the next iteration of our web application.

As for future plans the team is polishing up part 1 of the project and in the next term part 2 will be the main focus. Due to a Non-Disclosure Agreement the team will sign off on it will be forbidden to document any of the progress pertaining to part 2 of the project.

4. System Architecture

The System's Architecture relies on the MVC Model. The VM is the view, the SCM is the controller, and the rest of the modules are the Models. Although the SCM does a bit more than it ideally should this is due to Cesium.js' library structure.

The clients will be able to interact with their unique view of the web application. Clients are given a list of parsed satellite TLE/CZML orbits to view, a variety of view reference modes of the Earth, the ability to rewind/fast-forward/play/pause time, and to upload satellite TLE files for more satellite orbits to view. It is also responsible in displaying the Vue component of the application for which the client can view all of the following mentioned above. When a clients select their view reference of the Earth, whether it be ICRF or ECI, the clients' view are immediately updated with the chosen options. It does not communicate with the server whatsoever although it does communicate with the TSM (4.1.6). The same applies with the selection of satellite TLE files/CZML files. These tasks are contained and only remain within the VM (4.1.2).

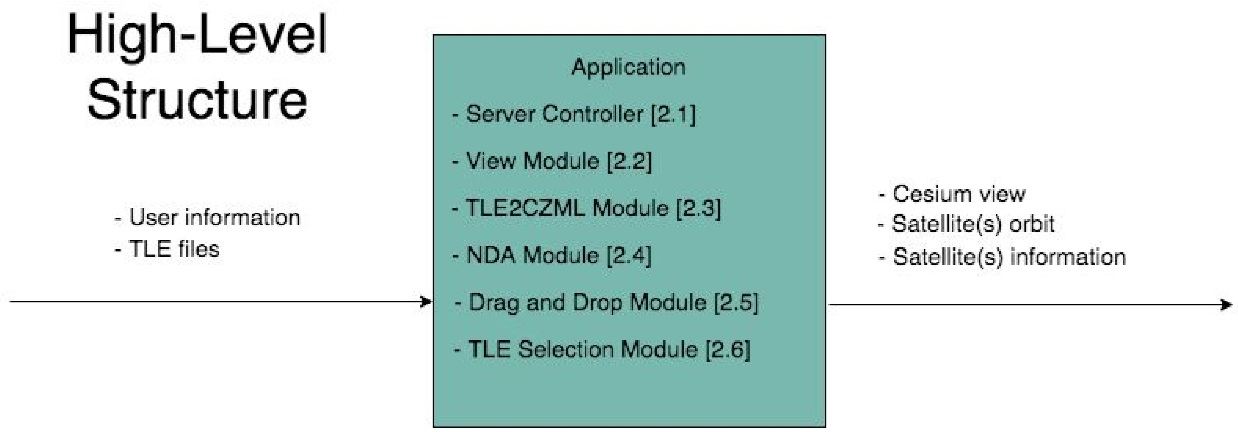
The SCM (4.1.1) handles creating the unique views for each client using socket.io so that they may view any satellite orbit they want. When given a TLE file, it will store it in the server's directory where it will wait for the T2CM to parse it into a CZML file. However, when another client uploads a TLE file the other clients will be notified and will show a new list of parsed TLE files/CZML files that contains the uploaded TLE. This module is the middle man between the rest of the modules.

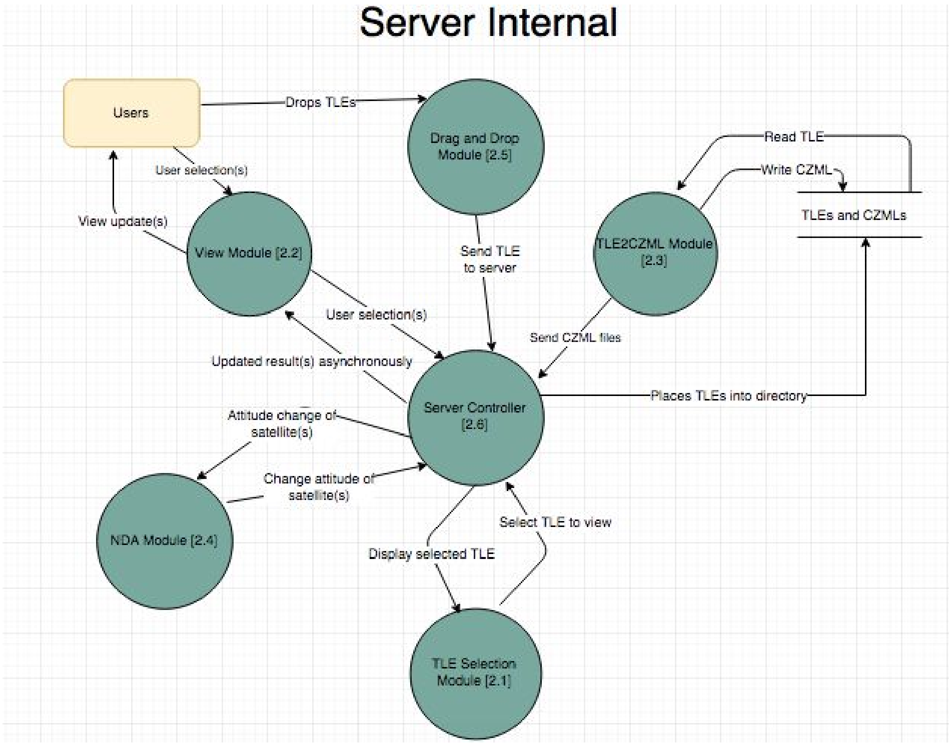
The DDM (4.1.5) simply handles the uploading of satellite TLEs to the SCM. This module is needed for clients to generate a view of satellite orbits that they are specifically looking for. It is a one-way module in that it only passes a TLE to the backend.

The T2CM (4.1.3) is the parser of the satellite TLE files. Cesium.js requires CZML files in order to view satellite orbits. This module simply takes the TLE provided from the SCM and extracts the necessary data utilizing the SGP4 Mathematical Model. It then passes the CZML file back to the SCM and then back to the VM.

The TSM (4.1.6) communicates with the VM so that clients may choose which orbit to view. It sifts through the large CZML files that has been passed to the frontend and returns the selected satellite orbit to the client's view.

The list of satellite TLEs and the client's view are only refreshed when a client has uploaded a satellite TLE file to the server. The VM then communicates with the DDM (2.5), sends the TLE files to the SCM, and then to the T2CM (2.3) where it is to be parsed into a CZML file. The CZML file then is sent back to the SCM and finally displayed on the clients' VM where they can see the newly parsed satellite orbit.





5. Policies and Tactics

5.1 Choice of which specific products used

Vim, Visual Studio Code, Sublime Text 3, JavaScript, CesiumJS API, satellite.js library, Node.js, Vue.js, socket.io, Vue-Socket.io, Formidable Node.js module, File-Loader Node.js module, Babel compiler, webpack static module bundler, and cross-env module.

5.2 Plans for ensuring requirements traceability

- Weekly teleconferences with the liaison ensure that the requirements are satisfied to the liaison’s discretion
- Requirements are given to development team members to divide and fulfill for the next meeting
- As requirements are fulfilled, team members may assist other members (with unfinished tasks) in order to fulfill the week’s requirements
- Requirements must always be done the same week they are given as to not conflict with requirements that append onto the initial requirements.
- To ensure requirements are strictly met, debugging is done within the week after an advisor, liaison, or development team find a bug

5.3 Plans for testing the software

Plans for testing the SAOV are to be done when the development team finishes implementing the weekly requirements, they are as follows:

5.3.1 Engineering trade-offs

- Our method of engineering, Scrum, allows the team to work quick but at the cost of planning. The team might not always have the correct idea or are unsure of how complex one requirement may be.
- Scrum requires the development team to do weekly sprints, which leaves room open for bugs as requirements are done quickly.

5.3.2 Coding guidelines and conventions

- Versions of the software with a bug fix will be uploaded to its respective GitHub branch
- Once a version is approved, the GitHub master will update the software to its most current version
- Comments indicating further review of code are to be left in until the code is finalized by the lead programmer

5.3.3 The protocol of one or more subsystems, modules, or subroutines

- The VM must constantly communicate with all other modules in order to provide a real-time visualization of TLE files.

- The VM is dependent on an internet connection, the Earth visual is powered by Bing Maps

5.3.4 The choice of a particular algorithm or programming idiom (or design pattern) to implement portions of the system's functionality

- The SGP4 Mathematical Model is used to extract data from the TLE files, this allows trajectory tracing
- Model-View-Controller Architecture

5.3.5 Plans for maintaining the software

- Weekly check-ins with liaison and advisers, which are in possession of the deployed website.
- Interface testing on various machines to ensure that there are no compatibility issues.
- Interface testing on various machines to measure hardware and software constraints.
- Weekly redeployment of the software to present the most up-to-date version.

5.3.6 Interfaces for end-users, software, hardware, and communications

- As described in 5.3.5, advisers and liaisons will be in possession of the deployed software for testing
- The development team will dedicate time to testing the website on different computers, at the same time, in order to test out instances of the website

5.3.7 Hierarchical organization of the source code into its physical components (files and directories).

- Organization of the code will be kept similarly to what CesiumJS provides.
- CesiumJS provides a file structure with models and source code by default, new directories will be made for the upload feature to direct to within the server

5.3.8 How to build and/or generate the system's deliverables (how to compile, link, load, etc.)

- The software will be stored on GitHub, which can be downloaded using Git
- Once the repository is downloaded, the program must have its dependencies installed using npm install
- With the files complete, the program can be launched using npm start

6. Detailed System and Lower Level Component Design

6.1 View Module

6.1.1 Responsibilities

The primary responsibilities of this component provides a GUI for the clients to interact with. With this module the clients are able to upload TLE files of satellites, alter their view of the

Earth, and alter the speed of time of the view. It also communicates with the TSM, the SCM, and the DDM. This module can be regularly updated by the clients and is the most dynamic of all the other application components.

6.1.2 Constraints

This module is in charge of displaying the satellites provided by the clients. It is capable of displaying at most 3 satellite orbits at once. If the clients choose to view more than 3 this module will slow down and clients will have a harder time visualizing a smooth orbit of a satellite. With each upload of a TLE file to the server, the size of the CZML increases and thus slows down the web application. It is also used to alter the view of the Earth. The clients may choose to view the Earth in a flat or spherical view (If spherical ICRF or ECI).

6.1.3 Composition

The clients are allowed to view the Earth in a spherical or flat view. When chosen to view the Earth as flat, the view resembles a physical map representation of the Earth. The satellite orbit is then represented to not orbit in a circular or ellipsoid manner but would resemble a sinusoidal path across the flat map. When chosen to view the Earth in ICRF the camera view would be static and the Earth would spin in place. The satellite would have a static orbital path as the Earth turns. In ECI the camera view would spin around with the Earth which stays in a static view. In this mode the satellite orbit would spin around the Earth along with the camera view.

6.1.4 Uses/Interactions

The upload of the satellite TLEs would send the data to the SCM to be converted by the T2CM. It then waits for the converted TLE back as a CZML so that it may update the list of satellite TLEs for the clients to view. Therefore, the TLEs are used used by the VM and the T2CM. The SCM simply transports it from the VM to the T2CM.

6.1.5 Resources

This module requires some time to receive large CZML files from the SCM. The upload of the satellite TLE file does not require much time as it is merely 160 Bytes but the typical CZML to the view are returned in 54 Megabytes. The display of the 3D satellite models plus the large CZML files will definitely slow down the client’s view if the client’s machine does not meet the minimum hardware requirements. When a client uploads a satellite TLE to the SCM all the client’s TLE list will update asynchronously.

6.1.6 Interface/Exports

6.1.6.a Subroutines:

6.1.6.a.i: viewInICRF

Definition: Changes the view of the Earth into International Celestial Reference Frame

Responsibilities: In charge of the client’s view of the Earth in the web application

Parameters: N/A

Returns: N/A

Uses: Cesium object camera functions, changes the camera to rotate with the Earth and enables lighting

Constraints: Only capable of setting the Earth view to ICRF

Interface: Drop-down list element

6.1.6.a.ii: icrf

Definition: Changes the view of the Earth into International Celestial Reference Frame

Responsibilities: In charge of the client's view of the Earth in the web application

Parameters: scene, time

scene: Cesium object property, the current state of the Cesium object

time: Given time to reference

Returns: If scene is not 3D it would return undefined, else it does not return anything at all

Uses: Cesium object camera functions, changes the camera and fixes it on a specific spot to view Earth rotate in place

Constraints: Only capable of setting the Earth view to ECI

Interface: Drop-down list element

6.1.6.a.iii: changeView

Definition: Changes the client's view of the web application with respect to the selected drop-down

Responsibilities: In charge of applying client's view change

Parameters: viewValue

viewValue: integer, 1 for viewInICRF and 2 for eci

Returns: N/A

Uses: Cesium object camera functions, changes the camera view in ECI or ICRF and enables/disables lighting

Constraints: N/A

Interface: Drop-down list

6.1.6.a.iv: socket.on('connect', …)

Definition: Keeps an open event listener for when the server and the client are able to connect with one another

Responsibilities: Listens to server-side socket event "connect"

Parameters: function

function: A function from the sockets.js which is called when a new

CZML has been uploaded

Returns: N/A

Uses: Keeps event listeners open for communication between client and server-side

Constraints: Deadlocks when a client uploads a TLE file to the server

Interface: N/A

6.1.6.a.v: socket.on('addingCZML', …)

Definition: Keeps an open event listener for when the T2CM sends back a CZML file to the client view

Responsibilities: Adds a new satellite orbit to view by the client

Parameters: czmlPicked

czmlPicked: A function from the socket.js which is called when a CZML has been chosen from to viewed by the client

Returns: N/A

Uses: Updates the list of TLEs that can be viewed by the clients on the client-side

Constraints: Causes a refresh of the client's view when a new CZML file is being uploaded by the client

Interface: N/A

6.1.6.b Exports:

6.1.6.b.i: TLEs

Definition: Two-Line Elements, data format which contains orbital elements of an Earth-orbiting object accurate for 3 to 5 days upon retrieval

Responsibilities: Contains encoded satellite orbit data

Type: .tle file

6.2 TLE2CZML Module

6.2.1 Responsibilities

The primary responsibility of this module is to convert satellite TLEs from a directory in the server to CZML files that Cesium.js can read and display to the client's view. This module communicates with the SCM which sends the CZML file back to the VM.

6.2.2 Constraints

It can only receive satellite TLE files which it converts to CZML. The data obtained will be accurate for up to 3 to 5 days.

6.2.3 Composition

This module reads from a directory filled with TLE files sent by the SCM to be converted into CZML files.

6.2.4 Uses/Interactions

This module collaborates with the SCM and DDM in order to convert the clients' satellite TLEs. The CZML that has been converted will be used in the VM. The VM will then generate a satellite orbit for the clients to interact with. The conversion takes a while and the satellites will be shown on the VM after a couple of seconds.

6.2.5 Resources

This module requires a directory to store all the TLE and CZML files for the SCM to pull and send to the VM. With the help of Socket.io, the client will not experience a deadlock when uploading a TLE using the DDM but the satellite to be viewed still requires time for it to show up.

6.2.6 Interface/Exports

6.2.6.a Subroutines

6.2.6.a.i satellite.sgp4(tle_line1, tle_line2)

Definition: Utilizes the Simplified Perturbations Model SGP4 to extract satellite's x, y, and z positions (Longitude, latitude, and altitude).

Responsibilities: Extracts and converts information from TLEs to CZML.

Parameters: tle_line1, tle_line2

tle_line1: First line of TLE

tle_line2: Second line of TLE

Returns: CZML file

Uses: Generates a visual representation of the satellite TLE sent by the client.

Constraints: Cannot take in other file types. Strictly TLE files.

Interface: N/A

6.3 Drag and Drop Module

6.3.1 Responsibilities

This module is responsible for providing the client with an area to drag TLE files onto in order to upload the file into the server and add the TLE to the optional view on the VM. Similar to the browse and upload buttons on the user interface, the drag and drop serves as an optional upload method.

6.3.2 Constraints

This module only accepts one TLE file at a time.

6.3.3 Composition

The area to the top left of the UI contains a small box where users are capable of dragging one file into the area in order to upload a TLE to the server to be read on the Cesium backend.

6.3.4 Uses/Interactions

The DDM communicates with the Server Controller and begins a cycle of conversion through the T2CM in order to translate the TLE into a CZML file so that the VM will then display the designated satellite and its path.

6.3.5 Resources

A TLE file is a necessary resource for this module to perform. It is also dependent on the Server Controller in order to communicate between other servers and their dependency on conversion and display.

6.3.6 Interface/Exports

6.3.6.a Subroutines

6.3.6.a.i <form action=”/” enctype=”multipart/form-data” method=”post” id=”formID”> … </form>

Definition: HTML form field to receive satellite TLEs from clients.

Responsibilities: Sends the satellite TLEs to the SCM for conversion.

Parameters: file

file: A satellite TLE file which will be sent to the server to be converted into CZML

Returns: N/A

Uses: Obtains client TLE for CZML conversion

Constraints: Only accepts TLE files.

Interface: HTML form field

6.4 TLE Selection Module

6.4.1 Responsibilities

This module is responsible for obtaining a TLE selection from the client via radio button. Once the selection has been made, this module generates a visual for the Satellite path. It also responsible for displaying the Earth's 3D model in two modes, ICRF and ECI.

6.4.2 Constraints

Only up to three TLE files may be selected at a time.

6.4.3 Composition

To the left of the UI, there is a selection list that can access converted TLE files in order to allow the user to display up to three paths.

6.4.4 Uses/Interactions

The TSM interacts solely with the Server Controller in order to access files already stored on the server, provided by the user. It waits for user input before sending any signals.

6.4.5 Resources

Similar to the DDM, the TSM is dependent on existing TLE files in order to send a signal to the Server Controller, to the VM.

6.4.6 Interface/Exports

6.4.6.a Subroutines

6.4.6.a.i socket.on('addingCZML', …)

Definition: Keeps an open event listener for when the client selects a TLE from the TLE list so the TSM knows which CZML to send back to the VM

Responsibilities: Adds a new satellite orbit to view by the client

Parameters: czmlPicked

> **czmlPicked:** A function from the socket.js which is called when a CZML has been chosen to be viewed by the client

Returns: N/A

Uses: Updates the list of TLEs by turning on the radio button of the TLE the client has chosen to view

Constraints: Causes a refresh of the client's view when a new CZML file is being uploaded by the client

Interface: N/A

6.5 Server Controller Module

6.5.1 Responsibilities

The SCM is responsible for communicating between modules. It sends signals for modules to perform and receives feedback for actions requested.

6.5.2 Constraints

The SCM only sends and receives information.

6.5.3 Composition

The SCM is a backend tool used to ensure the application is stable around all its modules.

6.5.4 Uses/Interactions

The SCM interacts with the VM, DDM, T2CM, and TSM. Files are stored in the server, so that the SCM make the VM display a file the TSM wants it to. The DDM sends files to the server, so that the SCM can then make a final judgment as to what to do with the files.

6.5.5 Resources

The SCM is dependent on all modules to perform their programmed tasks in order to let other modules know to update the UI, perform a conversion, or store a file on the server.

6.5.6 Interfaces/Exports

6.5.6.a Subroutines

6.5.6.a.i socket.on('connect', …)

Definition: Keeps an open event listener for when the server and the client are able to connect with one another

Responsibilities: Listens to server-side socket event "connect"

Parameters: function

function: A function from the sockets.js which is called when a new

CZML has been uploaded

Returns: N/A

Uses: Keeps event listeners open for communication between client and server-side

Constraints: Deadlocks when a client uploads a TLE file to the server

Interface: N/A

6.5.6.a.ii socket.on('addingCZML', …)

Definition: Keeps an open event listener for when the client selects a TLE from the TLE list so the TSM knows which CZML to send back to the VM

Responsibilities: Adds a new satellite orbit to view by the client

Parameters: czmlPicked

czmlPicked: A function from the socket.js which is called when a CZML has been chosen to be viewed by the client

Returns: N/A

Uses: Updates the list of TLEs by turning on the radio button of the TLE the client has chosen to view

Constraints: Causes a refresh of the client's view when a new CZML file is being uploaded by the client

Interface: N/A

6.5.6.b Exports

6.5.6.b.i CZMLs

Definition: Uncompressed Cesium Language (CZML) containing information to display proper satellite orbits at 3 to 5 days, at most, for accuracy. Data is retrieved from TLEs (Two-Line Element) provided from the client

Responsibilities: Contains decoded satellite orbit data for Cesium to read

Type: .czml file

7. Database Design

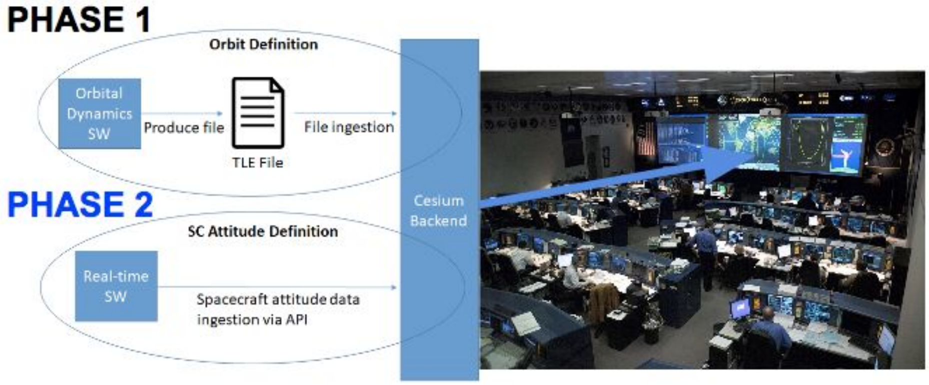
No databases have been used during the first semester of the software's development. A simple directory called 'TLEs' is used to store the TLE and CZML files that Cesium reads and outputs to the VM.

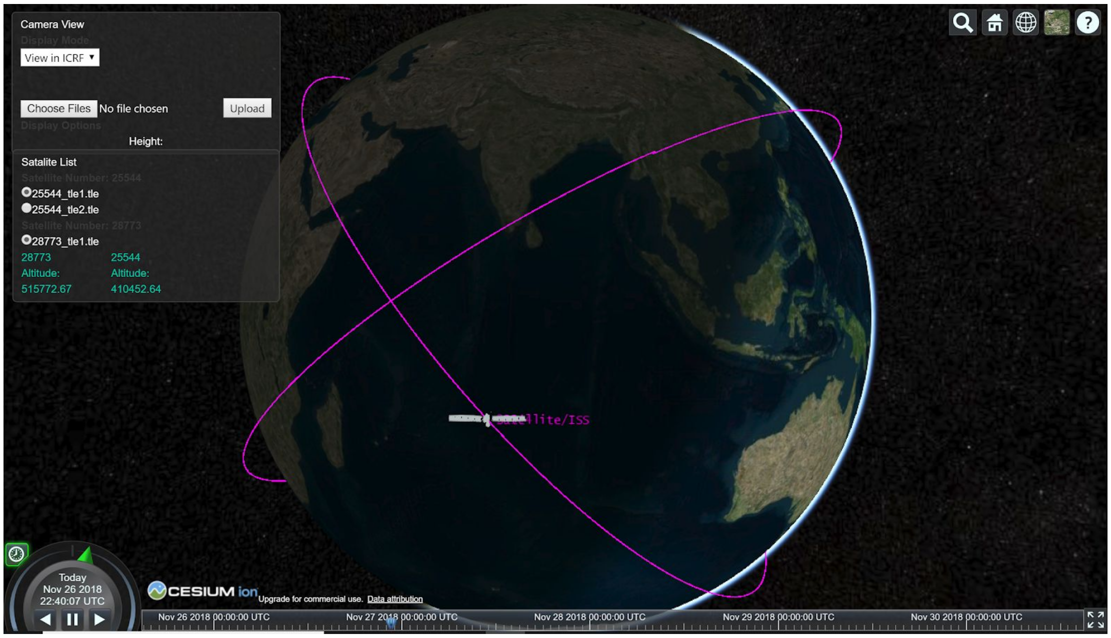
8. User Interface

8.1 Overview of User Interface

A client will be able to access functions by clicking the buttons in their respective areas. The browse function will allow users to upload two-line element files, which will be assumed to be in the possession of the user. A similar function, the drag and drop area, will allow users an alternative way to upload two-line element files. The user can click on a file and drag it onto the designated area on the website to upload their TLE files. The user can then interact with an uploaded file through the radio buttons generated as the files are uploaded. Clicking on the radio button accesses the display function that generates a visualization of the satellite orbit around the Earth. The user can interact with the display through mouse clicking and movements to rotate the display and obtain a three-dimensional view of the satellite visual.

8.2 Screen Frameworks or Images





No file chosen

Satellite Number: 25544

○25544_tle1.tle

○25544_tle2.tle

Satellite Number: 28773

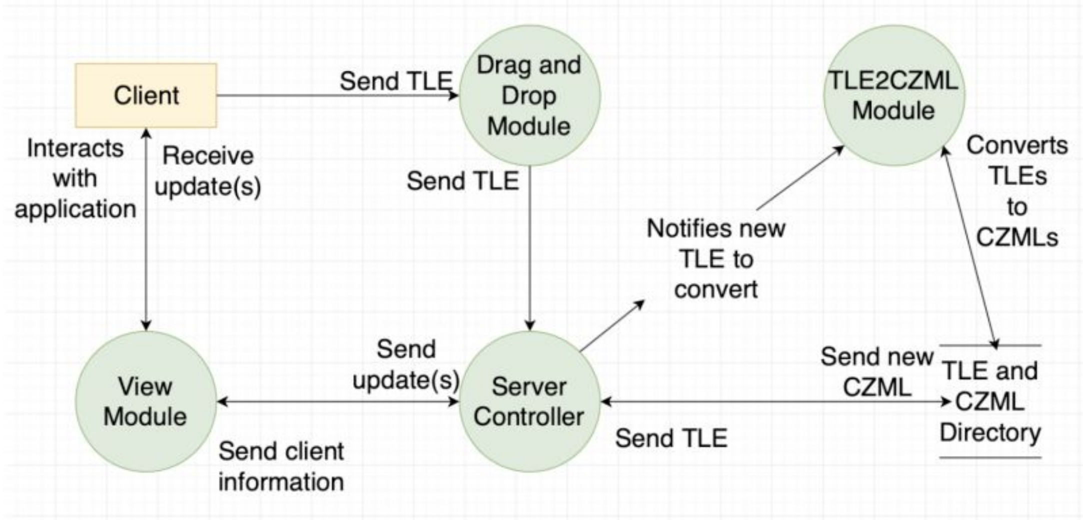
○28773_tle1.tle

28773	25544
Altitude:	Altitude:
515772.67	410452.64

Today
Nov 26 2018
22:40:07 UTC

ellite/ISS

8.3 User Interface Flow Model



9. Requirements Validation and Verification

Requirement Validation and Verification for Modules	
Module	**Requirement Description and Testing method**
2.1	3.1.1 - The module was provided with TLE files. Upon successful submission, the satellite path was traced with no recorded issues for up to three satellites.
2.2	3.2.4 - The VM was provided with TLE files. Upon submission, the files are stored and read by the server. They are also appended to the list in view.
2.4	3.4.2 - The designated space to drag a file onto the web application has been allocated. It was tested and works the same as if the user were to manually upload via the browse function.
2.5	3.5.1 - Options have been tested to ensure that the view responds to the user input via the mouse. Satellite altitude is displayed for each satellite clicked on and the user is free to upload and choose which satellites they wish to display.

2.6	Testing for this has been done from the previous modules.

10. Glossary

CZML	**Cesium Language**
DDM	**Drag and Drop Module**
RTAM	**Real-Time Attitude Module**
SAOV	**Satellite Attitude and Orbit Visualization**
SCM	**Server Controller Module**
T2CM	**TLE2CZML Module**
TLE	**Two-Line Element Set**
TSM	**TLE Selection Module**
VM	**View Module**
ICRF **ECI**	**International Celestial Reference Frame** **Earth-Centered Inertial**

11. References

cesium.js API ver. 1.51 by Analytical Graphics, Inc., Bentley Systems, and Rafael: 2018
https://cesiumjs.org/refdoc/

satellite.js library ver. 2.0.2 by Shashwat Kandadai: 2018
https://github.com/shashwatak/satellite-js/wiki

SPACETRACK REPORT NO. 3 Models for Propagation of NORAD Element Sets by Felix R. Hoots and Ronald L. Roehrich - Package Compiled by TS Kelso: December 1988
https://www.celestrak.com/NORAD/documentation/spacetrk.pdf