太阳系系统

软件构架文档

版本 <1.0>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| **2011-07-21** | **0.1** | **文档的建立** | **马仕青** |
| **2011-08-01** | **0.3** | **修改和完善了文档** | **马仕青** |
| **2011-08-03** | **0.7** | **完善文档** | **马仕青** |
| **2011-09-01** | **1.0** | **完成** | **马仕青** |

目录

[1. 简介 4](#_Toc303814964)

[1.1 目的 4](#_Toc303814965)

[1.2 范围 4](#_Toc303814966)

[1.3 定义、首字母缩写词和缩略语 4](#_Toc303814967)

[1.4 参考资料 5](#_Toc303814968)

[1.5 概述 5](#_Toc303814969)

[2. 用例视图 6](#_Toc303814970)

[2.1 用例实现 6](#_Toc303814971)

[2.2 VOPC： 7](#_Toc303814972)

[3. 逻辑视图 9](#_Toc303814973)

[3.1 概述 10](#_Toc303814974)

[3.2 在构架方面具有重要意义的设计包 10](#_Toc303814975)

[4. 进程视图 13](#_Toc303814976)

[5. 部署视图 13](#_Toc303814977)

[6. 实现视图 14](#_Toc303814978)

[6.1 概述 14](#_Toc303814979)

[6.2 层 14](#_Toc303814980)

[7. 数据视图 14](#_Toc303814981)

软件构架文档 （简化版）

# 简介

软件构架文档的简介应提供整个软件构架文档的概述。

## 目的

本文档将从构架方面对系统进行综合概述，其中会使用多种不同的构架视图来描述系统的各个方面。它用于记录并表述已对系统的构架方面作出的重要决策。

开发人员在开发中，分布培训人员在分发系统时候，测试人员在进行验收和测试的时候都需要使用本文档。

## 范围

本文档仅适用于太阳系系统。文档包括太阳系系统的用例视图、逻辑视图、进程视图、部署视图等视图。并且整个系统将基于此文档编写。将是涉及到了设计人员，编码人员，部署分配等各种用户使用。

## 定义、首字母缩写词和缩略语

1. 日全食：是日食的一种，即太阳被月亮全部遮住的天文现象。如果太阳、月球、地球三者正好排成或接近一条直线，月球挡住了射到地球上去的太阳光，月球身后的黑影正好落到地球上，这时发生日食现象。 在地球上月影里的人们开始看到阳光逐渐减弱，太阳面被圆的黑影遮住，天色转暗，全部遮住时，天空中 可以看到最亮的恒星和行星，几分钟后，从月球黑影边缘逐渐露出阳光，开始生光、复圆。由于月球比地球小，只有在月影中的人们才能看到日全食。
2. 行星连珠：对于“行星连珠”现象，至今并没有一个严格的定义，通常用肉眼望去，行星差不多处在一条直线上，人们就称之为“行星连珠”。
3. 太阳系：太阳、行星及其卫星与环系、小行星、彗星、流星体和行星际物质所构成的天体系统及其所占有的空间区域。
4. 行星： 行星通常指自身不发光，环绕着恒星的天体。其公转方向常与所绕恒星的自转方向相同。一般来说行星需具有一定质量，行星的质量要足够的大且近似于圆球状，自身不能像恒星那样发生核聚变反应。
5. 视角：视线与显示器等的垂直方向所成的角度，观察物体时，从物体两端（上、下或左、右）引出的光线在人眼光心处所成的夹角。
6. 视野：物理意义指眼固定注视一点时或通过仪器时所能看见的空间范围。
7. 项目成员：参与本项目编写、制作、设计成员。
8. 逻辑视图，设计的对象模型（使用面向对象的设计方法时）。
9. 进程视图，捕捉设计的并发和同步特征。
10. 实现视图，描述了软件到硬件的映射，反映了分布式特性。
11. 部署视图，描述了在开发环境中软件的静态组织结构。
12. 用例视图，即所做的各种决定，可以围绕着这四个视图来组织，然后由一些用例 （use cases）或场景(scenarios)来说明，从而形成了第五个视图。

## 参考资料

1. 软件工程——实践者的研究方法》，Roger S.Pressman，郑人杰、马素霞、白晓颖等译。

2. 百度百科。

3. 维基百科。

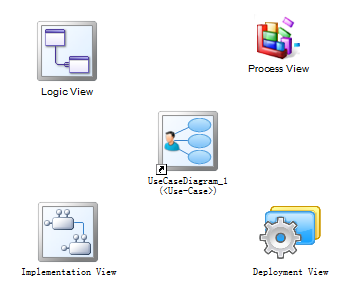
4. 《太阳系系统SRS》

5. 《太阳系系统UseCase规约》

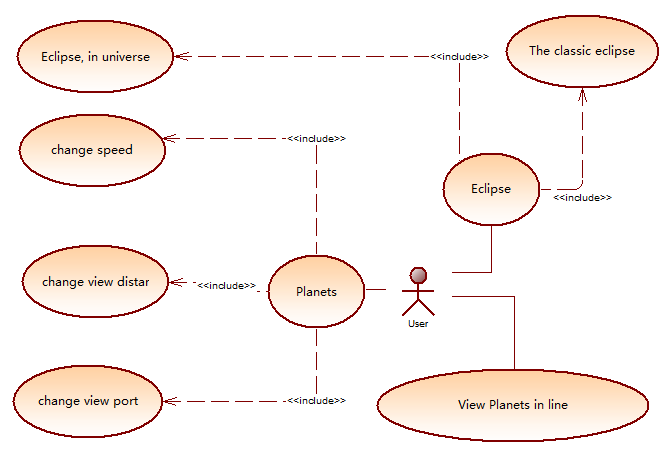
## 概述

本软件架构文档对项目太阳系系统的架构进行了阐述。

用例视图列出用例模型中的一些用例或场景，这些用例或场景应体现最终系统中重要的、核心的功能。逻辑视图介绍了在构架方面具有重要意义的类，并说明它们的职责，以及几项非常重要的关系、操作和属性。部署视图说明用来部署和运行该软件的一种或多种物理网络（硬件）配置。实现视图说明实现模型的整体结构、软件分解为实现模型中的层和子系统的情况，以及所有在构架方面具有重要意义的构件。数据视图从永久性数据存储方面来对系统进行说明。这几个视图共同定义了软件的架构。

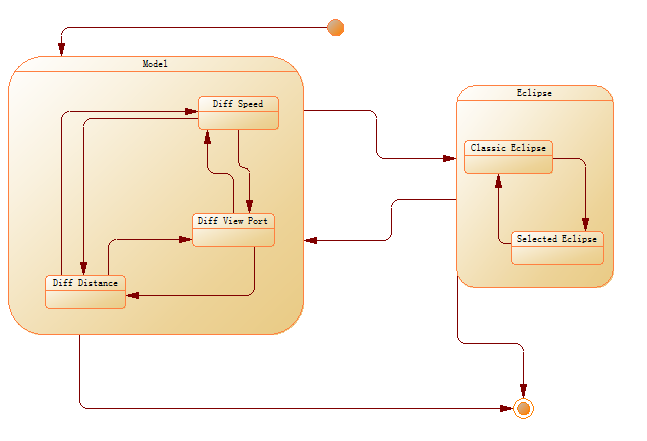


# 用例视图



## 用例实现

具体的各个用例的事件流等请参见[《太阳系系统用例规约》](5090379171_马仕青_UseCase规约_SRS.docx)文档。



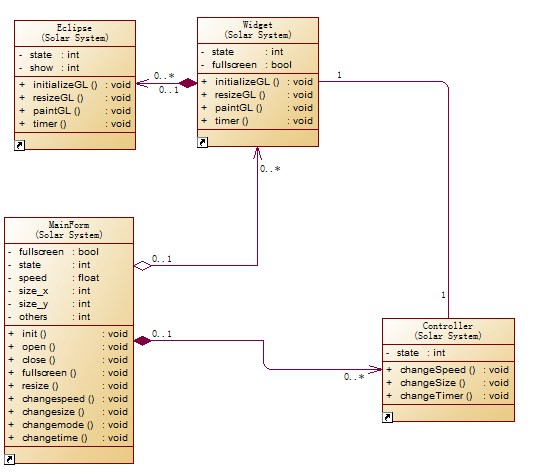
以上是本系统的状态转换图。表明了用户在两个视图模式和各种视图模式中的视图中转换的状态迁移。

## VOPC：

本系统的用例共分为三大类：Elipse，Stars in line 和 Normal。

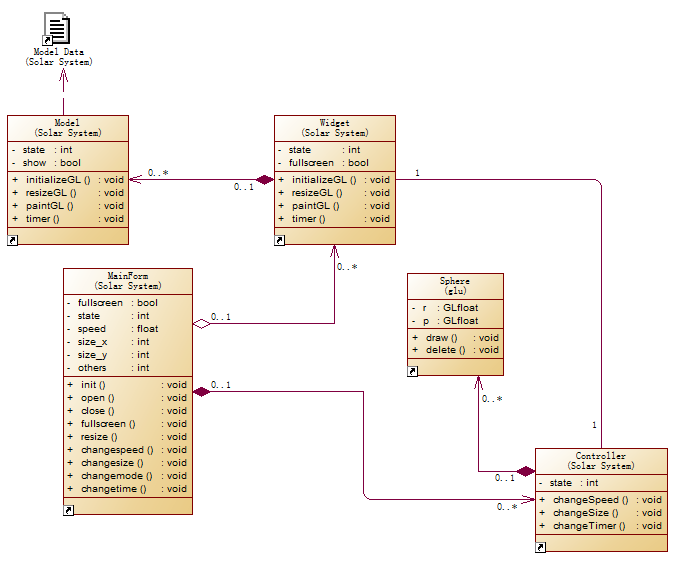
这三种用例的实现类似：都是通过在一个Widget中选择性的绘制组件从而实现了功能。而每一个Widget的控制类通过调用不同的函数实现了视角转化，试图缩放等一系列功能，从而实现了这个大的用例中包含的小型用例。

各个用例的视图如下：

1. Eclipse：

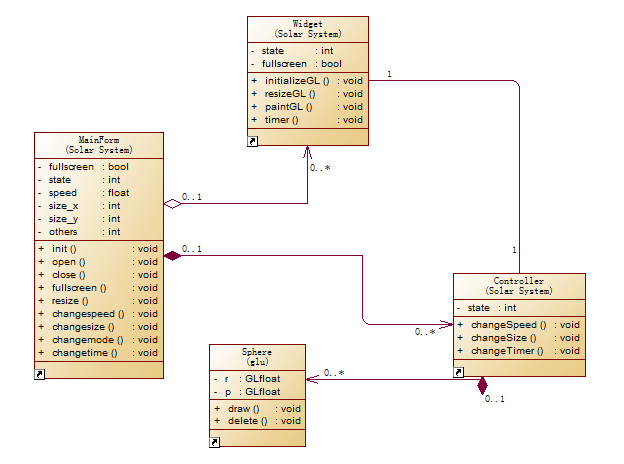
Elipse是通过计算得到的。因此，不需要读取数据源。

1. Normal：



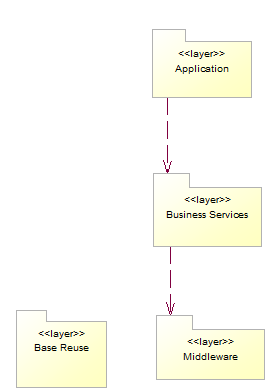
正常状态下的太阳系，需要通过读取文件的方式得到相关的轨道等的数据。同时需要更新数据，保证同步等。

1. Stars in line：



这个用例和上面的Eclipse实现方式类似。

# 逻辑视图



## 概述

本节说明设计模型在架构方面的重要意义。例如设计模型被分成了子系统和包。而每个重要的子系统和包有进一步的被分开成为更小的组件。共同组成了这个程序和系统。本节这是对这些内容的区分和组织做一个介绍说明。而具体的实现和设计，可以参考我们的[相关的设计模型](Project/Project.prj)。

本系统主要分成了3个逻辑层和1个全局层（Global）：应用层（Applications）、业务层（Business Server）和中间件（Middleware）。

应用层（Applications）:

本层主要还是对特定的应用程序的实现的部分。是很少可以复用的部分。本层包含了许多用户界面设计类的实现，通过对用户界面和界面上的逻辑的封装来区分和隔离了对下层的影响。

业务层（Business Servers）:

业务层是一个以业务服务为基础的层。用来组织本系统中的可能需要的各种业务逻辑，将系统中相关的、具有高度相关性的业务组织在一起，形成一个整体的服务。这样的封装使得业务层的逻辑和代码可以实现一定程度上复用。

中间件层（Middleware）:

本层中包含了中间件等底层的服务提供软件和包。通常都是使用特定的公司开发的软件的。本系统没有自己开发中间件。

全局层（Global）：

这一层中包含了每个层中都可能使用到各种包的信息。是一个基本的复用层面。大多是由使用的语言等提供的。本系统不自己开发。

## 在构架方面具有重要意义的设计包

* **Base Reuse**

**等**

**说明：**

**这些Base Reuse的类大多是基本的数据机构和数据类型。我们使用他们进行数据的记录和整理使用。他们提供了一些已经实现的方法，使得我们的操作更为的方便和快速，可以在各个层面上复用。**

* **Middleware**

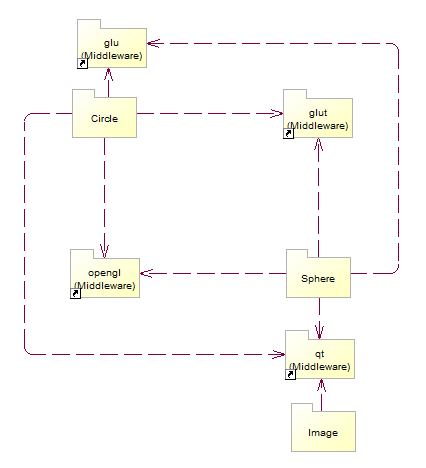
**等**

**说明：**

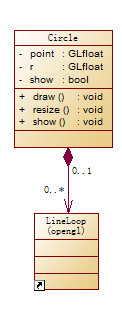
**这些Middleware大部分是提供了一个跨平台的实现接口。使用他们，我们可以忽略平台之间的差异，更为方便的进行跨操作系统的移植等工作。**

**这些包大多是被其他组织或是个人实现的。我们不需要进行设计工作。**

* **Business Services**



**说明：**

**这一层是包含了很多经过我们封装的类。但是这些类可以经过提供给其他的软件开发者或平台直接使用实现复用的价值，和我们的系统是一定程度上分离的。由于需要实现的内容比较简单，而且，由于我们的基础就已经是在glut等经过一层封装的库的基础上实现的，所以这里实现的这是一些较为简单，更为方便使用的组件。并没有包含更多的子系统等内容。**

1. **Circle包：**

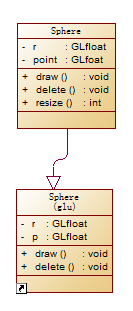
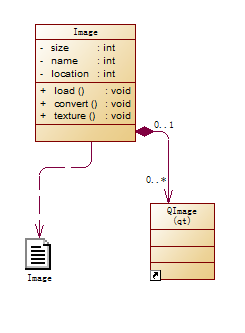
**这个包实现了一个基本的画轨道的功能。通过使用OpenGl中的LineLoop来实现。不同的是，我们需要一个数据源来实现，通过指定的数据进行操作。**

1. **Image包：**

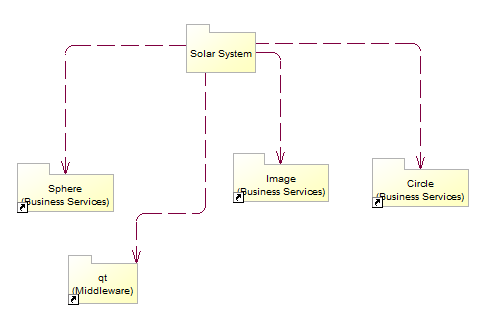
**这个包事实上是一个贴图包，是OpenGl和Qt的整合体。通过调用OpenGl的函数和Qt制作纹理等功能实现了将各个种类的图片（包括Png，jpg等形式）执行成为某个OpenGl物体的纹理。**

1. **Sphere包：**

**这个包是一个绘制球体的包，基本上在GLU的功能上没有实现过多的修改。只是增加了颜色等更多的属性。**



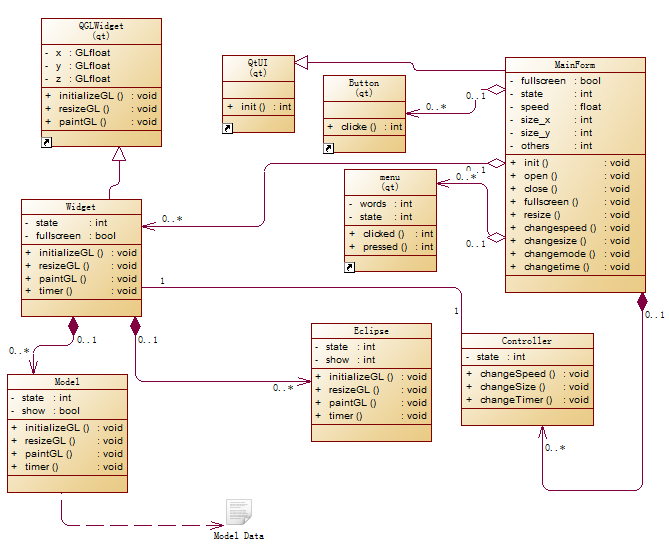
* **Application**



**说明：**

**这一层是和我们的业务逻辑息息相关的类组成的。包含了本系统中特有的，几乎不能实现复用的代码和设计部分。**

***Solar System：***



**内容说明：**

1. **主窗体类继承自Qt的基础类，在基本的部分的基础上添加了很多的组件。例如menu和button等。这些也都是来自于Qt本身提供的类。我们是在他们的基础上进行了创建和使用。**
2. **Controller是本窗体的主要实现类。包含了各种时间的处理逻辑。主要是通过主窗体获取时间然后通过控制Widget实现动态的改变界面展示的内容，从而实现了和用户的交互。**
3. **Widget是主要的显示类。继承自Qt本身和OpenGL组合的GLWidget类，在其中的事件中添加了OpenGL的函数。实现交互使用。它是由2个模型中挑选一个进行更新的。每个模型又都对应了一个相应的元数据文件。通过读取一定的数据进行初始化等的工作。通过控制类进行交互。**

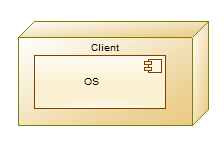
# 进程视图

本程序采用的是单进程该模式，因此不需要进行进程过的设计和开发工作。因此没有进程视图。

# 部署视图

部署试图是从系统软件和硬件配置的角度，描述系统的网络逻辑拓扑机构。模型包含了各个物理节点的硬件和软件配置，网络的逻辑拓扑结构，节点之间的交互和通讯关系等。同时，表达了进程视图中的各个进程和物理节点之间的映射。

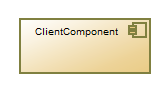
本程序是一个单机版本的程序，没有分布在各个节点上进行。同时，一个单进程的程序。



Client：表示运行本程序的终端。本程序需要可以运行openGL的电脑。

OS：表示客户端运行的操作系统。提供窗口系统，可以让openGL使用 。

# 实现视图



## 概述

ClientComponent:

由于本程序由于基本上是一个演示程序，因此，只提供了一个简单的客户端版本。所以只需要实现一个简单的单机版即可。

## 层

本系统由于实现简单，只是提供了单机版，只是提供了一层的结构。本层中的具体结构可以参看逻辑试图的相关内容。

# 数据视图

本系统保存了一部分天文数据用来加快绘制openGL图形的过程。

这些数据保存在三个文件中：一个是正常情况下的模型的天体运动的参数的文件，一个是日全食的状态下的各个行星的运行位置等参数的状态，还有一个是行星连珠的状态下的各个行星的运动位置和状态的参数。这些参数用于初始化了相关显示参数的缓冲区。数据量都很小，同时容易保存和修改，有益于减少计算加速图形的绘制。