Object Design Document
for the
Rotation Curve Modeler (RoCM)

Version 2.0

Robert Moss, Alex Clement

August 2, 2014

Contents

1 Introduction

1.1 Object Design Trade-offs

1.1.1 Buy vs. Build

There is no product in circulation that allows a user to plot all the existing theories that solve the rotation curve of galaxies. For this reason, we will have to build such a system, because buying it is not an option. In order to expedite the process of solving the rotation curve problem, a piece of software like RoCM will have to be created.

1.1.2 Space vs. Speed

Speed is a major priority for RoCM. The latency in updating each model will hinder the functionality of the application. RoCM 1.0 benchmarks extremely well with heavy computations. This gives light to RoCM 2.0 to continue to use JavaScript for the galactic models because the speed has not presented itself as an issue. The amount of space needed to hold the entire program will be negligible, because it will be held on an external server.

1.1.3 Delivery Time vs. Functionality

If the development of RoCM is behind schedule, certain features and models can be put on hold. The modularity of the design allows for this. Although, to get the full functionality of the tool, all the models will need to be implemented before it's release.

1.1.4 Delivery Time vs. Quality

If testing runs behind schedule, the software can still be released and updates can be released to fix bugs at a later date. As long as the core functionality of the rotation curve plotter and several models are provided, the product can be delivered.

1.1.5 Files vs. Databases

Files would not be as beneficial as a database because the data for RoCM is kept external from the user. A database would provide a sufficient structure to organize the galactic data and parameters. In using a file, users can theoretically edit and thus corrupt the file and the data inside. An "off site" database would be more secure and less likely to be modified.

1.2 Interface Documentation Guidelines

Classes, Interfaces & Packages: These names should be in *Pascal Case.*

- i.e.: CurvePlot, GalacticModel, ParamSlider

Constants: All constants should be entirely in *Upper Case.*

- i.e.: MODEL, PARAMS

Functions: Identifier and Method names should be *lowercase* and separated by an *underscore.*

- i.e.: get_range(), update_line()

Local Variables: Variables should be in *Camel Case.*

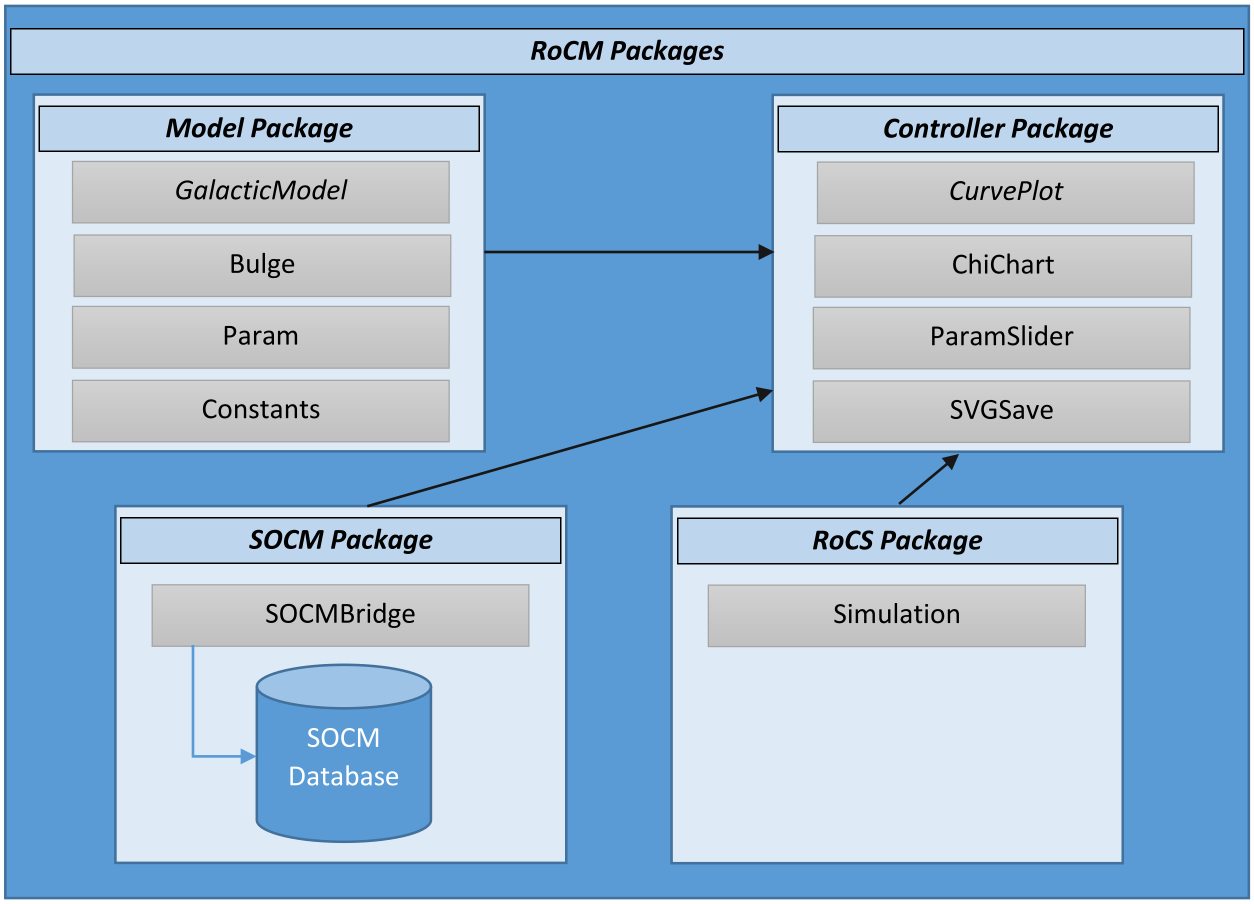
- i.e.: data, errorBar, fileName

1.3 Definitions, Acronyms, and Abbreviations

- Application Specific Definitions
  - RoCM - Rotation Curve Modeler
  - SOCM - Scholarly Observed Celestial Measurements
  - RoCS - Rotation Curve Simulator
- Industry Definitions
  - WIT - Wentworth Institute of Technology
  - ODD - Object Design Document
  - JavaScript - A web based programming language.
  - D3 - Data Driven Documents: A JavaScript library for data visualization.
  - Ruby on Rails - A web development framework written in the Ruby programming language.
  - JQuery - A JavaScript library for easy UI development.
  - LaTeX - A document preparation system used widely throughout science and mathematics.
  - SVG - Scalable Vector Graphics: A loss-less graphics format.
- Technical Definitions
  - MoND - Modification of Newtonian Dynamics
  - TeVeS - Tensor-vector-scalar gravity
  - MATLAB - A mathematical programming language (MATrix LABoratory).
  - Mathematica - A mathematical programming language.
  - DB - Database
  - UI - User Interface
  - GUI - Graphical User Interface
  - HTML - HyperText Markup Language
  - div - HTML tag to define a division in a document
  - DOM - Document Object Model. A convention for representing and interacting with objects in HTML.
  - API - Application Programming Interface
  - km/s - Kilometers Per Second
  - kpc - Kiloparsecs

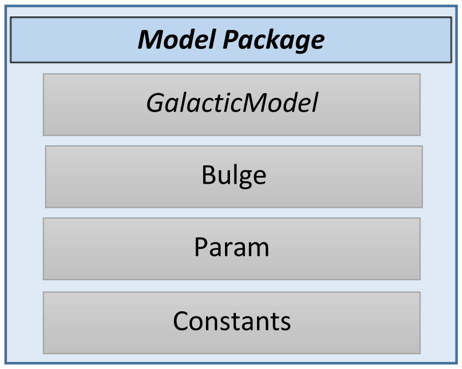
2 Packages

2.1 Package Diagram



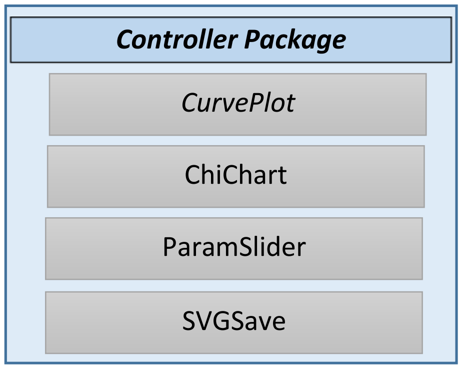
2.2 Package Definition

2.2.1 Model Package



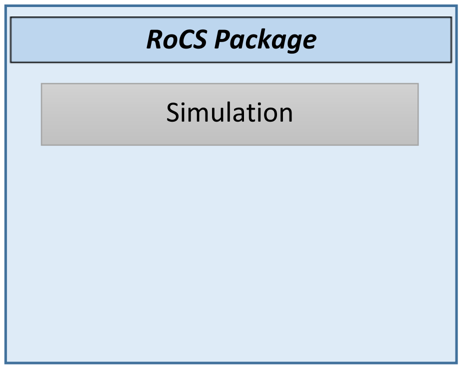
The model package will consist of the GalacticModel function set. It will include the Bulge contribution as a separate object. The Constants object will be initialized within this package because the GalacticModels are the only set of functions that use the Constants. The Param variable type will also be under this package.

2.2.2 Controller Package



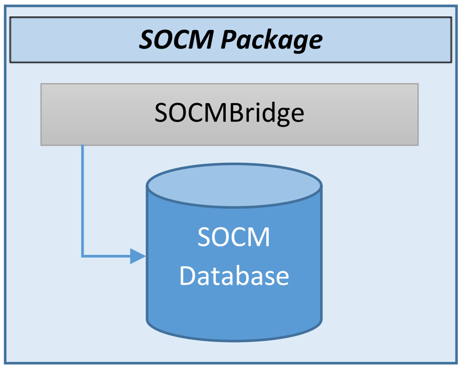
The controller package will handle all of the graphing and curve plotting for RoCM. The CurvePlot object will deal solely with the data plotting and curve plotting functionality. The ChiChart histogram will generate it's own graph within this package. The ParamSlider object, which updates the CurvePlot and ChiChart, will be included in the controller package. Lastly, the SVGSave object which handles the conversion of the computed SVG elements into a static, savable, SVG file.

2.2.3 RoCS Package



The RoCS package will handle all of the rotation curve simulation done in D3. The simulation function is the key component to this package.

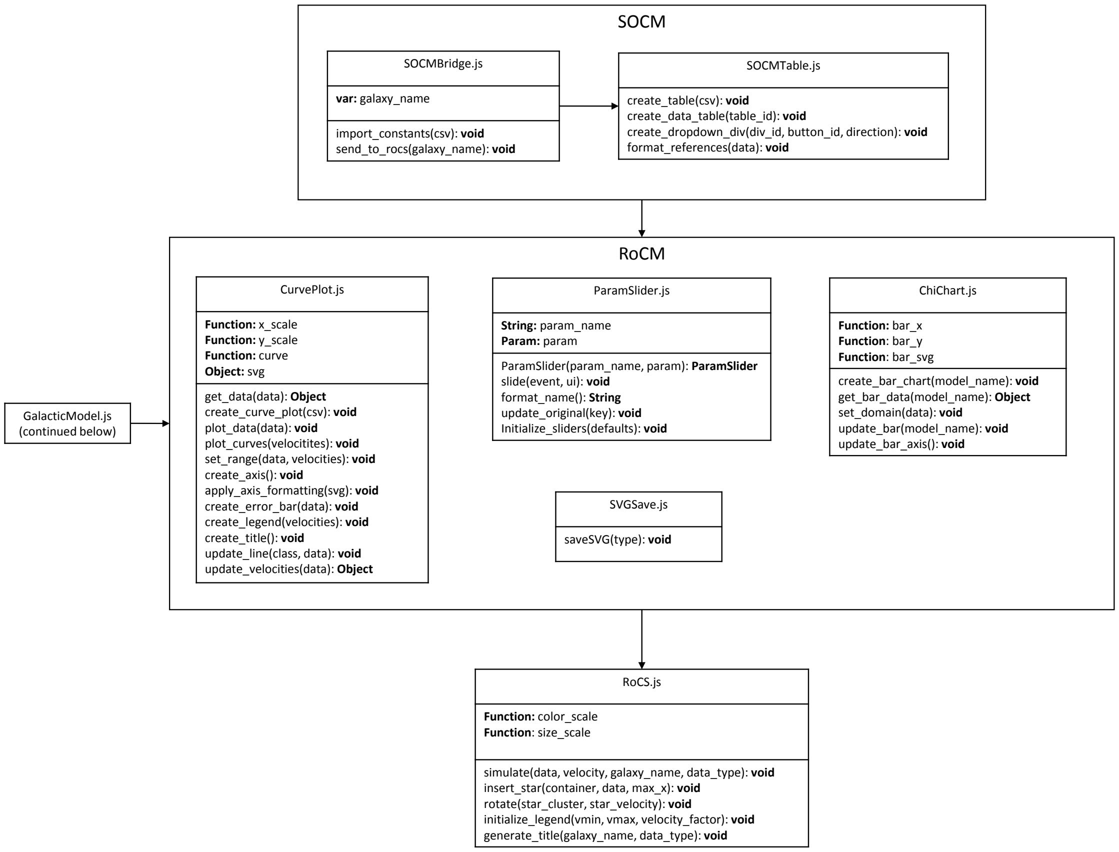
2.2.4 SOCM Package



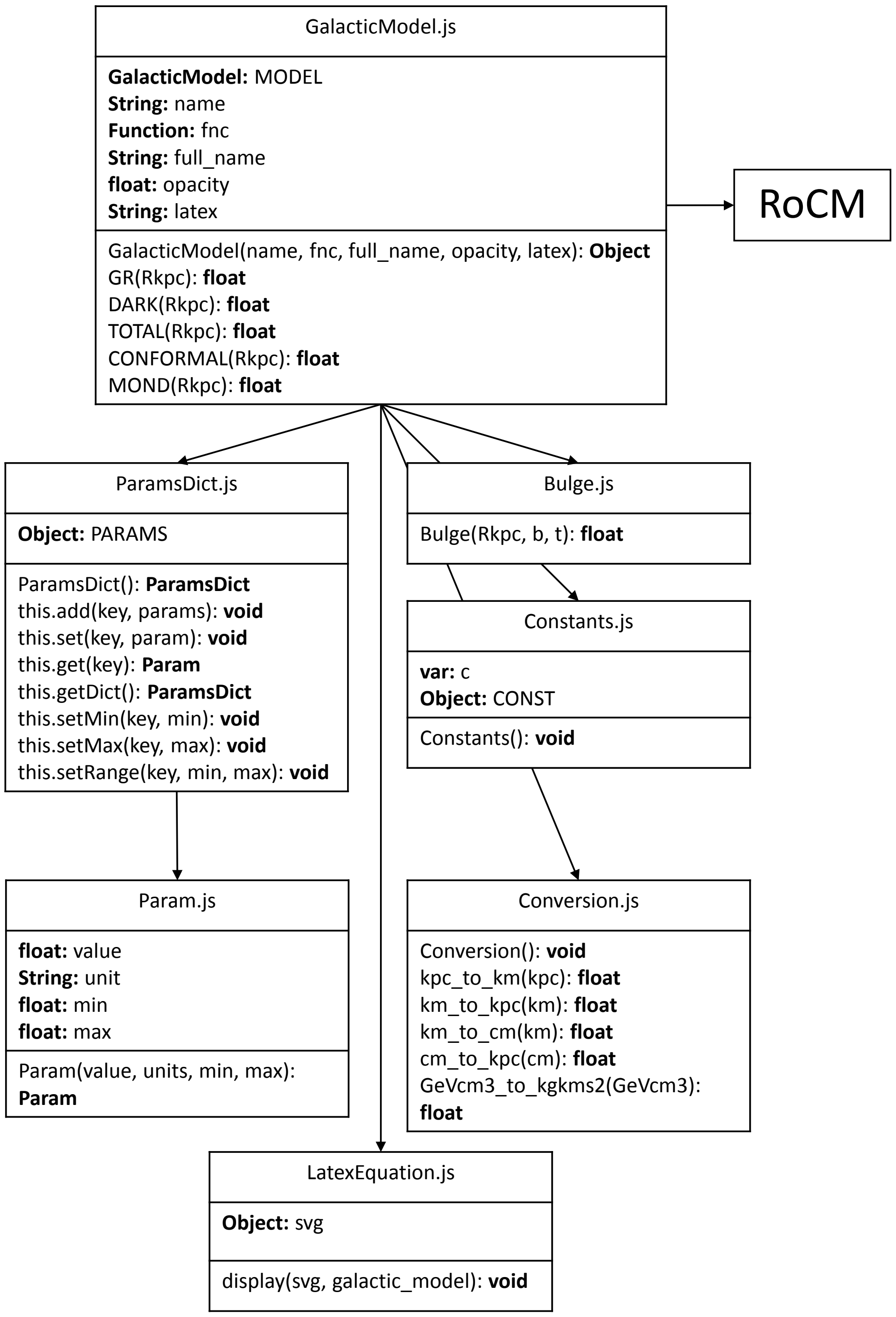
The SOCM package is an entry point into the application. The SOCMBridge will query galactic data from the SOCM Database. All data retrieval will be handled within this package.

3 Class Interface

3.1 Class Diagram


SOCM
SOCMBridge.js
var: galaxy_name
import_constants(csv): void
send_to_rocs(galaxy_name): void
SOCMTable.js
create_table(csv): void
create_data_table(table_id): void
create_dropdown_div(div_id, button_id, direction): void
format_references(data): void
RoCM
CurvePlot.js
Function: x_scale
Function: y_scale
Function: curve
Object: svg
get_data(data): Object
create_curve_plot(csv): void
plot_data(data): void
plot_curves(velocitites): void
set_range(data, velocities): void
create_axis(): void
apply_axis_formatting(svg): void
create_error_bar(data): void
create_legend(velocities): void
create_title(): void
update_line(class, data): void
update_velocities(data): Object
ParamSlider.js
String: param_name
Param: param
ParamSlider(param_name, param): ParamSlider
slide(event, ui): void
format_name(): String
update_original(key): void
Initialize_sliders(defaults): void
ChiChart.js
Function: bar_x
Function: bar_y
Function: bar_svg
create_bar_chart(model_name): void
get_bar_data(model_name): Object
set_domain(data): void
update_bar(model_name): void
update_bar_axis(): void
SVGSave.js
saveSVG(type): void
GalacticModel.js
(continued below)
RoCS.js
Function: color_scale
Function: size_scale
simulate(data, velocity, galaxy_name, data_type): void
insert_star(container, data, max_x): void
rotate(star_cluster, star_velocity): void
initialize_legend(vmin, vmax, velocity_factor): void
generate_title(galaxy_name, data_type): void

3.2 Class Diagram (Model Package)



3.3 Class Definition

3.3.1 CurvePlot

CurvePlot will hold all of the graphing and plotting functions. It can import data from SOCM by calling *get_data(data)*. The imported data will need to be manipulated in order to generate a multi-line plot, this is done through *update_velocities(data)*. It then will dynamically create a graph, with ranges defined by *set_range(data, velocities)*. Each velocity curve will then be generated dynamically through the function call, *plot_curves(velocities)*. An interactive legend will also be generated with *create_legend(velocities)*. The *x_scale* and *y_scale* are static functions in the place of variables for scaling from pixels to (x,y) coordinates on the generated D3 graph.

3.3.2 ChiChart

The ChiChart histogram is an implementation of a χ^2 (chi squared) distribution. This shows the validity of the model vs. the observed data. The ChiChart will get the data via *get_bar_data(model_name)*. After the data has been retrieved, the domain will be set (via *set_domain(data)*) and the bar will be updated with the current χ^2 of the model vs. the observed data (via *update_bar(model_name)*).

3.3.3 ParamSlider

Each ParamSlider will be created dynamically. All the slider needs to be created is an associative parameter name (one that is in PARAMS), and a Param object. The Param object has 4 fields: value, unit, min, and max. These dictate the value of the parameter, the units in which it's defined, and the min and max values of the ParamSlider. The *slide(event, ui)* function will handle all the updating that needs to be done to the CurvePlot and ChiChart. The function *update_original(key)* can be used to reset the original value of the parameter.

3.3.4 GalacticModel

The GalacticModel object will reside in the global namespace. This global MODEL variable will have each implemented model as an indexable function (ex: MODEL["CONFORMAL"] will return the equation for Conformal Gravity. From here you can use that return object as the direct Confromal Gravity velocity function). Each defined model will have a name (short name for the model), the implemented equation (in the form of a v(R) function), a full name (full name of the model, for the legend), the opacity of the curve (optional), and the LaTeX code of the equation (optional).