University of Sussex

Final year project

J2ME Planetarium for Mobile Phones

Author:
Simon Fleming

Supervisor:
Dr. Dan Chalmers

June 28, 2008

Statement Of Originality

This report is submitted as part requirement for the degree of Computer Science. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Simon Fleming

Acknowledgements

I would like to thank my supervisor **Dan Chalmers** for his help and encouragement in choosing this project in the summer of 2007. Also for his time and advice on various aspects of the functionalities and complexities throughout the duration of the project.

I am very grateful to **Filip Questier**, met through an astronomy chat group. His advice was invaluable in helping me get to grips with many of the astronomy calculations that were so new to me. Thank you for your time and patience.

I would like to thank the J2ME community at **irc.ubuntu.com** for their help and advice throughout the project.

Finally, thanks to my family (**Olie**, **Kelly** and **Jeananne**) and **Becca** for their patience and support throughout.

Summary

"The slowly evolving map of the constellations has played a multifaceted part in human history. It has fuelled superstition, furthered scientific astronomy, aided navigation, and created a sense of oneness with the universe... The night sky is the oldest shared human experience"

Professor John D. Barrow [2](p. 12)

Since man first gazed at the night sky, the stars and planets have intrigued and captivated. Figures and objects were seen within the patterns of the stars and often mythical stories attached. Over time, these numerous figures and objects became widely known and were named; now used to identify regions of the sky known as constellations.

Today, mathematical calculations exist which mean these constellations can be accurately plotted, allowing star gazers the world over to locate and identify individual objects of interest. No matter where you are in the world, the night sky can it's constellations can be seen, linking us all to the world we inhabit.

This project involves the research, design and implementation of an application which presents the visible constellations and other objects of interest to star gazers on a mobile phone platform, allowing them to navigate and explore their area of the vast night sky.

Contents

Chapter 1

Introduction

1.1 Overview of the Proposed System

1.1.1 Aims and Objectives

The aim of this project is to create a Java 2 Micro Edition (J2ME) application for displaying the stars and constellations on mobile phones. Programs currently exist primarily in the form of desktop applications and web applets, which pose obvious restrictions to viewers. Mobile phones are both available and accessible; providing the perfect tool for aiding budding star gazers. The application will show the night sky as seen from a specific location on earth, focusing primarily on; constellations; more recognisable individual stars; and other objects of interest.

1.1.2 Target Users

The proposed system is intended to attract and encourage interest in the stars from amateurs; users who have little or no access to star gazing equipment. Users could be of any age, gender and geographical location with a fairly modern, Java enabled mobile phone with Internet access and a colour screen.

1.1.3 Problem Areas

The main challenges associated with this project are:

1. *Limitations with mobile phones* - Offers a very limited size display to show a vast area of sky, limited computation abilities and storage, and limited control options.
2. *The research and implementation of astronomy mechanics* - Starting with a limited knowledge of astronomy, time will need to be invested to isolate and understand the associated calculations, representations and data sources.

1.1.4 Report Structure

Professional considerations are first examined, followed by a full requirements analysis and specification. A high-level overview and in depth design of the system follows. Next, details regarding the implementation and testing of the finished application are presented. A conclusion discussing reflections and future work finishes the report.

Chapter 2

Professional considerations

Many potential employers in the computing industry have their own set of guidelines. So to analyse and identify relevant sections of a professional set of standards will be a useful task.

2.1 Code of Conduct

The British Computer Society (BCS) [32] is the leading professional body for those working in IT, hence their code of conduct will provide the most useful reference point.

2.1.1 Professional Competence and Integrity

This is the main section of the Code of Conduct that is relevant to my project.

Point 14 states: *"You shall seek to upgrade your professional knowledge and skill, and shall maintain awareness of technological developments, procedures and standards which are relevant to your field..."*

This compliments my choice to conduct the project using J2ME; although Java is already known to an advanced level, the new techniques and practices required to create a working application using this subset of Java will be a great addition to my existing programming skill set. To research and implement the newer additions to J2ME, such as the Location API, will help me to gain awareness of the current J2ME technological developments.

Point 15 states: *"You shall not claim any level of competence that you do not possess. You shall only offer to do work or provide a service that is within your professional competence."*

With solid experience and understanding of Java; learning the new language of J2ME should prove relatively straight-forward. The challenging aspect will be learning the key mechanisms which relate to astronomy. However, I will be investing a great deal of time into this area, and believe that a correct implementation of the finished application is definitely achievable.

2.2 Code of Good Practice

The Code of Good Practice [33] is intended for BCS members working in IT, supplying or using IT systems or services. It is not mandatory, however members are expected to be familiar with its contents and are advised to follow any relevant practices.

The sections on Methods and Tools, Requirements Analysis, Specification and Software Development will fall inside the scope of my project and this document will be a useful reference tool during software development, particularly when conducting the analysis and design phases.

Chapter 3

Requirements Analysis

The aim of this section is to examine all relevant topics to gain a thorough understanding of the mechanics, requirements and capabilities for the finished application. Both the technical and theoretical aspects of the project will be evaluated.

3.1 The J2ME Platform

3.1.1 Overview

J2ME is a software development platform specifically designed to operate on small devices (created by Sun Microsystems). Given the wide range of such devices and their varying hardware capabilities it would be impossible to take full advantage of this on one single platform. J2ME is made from a number of specifications and technologies for use with the subset of devices within the domain.

The available subset for a given device is defined by one or many *profiles* which advance the capabilities of a *configuration*. The profile(s) and configuration are dependent on the hardware within a device and also the intended end users needs [38](p. 4).

3.1.2 Configurations

A *configuration* is a specification used to define a software environment for a range of devices; each shares a set of characteristics that must be present to operate. For example: [38](p. 4):

- The type and quantity of memory available.
- The processor type and speed.
- The type of network connection available to the device.

Small device vendors are required to implement a configuration specification in full, such that developers can rely on a consistent programming environment. *Configurations* consist of a Java Virtual Machine (VM) and a collection of Java classes which provide an environment for running application software.

J2ME currently defines two possible configurations [38](p. 5):

1. *Connected Limited Device Configuration (CLDC)*
   Aimed at the low end of the consumer electronics range, a CLDC platform may be a mobile phone or PDA with around 512KB of available memory.
2. *Connected Device Configuration (CDC)*
   CDC addresses the requirements for devices which are in between CLDC and a full desktop system running Java 2 Standard Edition (J2SE). Typically having a higher memory capacity at around 2 MB or more, and more powerful processors, CDC is found on the higher-end of PDAs and mobile phones, set-top boxes and web telephones.

As this project is aimed at mobile phones the CLDC configuration will be the assumed Java environment available to the application throughout the project. I will be using CLDC version 1.1 which supports floating point operations and the associated classes and methods [37]; these will be required for astronomy calculations.

3.1.3 Profiles

Profiles build on a configuration through additional classes providing a more complete development platform. The profile specific to mobile phones is the *Mobile Information Device Profile (MIDP)*. This adds networking, user interface components, and local storage to the CLDC configuration [38](p. 7). MIDP 2.0 includes a game API which provides additional support for user interfaces and game graphics. The advancement in graphical capabilities will enhance the possibilities for representation of the night sky in this project.

3.2 Astronomical Research

A complete and thorough understanding of the main principles, calculations and representations of the constellations and stars is key to the success of this project.

3.2.1 Celestial Sphere

"Everything in space is so far away that all sense of distance is lost, and the stars, planets, Moon and Sun appear to be tacked upon this illusory bowl..." [14](p. 14)

The Celestial Sphere [24] will need to be represented within the proposed system in order to plot stars and other objects of interest as they appear in the night sky. Please see appendix 11.1.1 for further details and illustration.

3.2.2 Stars and Constellations

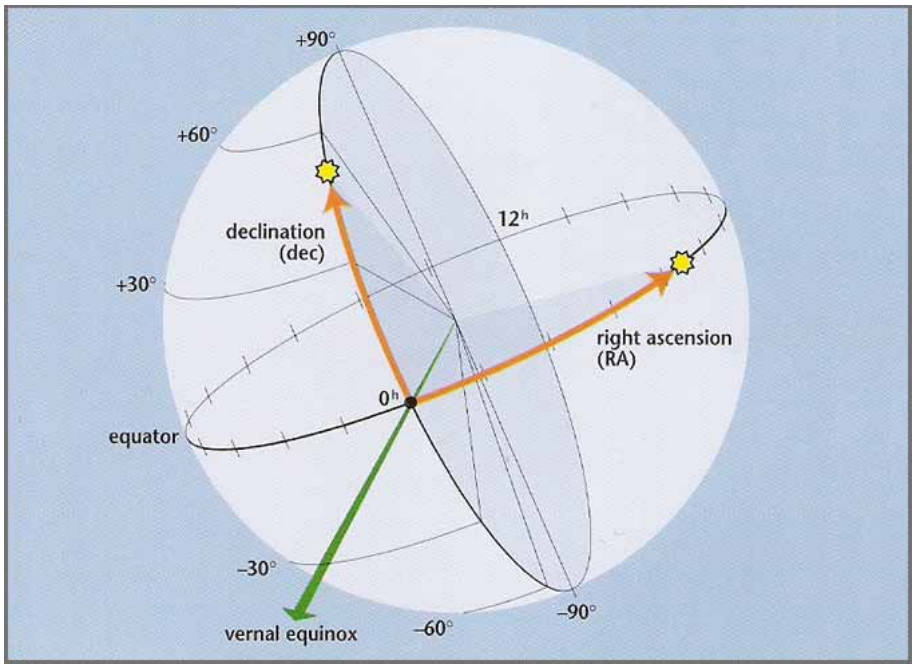
The Stars

Stars are distant objects of varying magnitude, like our Sun, each of which appear to be attached to the Celestial Sphere at a specific location. Those stars with a magnitude greater than 6 will not be included as these are not visible to the naked eye [4], also helping to restrict the amount of information that needs to be stored alongside the application (appendix 11.1.4 for more detail).

Stars are positioned on the Celestial Sphere by means of two equatorial co-ordinates: right ascension and declination, measured for a given instance of time known as an Epoch.

From a given location on earth specified by Longitude and Latitude and a specific date and time, the Equatorial coordinate system can be converted into Horizon Coordinates (altitude and azimuth

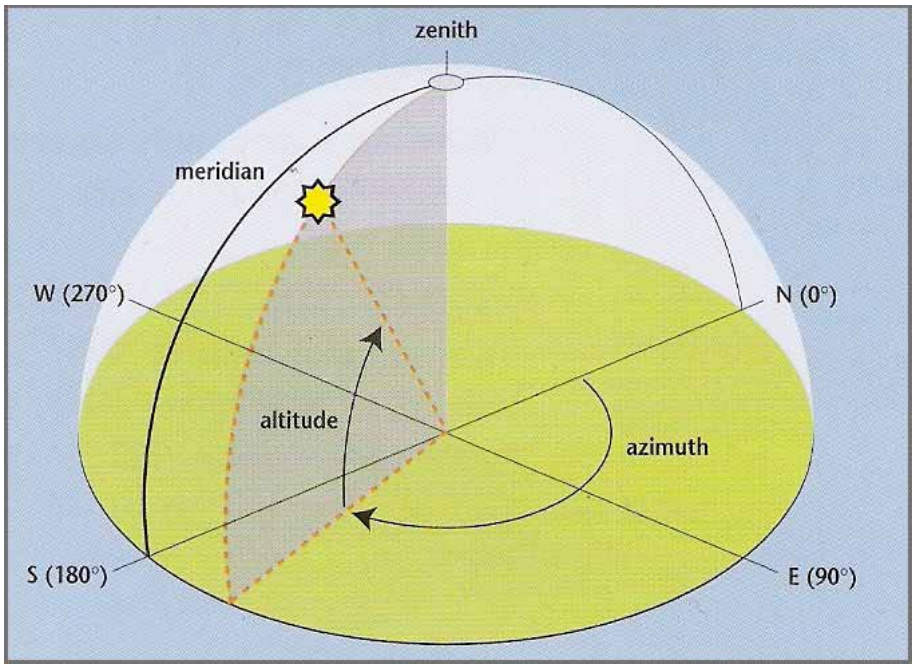
Figure 3.1: Equatorial coordinate system: Right Ascension and Declination Diagram [10]



positions) (see Figure (3.2)). As the proposed system will be presenting a view of the night sky, this translation will be useful. (A detailed description of the algorithm used to plot the position of a star from its right ascension and declination in conjunction with a location and date/time can be seen in appendix 11.1.2)

The algorithm used to convert equatorial coordinates to horizontal coordinates makes use of inverse trigonometry functions which are not built into the Math class within J2ME. Fortunately, an open source J2ME class including inverse trigonometry methods was found [15].

Figure 3.2: Horizon coordinate system: Altitude and Azimuth Diagram [10]



The Constellations

Constellations are named patterns of stars that resemble or represent a particular god, animal or person from history or legend. There are 88 officially recognised star constellations [14](p. 94) which can be found in Appendix 11.1.5. Stars in a Constellation are connected via straight lines; although imaginary, these lines will be represented in some form within the application. The imaginary lines are only drawn between *some* of the stars within a constellation: there are no official standards.

3.2.3 Projections

The Celestial Sphere will need to be represented on a flat surface; a 2D mobile phone screen. The Celestial Sphere has its own meridians of longitude and parallels of latitude so the principles of map projection can be applied. From careful investigation into the types of projection available, I believe the most suitable category is the *Zenithal* projection, which represents a sphere as a circle. Calculations can now be made to plot points on the plane from a sphere. Appendix 11.1.7 provides further details on this.

3.2.4 Times and Dates

All calculations of the positions of stars and other objects of interest in the sky require the correct date and time: these can easily be obtained from the J2ME platform [38] (page. 24). However, to make them of any use to the calculations, they need to be converted to a number of different formats (sidereal time and the Julian data), calculated using well known algorithms [19](p. 409).

3.2.5 Orbital Elements

Due to their proximity; planets, comets and satellites do not appear fixed to the celestial sphere, but move against the background of stars. These can only be plotted by using up-to-date information. Any calculations involving the positions of such objects will require the use of remote data (appendix 11.1.8 provides further details).

Orbital elements can be broken into two broad categories based on the orbit that they follow:

- **Elliptic Orbits**
  The planets, moon and some comets and satellites have elliptic orbits. These orbits are circular and are specified by an *eccentricity* which describes the shape of the orbit. A well known algorithm can be followed to calculate an apparent position for an object following an elliptic orbit (see appendix 11.1.3).
- **Parabolic Orbits**
  Some comets fall into the parabolic orbit category, these orbits require additional calculations to plot correctly.

3.2.6 Viewing Conditions

It would be useful to aid users by informing them when the best times for star gazing are approaching. The following factors affect viewing conditions:

1. **Light Pollution** - areas with large sources of light such as frequent street lamps will inhibit visibility. Yet without using some kind of international database system, this information is unobtainable. Functionality to adjust the displayed stars based on magnitude can be included within the application to aid perception.
2. **New Moon** - as a new moon approaches, less light is generated which aids visibility. Including the position and phases of the moon would be a very helpful addition.
3. **Weather** - Information on weather which may hinder or make viewing uncomfortable, cloud cover or extreme wind for example, will be a valuable addition.

3.3 Data Sets

3.3.1 Stars and Constellations

Having decided that electronic data was required due to the vast array of stars and constellations visible with the naked eye, I managed to find a site offering hundreds of different electronic star catalogues.

I selected the *Bright Star Catalogue, 5th Revised Ed. (Hoffleit+, 1991).* It was chosen from the *The VizieR Catalogue Service* [9] because; it allowed for custom exports of the data to a specified format; contained all of the stars visible with the naked eye; could be downloaded; and included information regarding star colour spectrums. The right ascension and declination for the J2000.0 Epoch was also included which provides all of the information required for plotting the visible stars. To make use of the data a parser was written in Java which iterates through the files, recording fields of interest in local variables and creating an Object which houses the important details for every star.

Unfortunately there seems to be no convention for the constellation lines; this meant that finding a data source was very time consuming. Eventually I was able to find a list of constellation lines directly from `www.skymaps.com`, however this referenced Tycho catalog numbers [28]. Another

parser was created to convert Tycho numbers to Henry Draper Catalog Number using the data from a third source [12] which contains a list of stars with both sets of numbers. From my investigation into the various line files in the same format available from this page, I chose constellation Figures based on designs originated by H. A. Rey and published in "The Stars: A New Way to See Them" [30] which appears to be the most accurate.

3.3.2 Comets

Harvard University supplies up-to-date information on the major comets currently visible [39]. An algorithm can be used to accurately plot a comets position (see appendix 11.1.3).

3.3.3 Astronomy News Feeds

Many relevant sites provide a Rich Site Summary (RSS) news feed supplying information about upcoming celestial events as well as general astronomy news. One particular website `www.astronomy.com` [1] provides a feed which could be supplied directly to the application running on the mobile phone.

3.3.4 Yahoo! Weather API

Yahoo! Weather [41] is an RSS feed providing weather information for most locations in the world. Data is obtained by making a request to Yahoo! Weather using either a US zip code or a Location ID and the result is an Extensible Markup Language (XML) document. The XML document has a strict format which can be parsed to obtain data of interest such as; sunrise and sunset times at the specified location; current weather conditions; a simplistic five day forecast.

An open source PHP script [40] was found which can perform the communication with Yahoo! Weather to extract required information about a given location. The script also uses caching to avoid multiple requests to obtain the same location details. The free open source script was downloaded, installed and slightly modified to parse the RSS feed document to extract and store the useful pieces of information.

The Yahoo! Weather RSS feed will require a complete list of Location ID's which will need to be parsed and saved from the available locations [41].

Yahoo! also provides a web service called ZoneTag [42]. Yahoo! Zonetag offers, via a web service, a translation from a mobile phones radio signal properties to a geographical location. Following extensive research and testing of this service it was apparent that key information was not available for UK locations and also that the data was not moderated; contradicting sets of results can easily be obtained. For these reasons I will not be taking investigation further.

3.4 Domain Analysis

Identifying and examining existing applications which perform similar tasks will help inform the design of the proposed application. Successes and desirable features can be included or considered, while failures and problems can be avoided.

There are hundreds of desktop, applet and PDA applications which display a view of the night sky based on a location. I have chosen the applications that I believe to be most relevant to the application I wish to develop. A table indicating a features summary of all reviewed applications can be seen in Figure 3.3. Further details of the domain analysis can be seen in appendix 11.2.1.

3.4.1 Application Features Summary

-	PicoSky [18]	MobPlanetarium [34]	Stellarium [36]	Proposed System
Magnitude Shown	4.0	6.0	huge	6.0
Sun	✔	✔	✔	✔
Moon	✔	✔	✔	✔
Constellations	all 88	all 88	all 88	all 88
Online Updates				✔
News				✔
Red Eye Mode		✔		✔
Weather				✔
Comets				✔
Search	✔	✔	✔	✔
Time Controls		✔	✔	

Figure 3.3: Summary of reviewed applications

3.5 Use Case Model

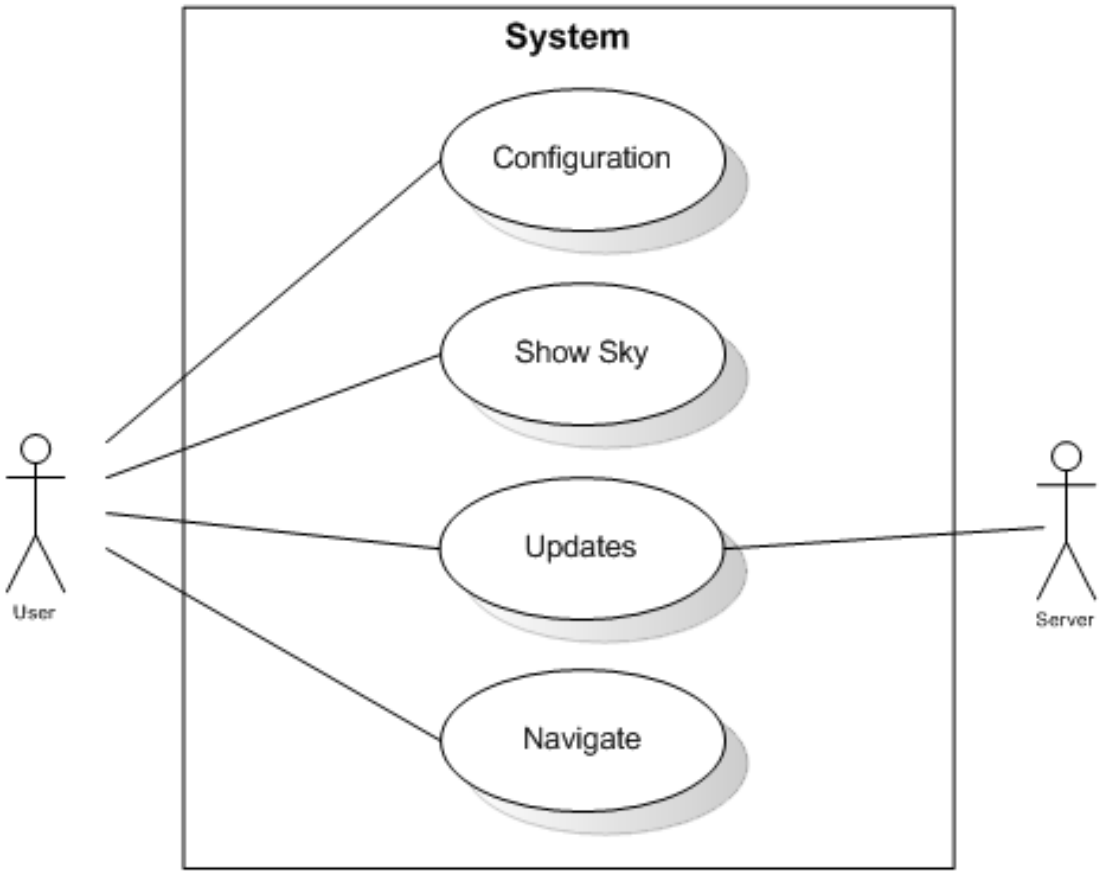
3.5.1 Overview

Developing use-cases helps to identify how end-users will interact with the system under a specific set of circumstances, depicting the software from the end-users point of view. The process is crucial as it helps to identify problems; displaying requirements without the complication of the actual implementation details [27](p.191). The Unified Modelling Language (UML) is used in this section to show the Use Case diagrams.

3.5.2 High-Level Use Case

The following High-Level Use Case Diagram (Figure 3.4) provides a complete view of the proposed system functions available to an end user.

Figure 3.4: High-Level Use Case Diagram



3.5.3 Low-Level Use Cases

The High-Level Use Case diagram can be broken down further into a set of Low-Level Use-Cases that provide a detailed description of elements of the proposed system. Use Cases are often written informally, however a recommended template has been used [27](p. 194).

Low-Level Use Case 1

Use-case: *Initial Startup*

Primary Actor: End User.

Goal: Users local latitude, longitude, date and time are correctly saved.

Preconditions: System allows data entry.

Trigger: User attempts to run the application for the first time.

Scenario:

1. User: selects that they wish to run the application.
2. System: loads the application and prompts the user for their longitude and latitude.
3. User: Provides longitude and latitude details.
4. System: System stores longitude and latitude details, prompts user for current local time.
5. User: Provides local time.
6. System: System stores local time details, prompts user for current date.
7. User: Provides current date.
8. System: System stores current date details, forwards user to the main menu.

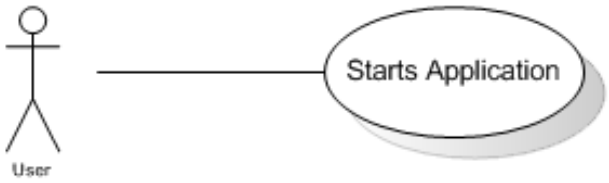
Exceptions:

1. User provides invalid data: provide error message and prompt for re-entry.
2. System is unable to store user selections: provide error message and attempt to resolve.

Open Issues:

1. How will a user longitude and latitude be collected?
2. Suitable time and date formats must be decided on to maintain system consistency.

Use Case Diagram:



Low-Level Use Case 2

Use-case: *Request Sky View*

Primary Actor: End User.

Goal: User is presented with a view of the night sky for their current location / date / time.

Preconditions: System has been programmed to render a sky view from a given location / date and time.

Trigger: User requests to view the current night sky from the main menu.

Scenario:

1. User: selects that they wish to view the sky view from the main menu.
2. System: Hides the main menu and loads required data into memory. Loading bar displayed to user. System presents user with their correct sky view.

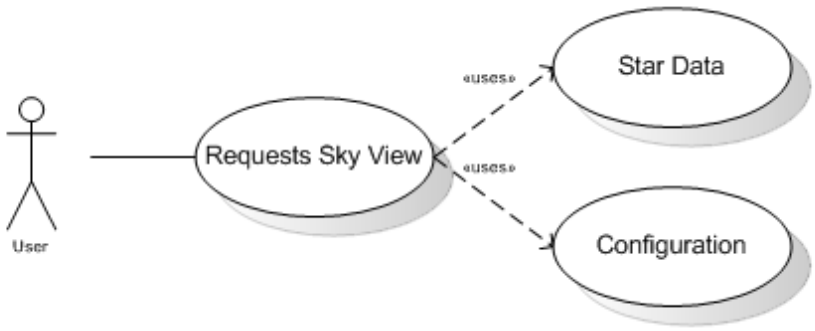
Exceptions:

1. System fails to load required resources: System to present user with suitable error message and attempt to resolve.
2. User selected incorrect location, date or time: System to provide quick route to system settings to allow user to make corrections.

Open Issues:

1. How will the initial sky view be presented?
2. Quick route to system settings needs to be decided.

Use Case Diagram:



Low-Level Use Case 3

Use-case: *Configuration Update*

Primary Actor: End User.

Goal: User successfully updates configuration details.

Preconditions: System provides menu option for configuration and allows user to update stored details.

Trigger: User requests to update configuration options from the main menu.

Scenario:

1. User: Selects a configuration option.
2. System: Loads appropriate user input form to obtain data.
3. User: User provides required data and confirms.
4. System: System saves the information and provides an indication that the update was successful, returning to the configuration menu.

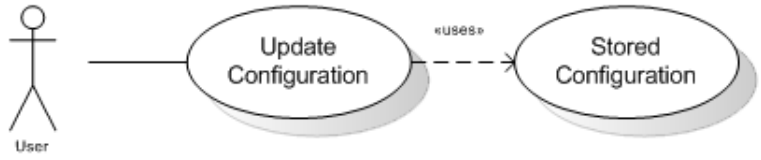
Exceptions:

1. User provides invalid input: system to alert user of error and again display the input form.
2. System fails to save data: user to be informed of error.

Open Issues:

1. How should data be formatted, what are the legal inputs?
2. Which configuration options should be available to update?
3. How will the configuration be stored?

Use Case Diagram:



Low-Level Use Case 4

Use-case: *Online Updates*

Primary Actor: End User.

Goal: User successfully retrieves relevant remote data update resulting in useful up-to-date information.

Preconditions: System provides a manual request option from the main menu to perform an update.

Trigger: Automatic or manual request to perform update.

Scenario:

1. User: Requests update from the main menu.
2. System: Packages up local configuration data and sends to the server.
3. System: Receives update and displays results to user.

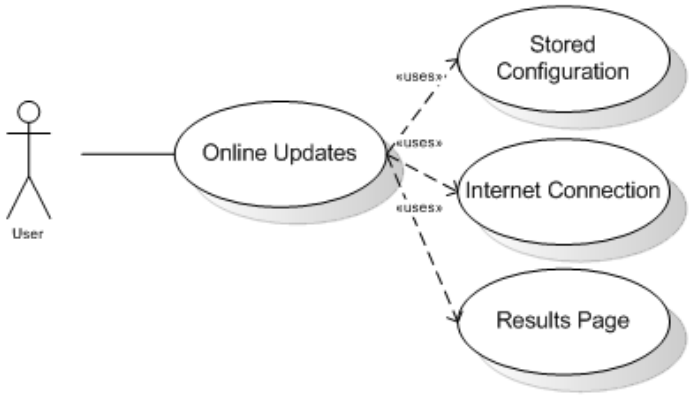
Exceptions:

1. System unable to contact server: System to inform user with useful error message.
2. Server receives invalid data: Server to send useful error message in response.

Open Issues:

1. What information will be available via the online update?
2. How will results be displayed, used and stored?

Use Case Diagram:



3.6 Questionnaire

A questionnaire was designed and implemented using PHP and MySQL to gain a better understanding of the key issues relating to user expectations [13]. The questionnaire was fully accessible in all browsers due to its use of XHTML and CSS standards compliant structure as recommended [26].

An online questionnaire has the following advantages: [26]

- Responses are usually received quickly.
- Copying and postage costs are low or non-existent.
- Data can be transferred immediately into a database for analysis.
- The time required for data analysis is reduced.
- Errors in questionnaire design can be corrected easily.

As the proposed system is targeted at a very broad range of users, the questionnaire was distributed by sending an email link to friends and family which quickly spread. The questionnaire received a total of **40** responses and the complete results can be seen in appendix 11.2.3. Some of the free form comments from users can be seen in appendix 11.2.4.

The most important information established from the questionnaire is listed below:

Figure 3.5: Questionnaire Results

The results of the questionnaire will help to inform both the requirements specification and the prototype.

Chapter 4

Requirements Specification

4.1 Functional Requirements

The functional requirements are split into two categories; core features that the system should implement and extended features that will be added to the system if possible.

4.1.1 Core Features

1. Create a view of the night sky from a given position on earth on mobile phone screen.
2. Draw constellation lines between stars.
3. Provide the ability to navigate this information by using zoom, rotation and selection options.
4. Introduce a server application that will provide the *framework* to feed data to roaming users.
5. Ability for the date and time to be set.

4.1.2 Extended Features

After understanding, designing and implementing the Core Features it would be interesting and constructive to improve the system to give a more complete and useful application. It is unlikely that there will be enough scope to address all extensions, however having a range now will give flexibility so that as the problem space unfolds and is better understood, appropriate and achievable extensions can be selected.

1. Location Environment Update

Providing up-to-date information detailing upcoming weather predictions would assist in selecting the best times for viewing the night sky. Yahoo! weather can be used to supply this information to users.

2. Events in the Night Sky

Information regarding upcoming celestial events such as visible comets via online updates would be an interesting enhancement, and one which is likely to encourage use of the product. Further to

this, attempting to access the phone alarm/alert mechanisms to alert users when good star gazing times approach would be a great feature.

3. Planets, Moon and Sun Positions

Including the position of the Planets, Moon and Sun within the views of the sky would be an interesting bonus, and may assist users by aiding navigation towards the constellations of interest. The current phase of the moon could also be included here.

4. Online Community

Creating a community for users of the application including such features as image galleries and forums would add an extra dimension to the application.

5. Introduce GPS/Compass Mapping

Although potentially very difficult, allowing a user to align themselves with the applications view via GPS/Compass mapping would be a unique advancement which would increase usability greatly. It is well worth investigating new and upcoming J2ME technologies for possible solutions.

6. A Game

Adding some interactivity to the application in the form of a game would improve enjoyment for users and may broaden its appeal.

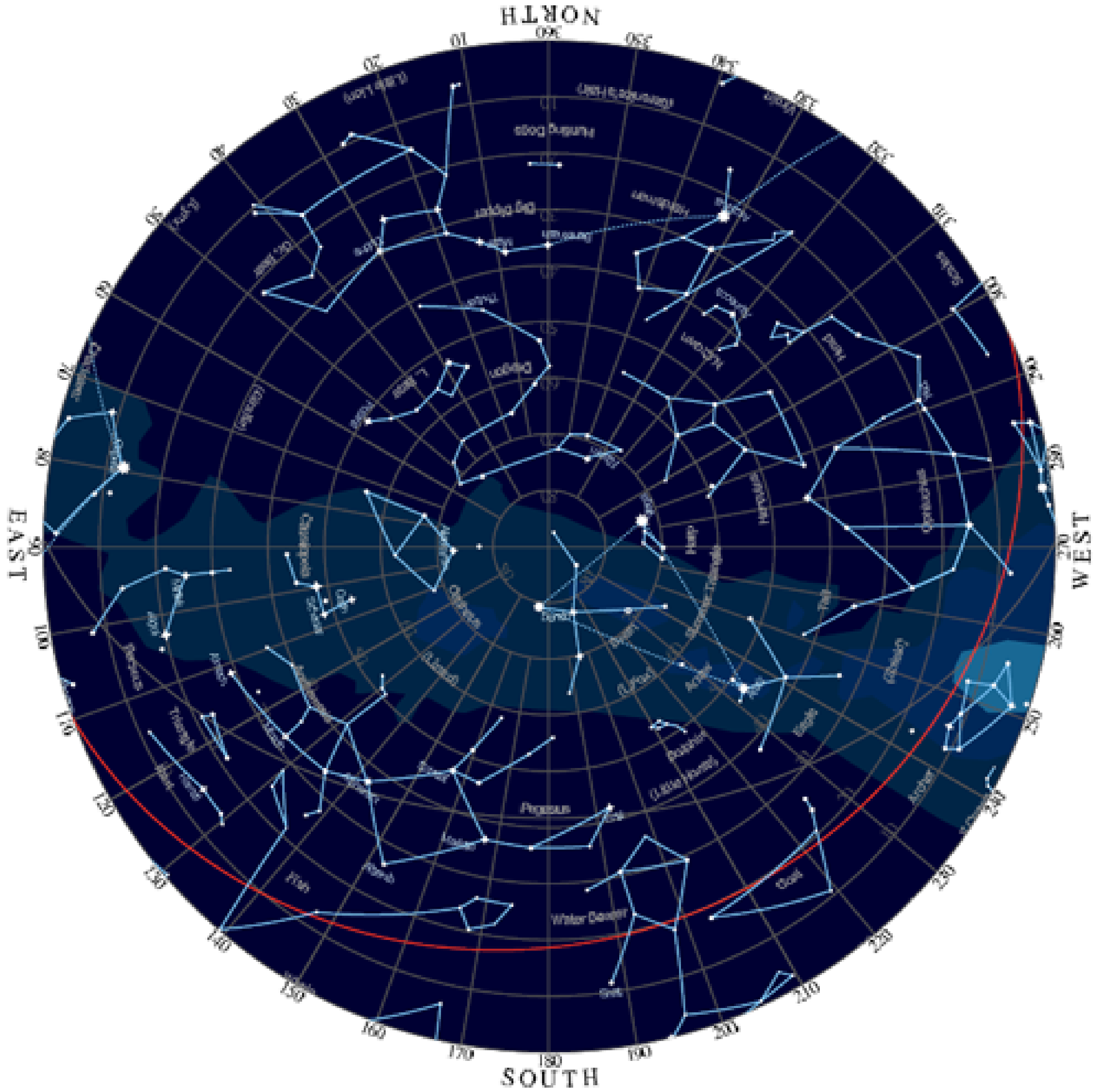
4.2 Non-Functional Requirements

1. Must allow a new user to easily navigate around a current night sky view.
2. Must have reasonable loading times.
3. Aid perception by adding realism to the application, for example a horizon.
4. Storage and file size of the final application must not be excessive and should be in line with similar applications e.g. J2ME games.
5. Experienced users must be able to use the application effectively.
6. Suitable colours, font sizes, layout, menus and program flows should be used.
7. Data transfer volume must be kept to a minimum.

4.3 Application look and feel

Having spent a great deal of time researching various night sky representations, I came across an electronic document which included an image of a star map which has stuck in my mind [43].

I would like the application to resemble this image. I particularly like the representation of the milky way and the use of colours.



4.4 Testing Overview

There are a number of possible approaches to testing the proposed system.

Firstly, in order to test the inner workings of the application, key algorithms should be checked against known results to ensure that correct results are being obtained. Testing resources indicating correct mappings from inputs to outputs for key algorithms are available [19].

To test that the rendered night sky is acceptable, third party applications can be used to compare celestial object positions. Field results could also be conducted; testing the application outside to see how the actual night sky can be compared to the rendered version.

Chapter 5

Prototyping

5.1 Low-fidelity Prototype

For initial testing, low-fidelity prototypes are useful as they are cheap, simple and quick to produce. Valuable thoughts and suggestions can also be obtained by asking users what they expect and how they might do things differently [26].

I constructed a low-fidelity prototype using paper, acetate and example screens; this can be seen in the appendix of the interim report. The prototype was tested on three potential users, the key findings were:

Aesthetics Users felt that they would like to be able to navigate through the various constellations and stars at the click of a button. Users were very keen to have the phone automatically align itself with the sky view like a compass. Selected stars and constellations should be highlighted in obvious colours; they would like a sense of realism within the render of the night sky. Users would like clear fonts which are easily legible and large enough for older users to read.

Star Gazing Users would like to be presented with a complete view of the night sky and then be allowed to navigate around it, looking at objects of interest.

Initial Setup Users felt that having to confirm the time and date was irrelevant; the time and date as seen on the mobile phone screen while making calls should be assumed correct. This step could then be avoided.

5.2 J2ME Exploration

It was important at an early stage to make sure that key formulae, graphics and controls I was expecting to use would actually work in J2ME. The main findings and open issues from this prototyping stage can be seem below.

1. **Emulation** - In order to create J2ME applications without running the actual program on a mobile phone, a J2ME emulator was used. This emulator was part of the Sony Ericsson Software Development Kit (SDK) [11].
2. **Deployment** - The Sony Ericsson SDK allows a one click deployment of an application to a mobile phone which is connected via USB cable. The main difficultly was that the device would only accept applications which had a very specific configuration file definition. Unfortunately, the default settings for this configuration file were not compatible with my device. Once understood, this difficulty was overcome.

3. **Drawing** - Tests were conducted early on with J2ME to see how drawing basic shapes such as circles and squares was performed; this is straightforward and almost identical to the methods used within J2SE. During the testing phase, a circle was drawn and points were plotted on the circle using trigonometry principles. This method worked correctly without any obvious delay with floating point operations.
4. **Controls** - J2ME offers a built in function which is automatically called when a button on the device is pressed. The function **public void keyPressed(int code)** includes the key code of the button pressed and a simple switch statement can be used to perform different tasks for different key presses. To find out the key code a simple program was written to monitor the values generated from the available buttons (results can be seen in appendix 11.2.5).
5. **The Internet** - Using the *javax.microedition.io.HttpConnection* class to make an HTTP request class within J2ME it is a straightforward process. J2ME was tested to discover the process of requesting a URL and receiving a response.

Chapter 6

High-Level Design

A J2ME prototype was created from the system overview during the requirements analysis phase (see appendix 11.3), to allow for evaluation and testing of the system to help inform the design. The following section expands on the system overview and attempts to focus on clear cohesion and coupling as suggested as good practice for object orientated design by Yourdon and Constantine (1979) [31] (p. 382). Cohesion is a measure of the level at which methods and code within a module interact to perform a specific task. The design will aim for high cohesion whenever possible. Coupling is the degree that each module relies on the other modules within the system. Where possible, the design will aim to keep coupling to a bare minimum.

6.1 Behavioral Model

Representing the system in terms of times, events and states will be a useful task to identify how it will respond to external stimuli [27](p. 249). These follow from the use-cases presented in the requirements analysis phase.

6.1.1 State Diagrams

The following state diagrams represent the system from the outside as it performs a task.

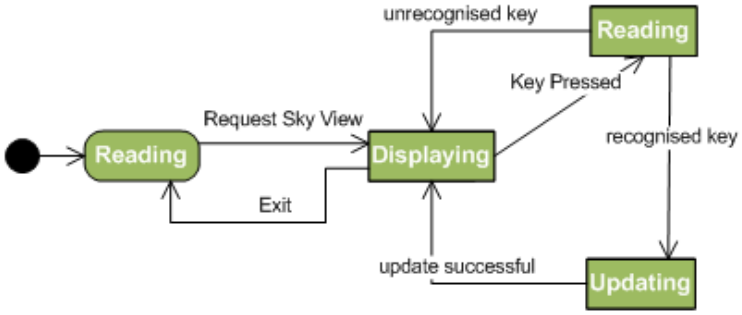


Figure 6.1: Request Sky View

Figure 6.2: Initial Setup/Configuration Update

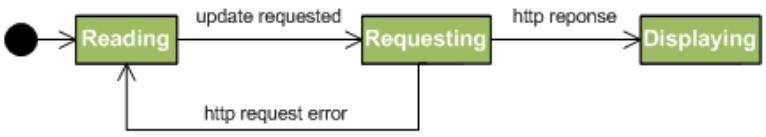


Figure 6.3: Requesting an update

6.1.2 Sequence Diagrams

The following diagrams represent how events cause transitions from object to object within the system [27](p. 251).

Sequence Diagram: Request Sky View

This shows how a star will be plotted; it will be a iterative process, happening once for every visible star held within the system (see Figure 6.4).

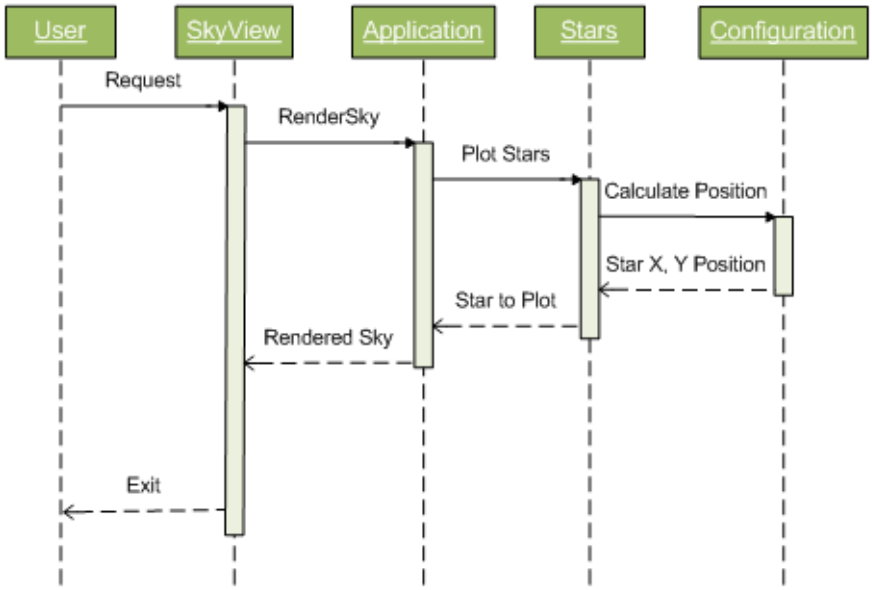


Figure 6.4: Request Sky View

Sequence Diagram: Initial Setup/Configuration Update

As the initial setup and configuration update involve a number of pieces of data to be supplied by the user, this diagram represents the process that will take place a finite number of times (see Figure 6.5).

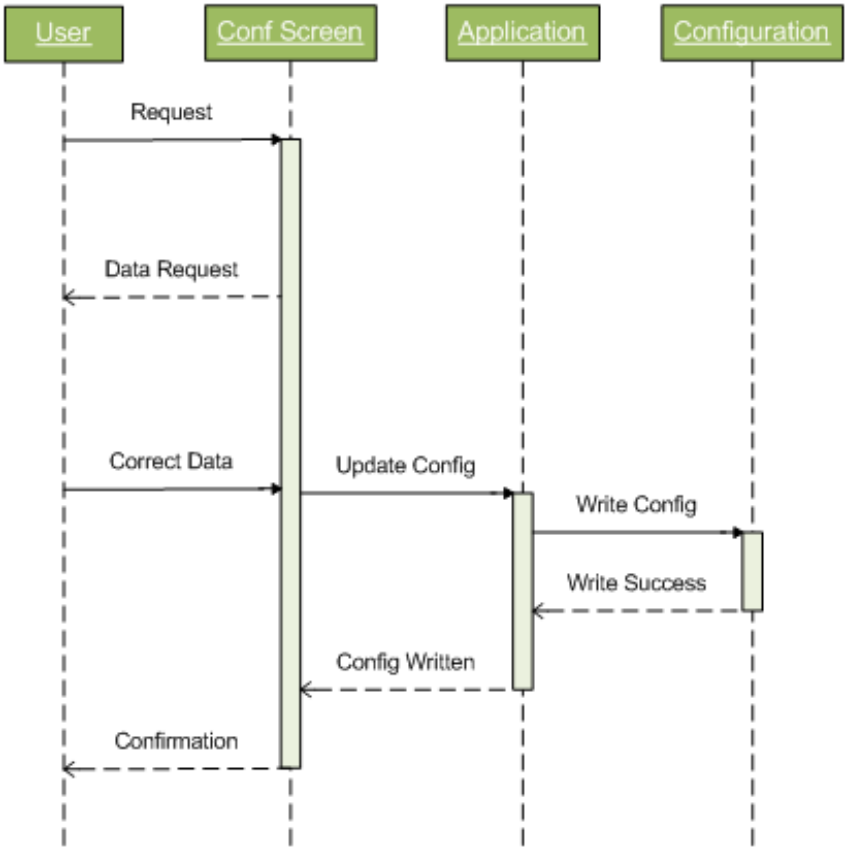


Figure 6.5: Initial Setup/Configuration Update

Sequence Diagram: Requesting an update

This shows the updating process made at the client, see Figure 6.6.

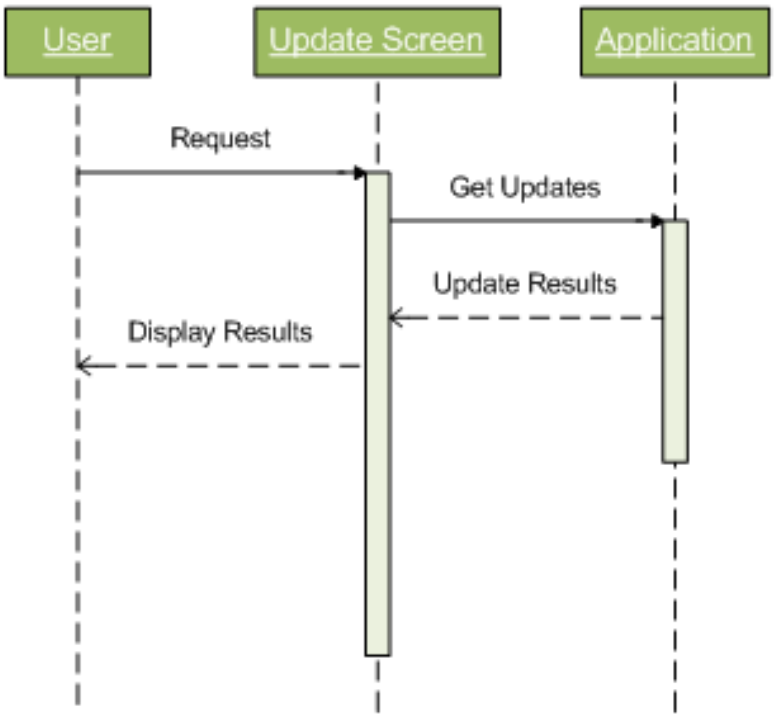


Figure 6.6: Requesting an update

6.2 Resource Files

Resource files will be used to load and store system objects such as star data and constellation figure lines. They are included within a J2ME application resource folder, fully available for reading and parsing at run time.

The initial prototype directly influenced the decision to use files for storage rather than static class variables as resource files can be easily updated and have less impact on total application file size than static data.

Due to loading and parsing the cost of running the application will rise slightly, but this is insignificant in relation to the benefits from the use of resource files. Although with implementation and testing of this method the loading time increased noticeably, overall this method saved greatly on storage space (by 265%, 845KB to 319KB). Please see appendix 11.7 for more details.

Originally I had planned to use static data, but problems arose with both star data and constellation lines. Star data was initially defined as static data arrays within class files. J2ME has a maximum of 32KB class files, this resulted in star data being split among nine separate files (Stars1..Stars9). This created a huge increase in file size (845KB as highlighted above).

Constellation lines were defined in terms of references to the nine separate static star data arrays, which has the advantage of points being already calculated as soon as stars are plotted. Searching each static array for the required constellation line links became very troublesome and made the system functions very unintuitive and long winded.

I decided that this form of data storage was inappropriate and that the storage policy would have to change; resulting in a complete rebuild of the application based around resource files (please see appendix 11.7 for further details).

Following a conversation with Dan Chalmers the increased overhead was reduced by loading fainter stars in the background while a user interacts with the application, reducing the noticeable loading time significantly.

6.2.1 Geographical Locations

Possible locations will also be stored in a resource file. Locations will be loaded when either the application is run for the first time when a user must select their location, or when a user wishes to change their location. This file will store Locations by Country, Town, Longitude and Latitude. A suitable initial dataset was located [29].

Following research of existing star maps, in particular the BBC Sky at Night newsletter, the possible locations a user may select has been heavily restricted; the idea is that a user will simply choose their *closest* geographical location. This decision was made to improve the user experience during setup as longitude and latitude values will not be needed at this stage; information which a novice star gazer simply *cannot be expected to know*. To decrease the severity of this issue, new functionality to enter a new location (including specific longitude and latitude details) will be implemented within the systems settings for more advanced users.

6.2.2 Constellation Figure Lines

Constellation Figure lines will be stored as point to point lines within the system. Points will be stored as right ascension and declination positions and converted to X,Y coordinates in the same manner as stars. This is because if constellation lines are dependent on the static star array indexes (as first intended), all stars would need to be present while using the system. The chosen method allows for non-visible stars to be ignored following examination, thus freeing additional memory

at runtime. The array index method also required a great deal of storage (see appendix 11.7) and additional methods and logic for managing drawing and moving between star positions. Point to Point lines require only the logic of drawing lines between two defined points. The prototype implementation which used array indexes for constellation figure lines can be utilised to produce the new required format by creating a file with the right ascension and declination coordinates of the indexed stars for each constellation.

6.3 J2ME non-volatile storage (Record Stores)

Non-volatile storage is essential for a user configuration to be stored and loaded. Without this, the user would have to specify settings every time the application was run.

Following extensive research, **javax.microedition.rms.RecordStore** was identified as the key mechanism for non-volatile storage. Record Stores provide the ability for a J2ME application to create/write a named collection of variables.

User preferences such as projection, colour, longitude and latitude, can all be recorded in this manner. A prefix identifying the particular variable is used to ensure that while enumerating a Record Store the appropriate value is assigned to the correct user setting. For example, a *hash* prefix will be used to identify a users city. For more information on record store prefixes, please see appendix 11.8. The Record Store can be checked to see if a user already has a saved configuration, this is useful for identifying the first launch of the application.

Other alternatives that were rejected included:

- **HTTP to server**. This option would manage user settings as string values, however there would be additional cost for saving and collecting user settings and they would only be available when a user has an active connection. The user settings are required for operation of the application and as such need to be stored on the device.
- **JSR 75** [21]. This mandatory package for J2ME allows for non-volatile storage, however rather than allowing for string values to be stored it focuses specifically on media such as audio, pictures and videos. Additionally special user permission is required for writing to files.

6.4 Client Application

The client application is responsible for providing the core requirements to the end user.

6.5 Packages

To improve organisation of the Client application, packages will be used to group together similar entities. Working with a large collection of class files each with its own set of methods/variables can become very difficult; packaging is the logical solution to improve the manageability and readability of the system. It will also help to identify any cohesion and coupling issues.

The client is represented by a collection of packages (Figure: 6.7).

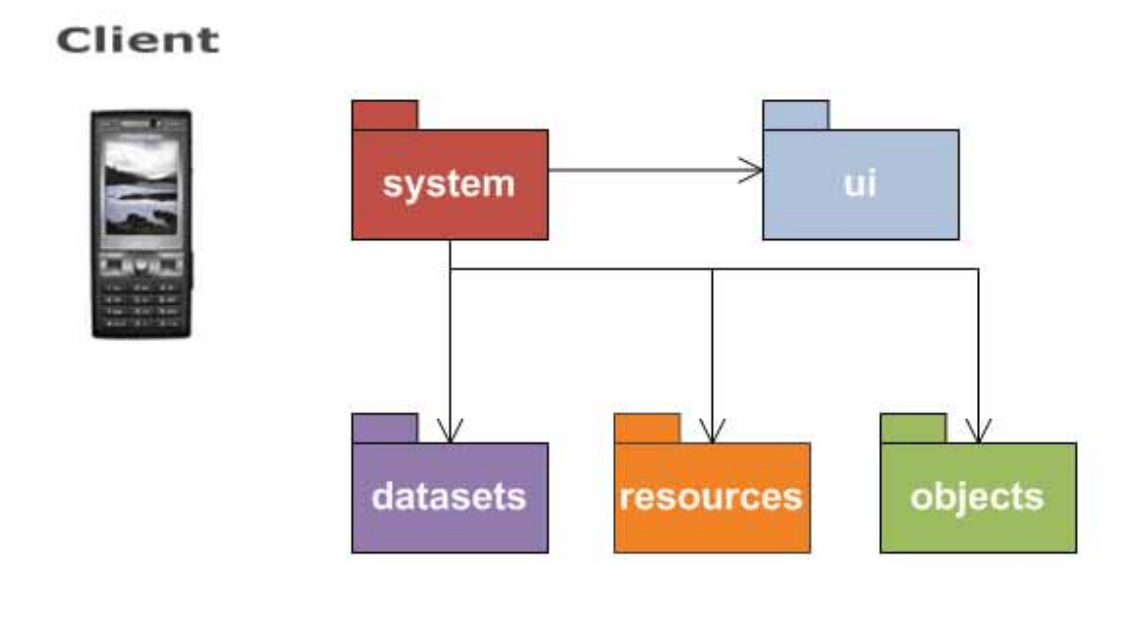


Figure 6.7: Client Packaging

The following pages contain class diagrams for each package. Appropriate attributes and methods have been provided, however suitable assessor and mutator methods have been omitted to improve readability. Set and Get methods will be implemented.

6.5.1 System

The system package contains the most widely used classes within the system. The relationship and details of each class within this package can be seen in Figure 6.8.

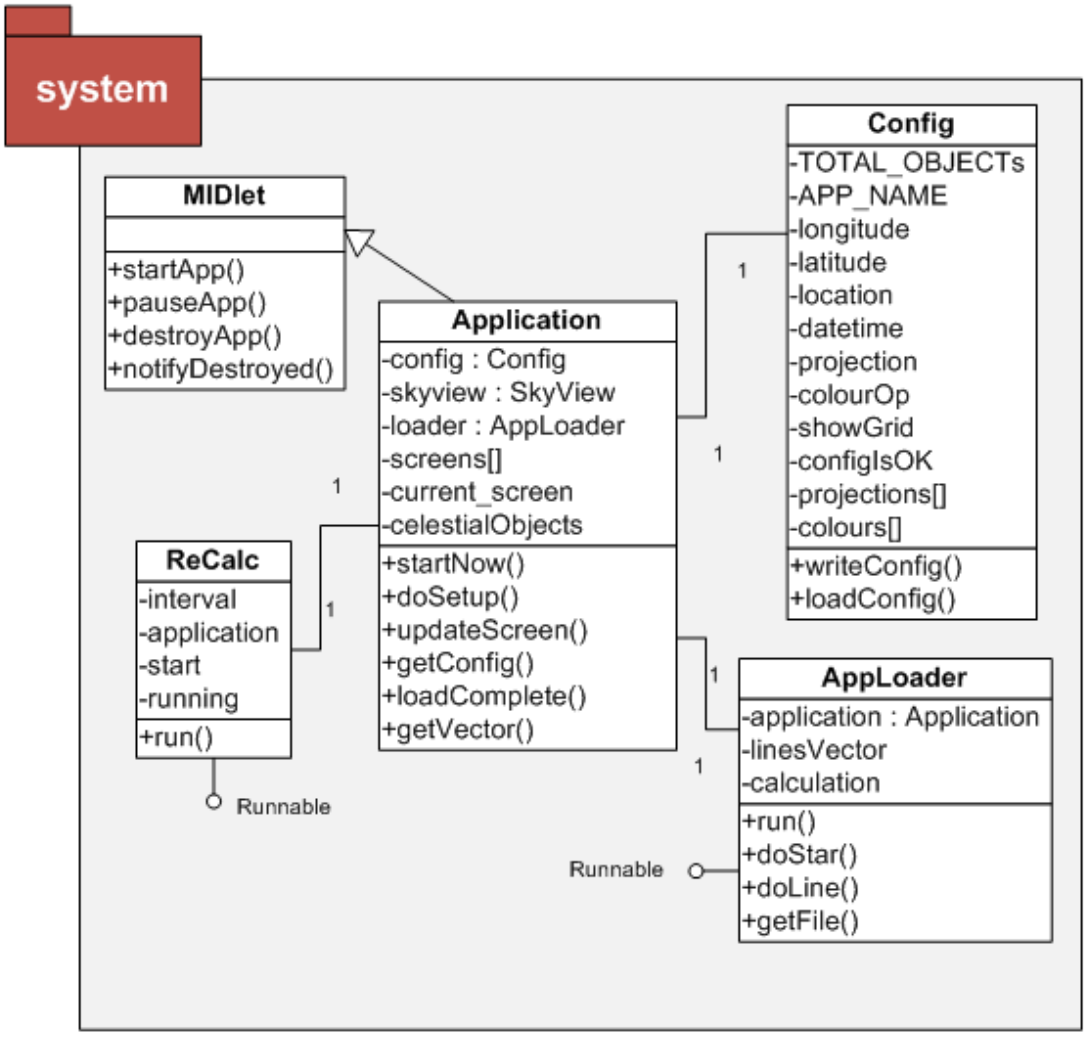


Figure 6.8: System Package

Application

This is the starting point of the system and extends `javax.microedition.midlet` to enable the running of the program. A reference to all other classes within this package will need to be recorded to manage the flow of control throughout the system. Application will control access to the system configuration so that key user attributes can be collected by any other referenced class. The main responsibility will be to initialize the loading of all the required data. Once successfully loaded within the system, the control will be passed to the SkyView.

All celestial objects used in the system will be referenced from the Application class. The objects will be stored in a Vector as it allows for an arbitrary length collection of data, allowing stars,

planets, the sun and moon and any other celestial object to be stored in one central location. Access to this key Vector is obtained via the **getVector()** method.

AppLoader

The AppLoader will be responsible for loading system data and communicating the progress with the loading interface. This class must contain references to the data files and must update the celestial object Vector stored within Application.

ReCalc

ReCalc will act as a simple timer Thread, having waited for the correct amount of time it will prompt for the SkyView to be reloaded with all CelestialObject positions recalculated.

Config

It will be the responsibility of this class to load and store user settings to non-volatile memory. Suitable public methods will need to be provided to allow direct access to key variables such as a users longitude and latitude coordinates at any time.

6.5.2 Objects

This package will be used to hold the different objects found within the system. The relationship and details of each class within this package can be seen in Figure 6.9.

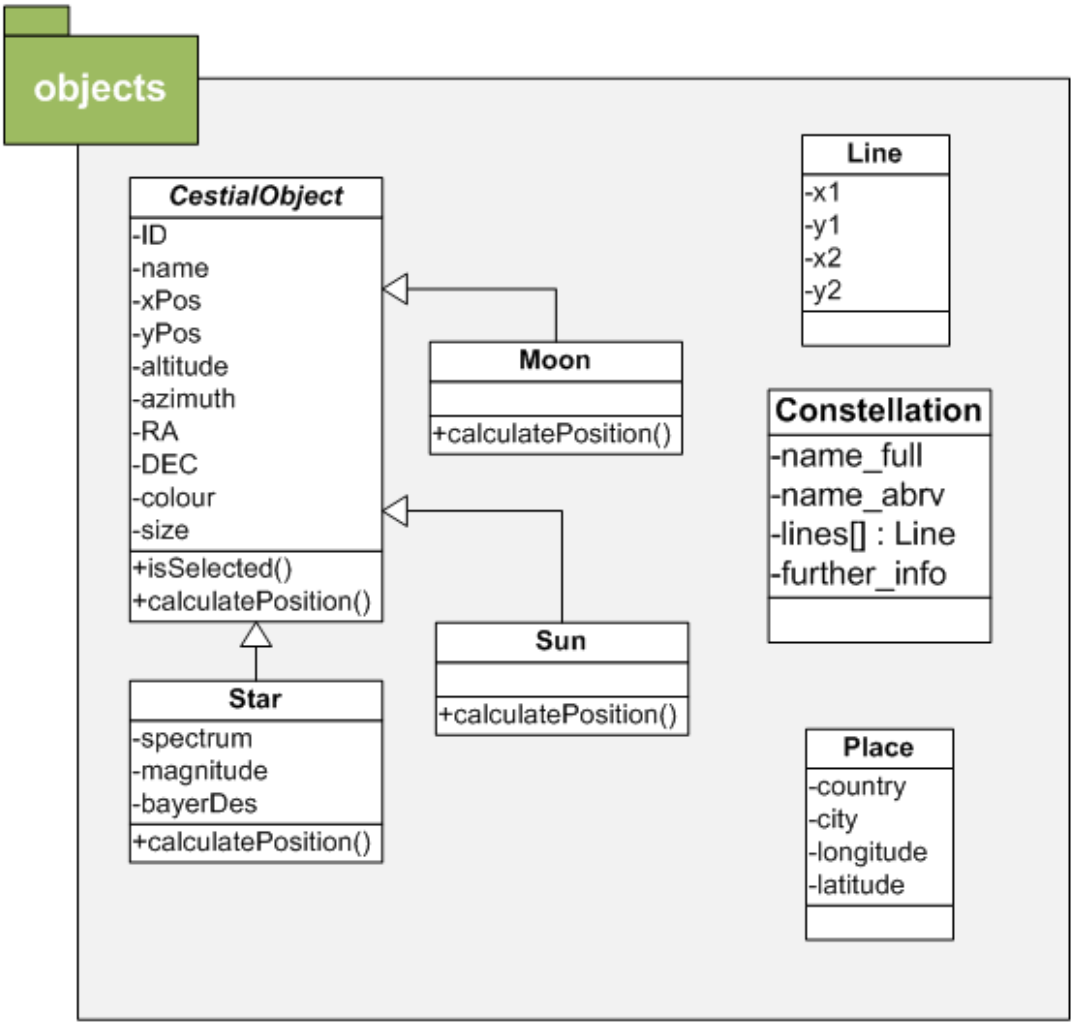


Figure 6.9: Objects Package

Celestial Object (Abstract)

This abstract class will provide the key attributes and methods of a Celestial Object. The name, right ascension, declination, colour and size will all be available here. The principle behind this design decision is that all objects will share these attributes and their associated methods. Objects extending this class will be required to implement a unique **calculatePosition()** method which will transform the objects right ascension and declination into horizontal coordinates; altitude and azimuth. These two calculated coordinates can then be used to plot an X,Y position on the mobile phone screen via a projection.

Grouping together the different objects in this manner will allow for iterating to take place neatly

and efficiently when converting objects between coordinate systems and when drawing to the display. Stars will be the main object of interest extending the CelestialObject class, however the Sun and Moon (as well as any other implemented object, such as satellites or comets) can also be represented in this manner; all that will be required is to specify the objects colour and size. Finally the **calculatePosition()** method will need to be implemented to generate an altitude and azimuth for the object when requested.

Place

Geographical locations will also be represented within the system as an object, containing the location name, longitude and latitude and any other associated information.

Line

objects.lines will be used to specify a specific line of a constellation figure. A Line consists of two tuples (X_1, Y_1) and (X_2, Y_2) which specify where a line should begin and end. To simplify this section of the design, Lines will be specified in the system by right ascension and declination points in line with the other celestial objects.

Constellation

Constellations consist of a name and description. Constellations will also contain a Vector of Line objects, which are used to represent the constellation figure lines.

6.5.3 User Interface

This package will contain all of the different screens that a user will see while using the system. The main key class will be SkyView, which is directly responsible for managing and generating a view of the night sky. The relationship and details of each class within this package can be seen in Figure 6.10.

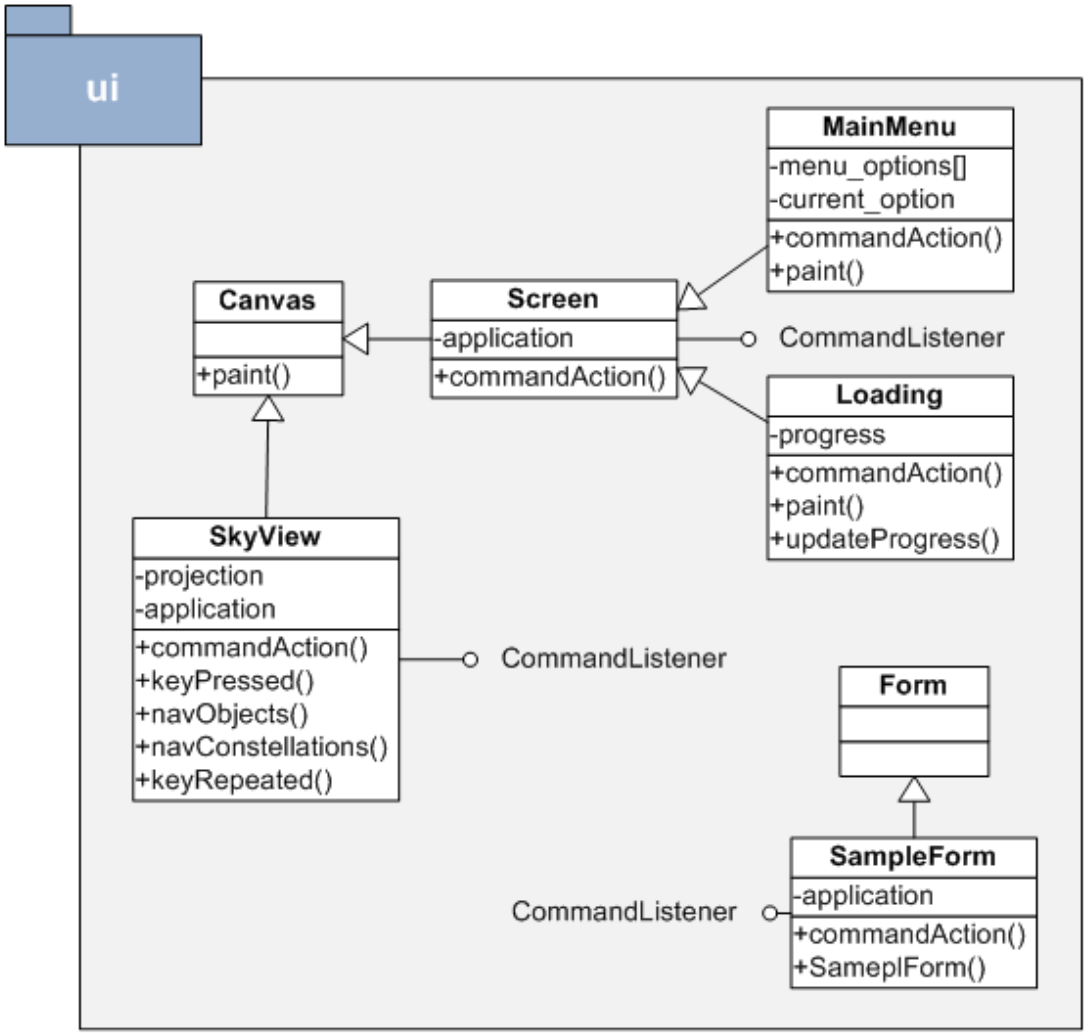


Figure 6.10: User Interface Package

SkyView

A rendered version of a users sky will be presented via SkyView. Suitable methods will be required to allow navigation and manipulation of the rendered sky triggered by key events on the mobile phone. Access to the main data structure holding the star data and constellation lines must be directly accessible for iterative plotting. As the SkyView extends `javax.microedition.lcdui.Canvas` it will be required to implement its own **Paint** method. This method must draw the rendered sky view according to the users current manipulation settings.

Screen

Screens can be characterized by having a specific graphical requirement, such as a menu interface or a loading bar.

The abstract screen object will allow for different types of screens within the system by extending the `javax.microedition.lcdui.Canvas` class. Each screen will implement its own paint method, allowing it to display required user interface elements. Screens will also be required to deal with user inputs through the *javax.microedition.lcdui.CommandListener* implementation. Several screens have been created to allow for menus and progress interfaces, including:

1. Main Menu
2. Loading Screen
3. Setup Menu
4. Updates Menu

Form

Any user input or text information screens will be represented by a specific class extending the **javax.microedition.lcdui.Form** class.

It became apparent during the creation of the prototype that any screen which displays only textual information or simple user inputs is more suited to a J2ME Form. These Forms are similar to screens however are much more easily implemented; options such as adding text or input fields are much more straight forward. For this reason a new abstract class will be created to represent forms and will be extended for specific user setting and information based screens.

6.5.4 Datasets

Datasets are included in the packaging to house any classes which directly contain a data source, seen in Figure 6.11.

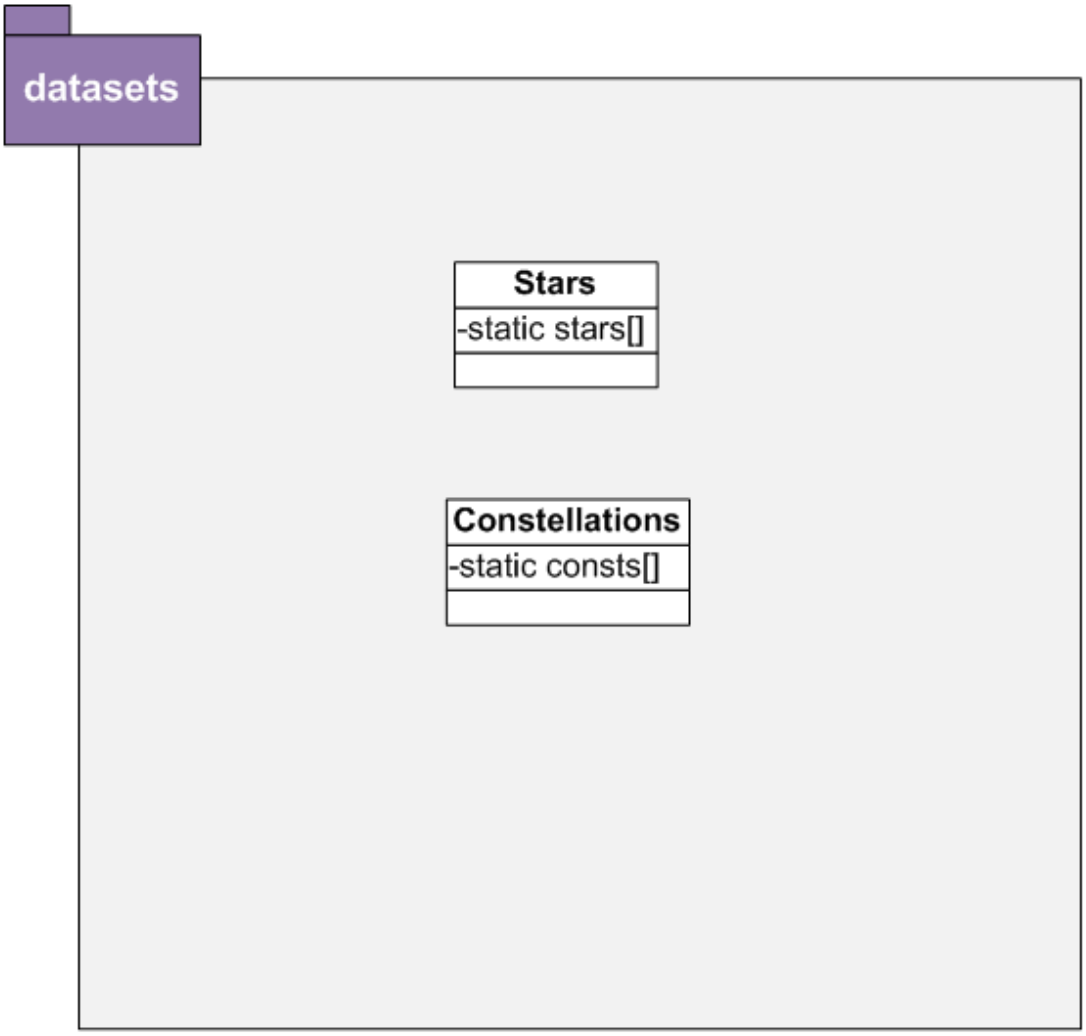


Figure 6.11: Datasets Package

The Constellations class will hold information regarding the 88 official constellations. Due to speed issues establishing a new HTTP connection on a mobile device, it has been decided that all textual information, such as description will be stored with a constellation locally, rather than allowing remote data to be used.

Initially Stars was to include a static array of all of the stars to be used within the system. However, after careful consideration this class will be used to store only the brightest of stars. This is because these stars can be loaded without having to read any data files, providing instant interaction potential for users.

6.5.5 Resources

The resources package, seen in Figure 6.12, contains different classes that provide important functionality for the system.

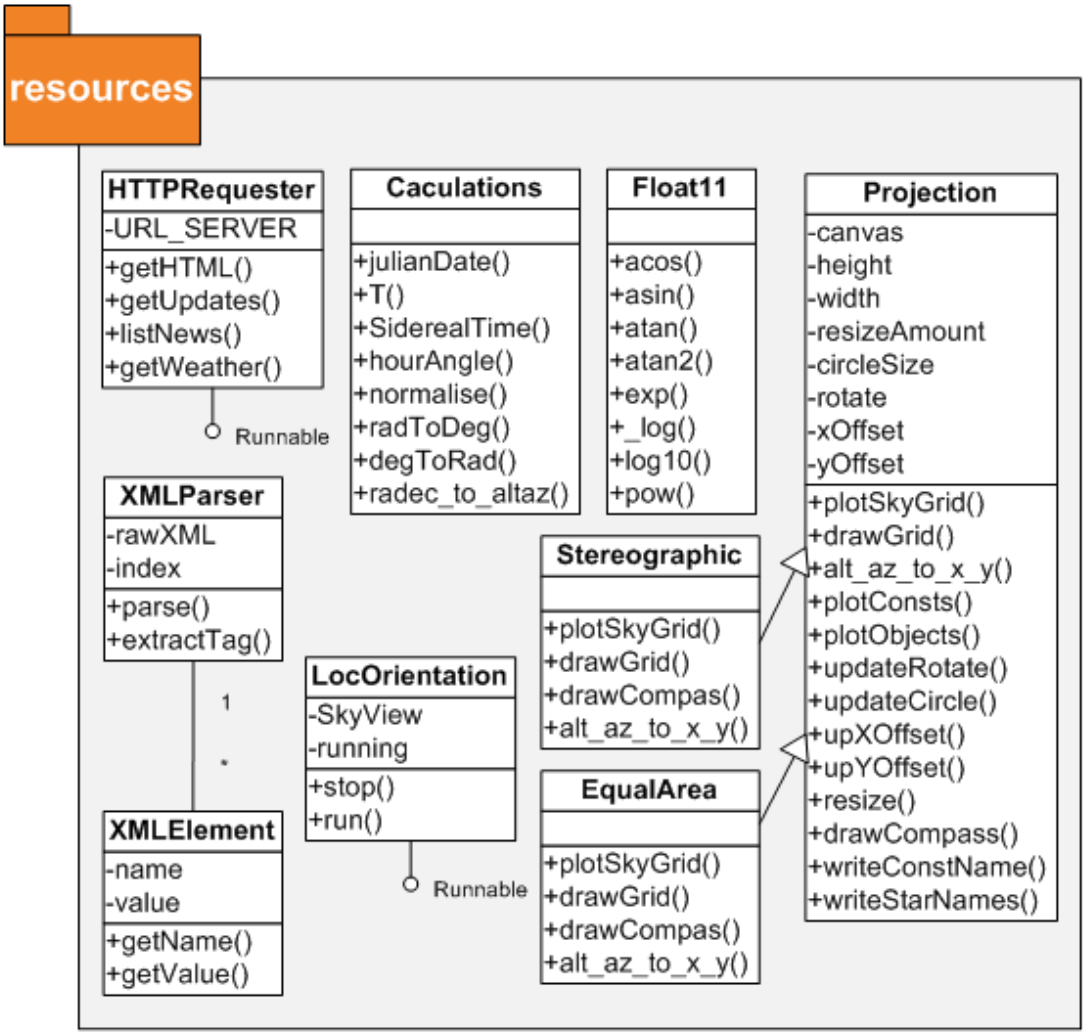


Figure 6.12: Resources Package

Projection (Abstract)

The abstract Projection class will be responsible for plotting objects and constellations onto the screen. Manipulation of objects will be performed via methods which adjust the X,Y coordinates of points. Classes extending this class, like EqualArea, will need to define the **alt-az-to-x-y()** method which will be used as a unique transformation from altitude and azimuth position to X,Y coordinates, which the main inherited Projection methods will make heavy use of.

Float11

The system relies heavily on trigonometry functions (which are not included in J2ME as standard) to calculate the position of celestial objects. This class, by Nikolay Klimchuk [15], causes no noticeable delay, even when used on thousands of unique calculations. Nikolay Klimchuk requests that any application created using his float class is provided to him free of charge. If the finished application uses this class then a complete working version of the application must be offered to him.

Caclulations

This class contains all of the necessary astronomical algorithms provided by Meeus [19]. A Calculation object is created and used to calculate the position of all of the systems CelestialObjects. If further astronomical functions are required, they will be added directly to this class. Initially astronomy calculation methods were included within every celestial object. This was a great waste of resources and created thousands of duplicate methods within the running application. This Calculations class was created to solve the issue, creating a single location for all astronomy based calculations.

HTTPRequester

The only class which will be used to communicate via HTTP will be the HTTPRequester class. The attempt to collect any remote data must be managed through this class via discrete methods. HTTPRequester will implement Runnable to allow all networking activities to operate within its own *java.lang.Thread* of execution.

XMLParser

A very simplistic XML (Extensible Markup Language) parser will be implemented in the application to parse information from the Server application. This will involve breaking down an XML document into individual elements.

XMLElement

In their most simplistic form, XML Elements consist of a name and a value. This information will be stored within its own class providing suitable accessor and mutator methods.

LocOrientation

This class will be run as a Thread and will deal with the *javax.microedition.location.Orientation* class. It will be owned by *ui.SkyView* and will adjust the orientation of the sky based on the current Azimuth of the device. It will be built such that devices which do not support the Location API will be unaware of its existence. Users fortunate enough to have the Location API will benefit from the automatic orientation of their rendered night sky.

6.6 User Interface Design

Following the first prototype and user testing the main user interface screens were created. Information and visual images for these interfaces can be seen later in this section.

6.6.1 SkyView

The SkyView screen will present a user with the complete view of the night sky (see Figure 6.13a). However, there is the potential for initial confusion as the vast number of celestial objects displayed may appear somewhat overwhelming. To avoid this when the skyview first appears, only the brightest objects will be displayed; the number of visible objects increase as a user zooms in.

Screen Layout

As a user zooms in on the the unique constellation figure lines will become more spaced-out and obvious (see Figure 6.13b). Users may rotate the display to align the rendered night sky with their view of the actual sky (see Figure 6.13c).

(a) Initial (b) Zoomed (c) Rotated

Figure 6.13: SkyView User Interface

6.6.2 Screens

Figure 6.14a provides an example of a user menu. Users will be able to navigate through a selection of available options and make a choice.

(a) Main Menu (b) User Input Form

Figure 6.14: User Screens/Forms

6.6.3 Forms

The inbuilt functions of a J2ME form allow for a screen to automatically scroll and to make use of inbuilt user input components. Forms will be used for collecting and adjusting all system user settings.

6.6.4 Controls

Following user testing and implementation of the first prototype some of the main controls for the system were decided. See Figure 6.15.

6.6.5 Loading

When loading data files or calculating the position of objects a loading screen will be presented to the user; showing a loading bar with a visual indication of the current progress.

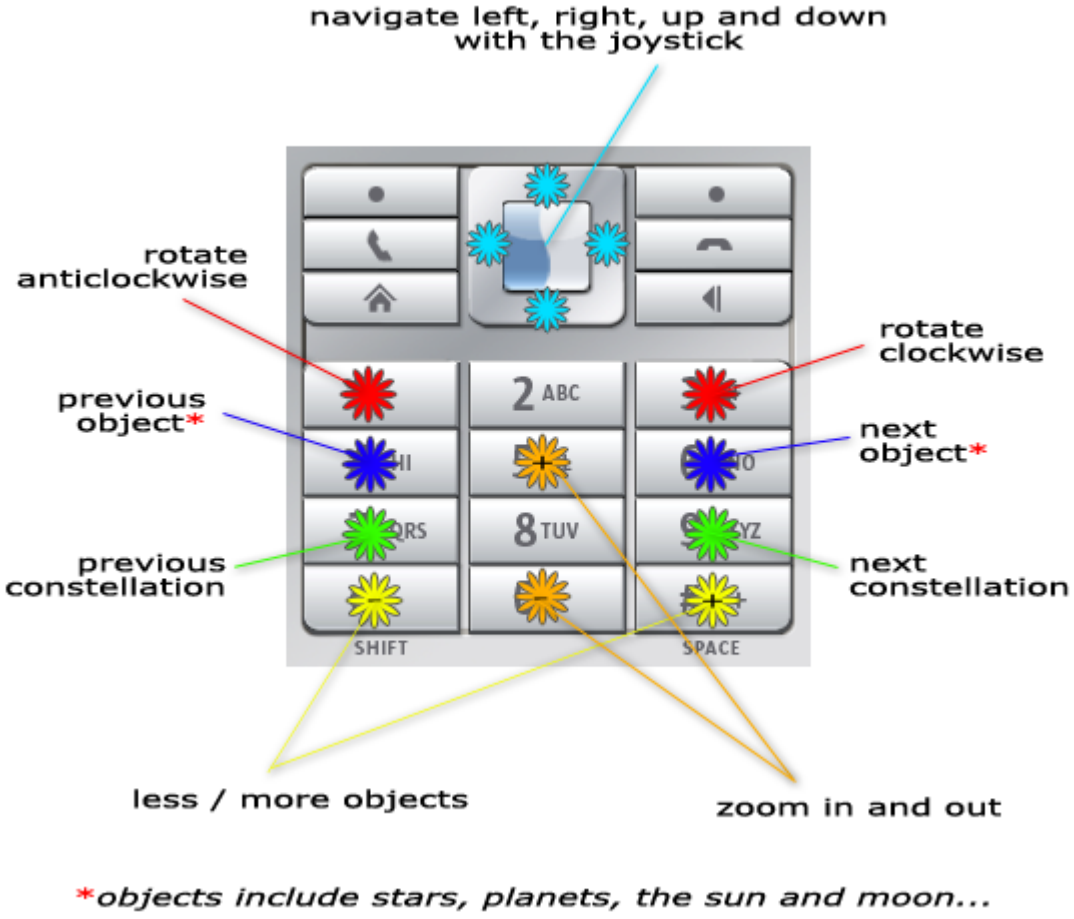


Figure 6.15: Application Controls

6.7 The Server Application

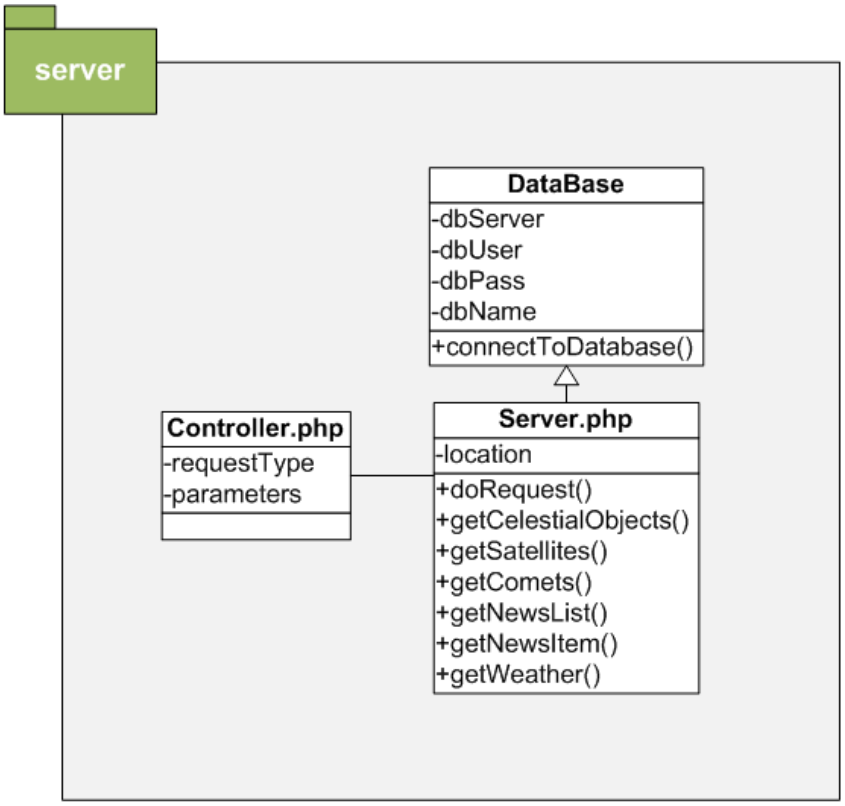


Figure 6.16: Server Application

The server application will be programmed in PHP and will utilise a mySQL database. A Controller file (Controller.php) will be used as a script to generate a new Server Object which houses a number of discrete operations. Access to the server operations will be made available via HTTP query string parameters. Two key variables will be used:

1. **request** - this will specify the type of server request in the form of an integer.
2. **parameters** - these will specify any details needed to perform the requested action.

The server will make use of a DataBase object for any interaction with the mySQL database. Requests available from the server will include:

1. **getCelestialObjects()**; returns visible CelestialObjects in an XML file format. These CestialObjects will then be generated on the client application and added to the local CelestialObjects vector.
2. **getNewsList()**; returns an XML document with the current news article available to read.
3. **getNewsItem()**; returns an XML document with a specific news article.
4. **getWeather()**; returns information regarding the users local weather conditions.

This design will easily allow for new functions to be added to the server as and when required, achieved by adding the appropriate methods and including a new unique request reference integer.

The server will make use of a mySQL database which holds information about the various different locations available. This should translate directly into a Yahoo Location ID for retrieving local weather information. The server will make use of external data sources to present a user with requested results. If necessary these data sources may need to be cached in order to present results in an acceptable amount of time.

6.7.1 Database Design

The Server database will include an exact copy of the geographical locations stored on the client phone along with additional information, such as a Yahoo location ID. The client will be able to send an integer reference for their selected location which will translate directly into a row in the locations table at the server. A control database will be used to store the locations of various remote data sources used in the application. This will allow for the system to change resource locations as and when required transparently. The database design in UML can be seen in Figure 6.17.

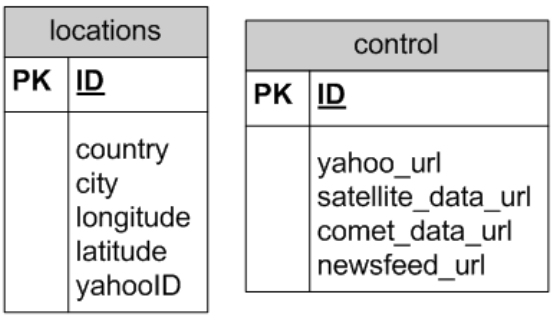


Figure 6.17: UML Database Design

6.8 Overall Client Application Structure

A complete class Diagram of the client system can be seen in Figure 6.18. The system packages have been colour coded in accordance with Figure 6.7 in the packages section.

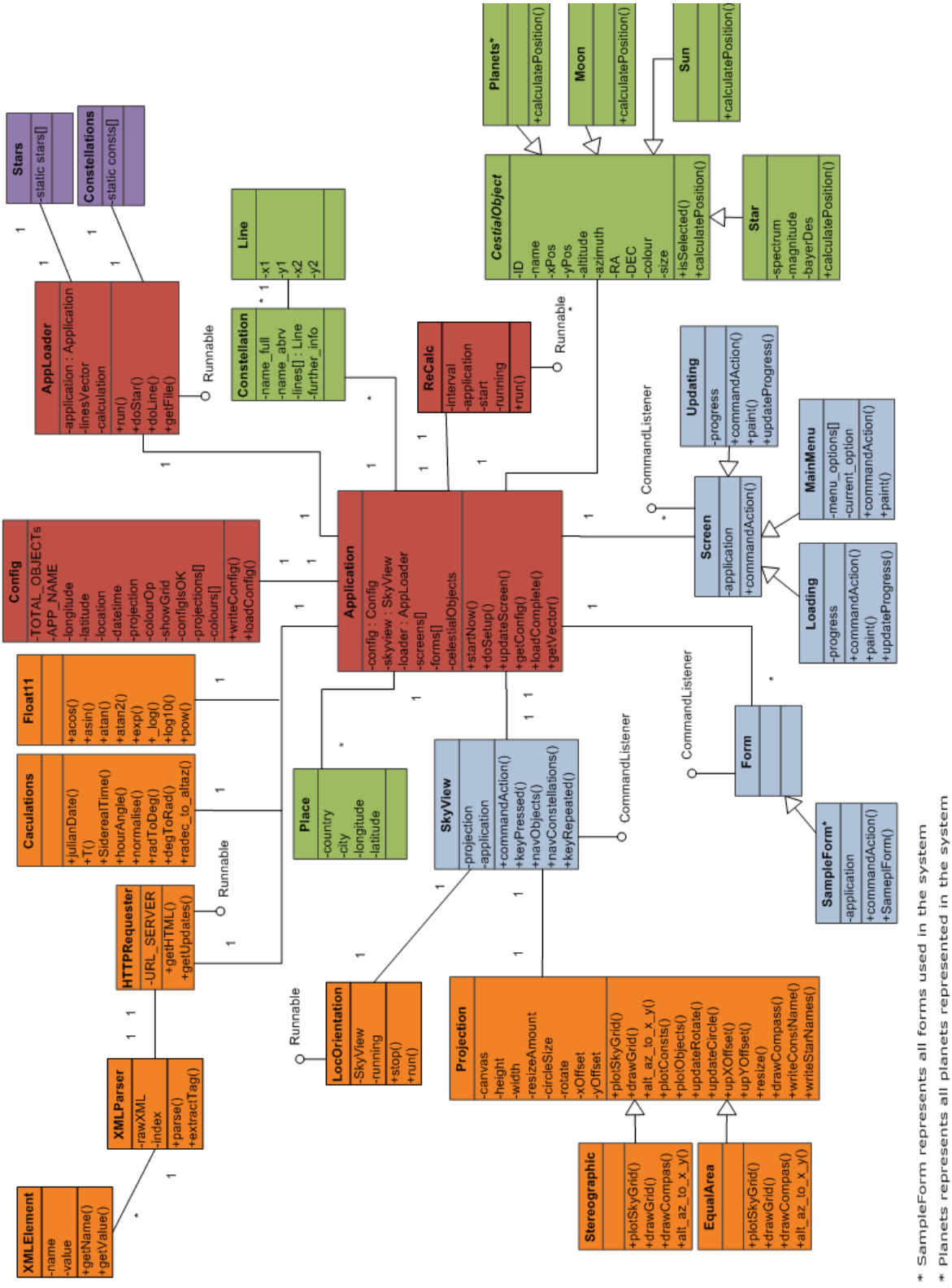


Figure 6.18: complete class Diagram of the client system

6.9 Client Sequence Diagram

A sequence diagram of the system can be seen in Figure 6.19 and a description can be seen below it.

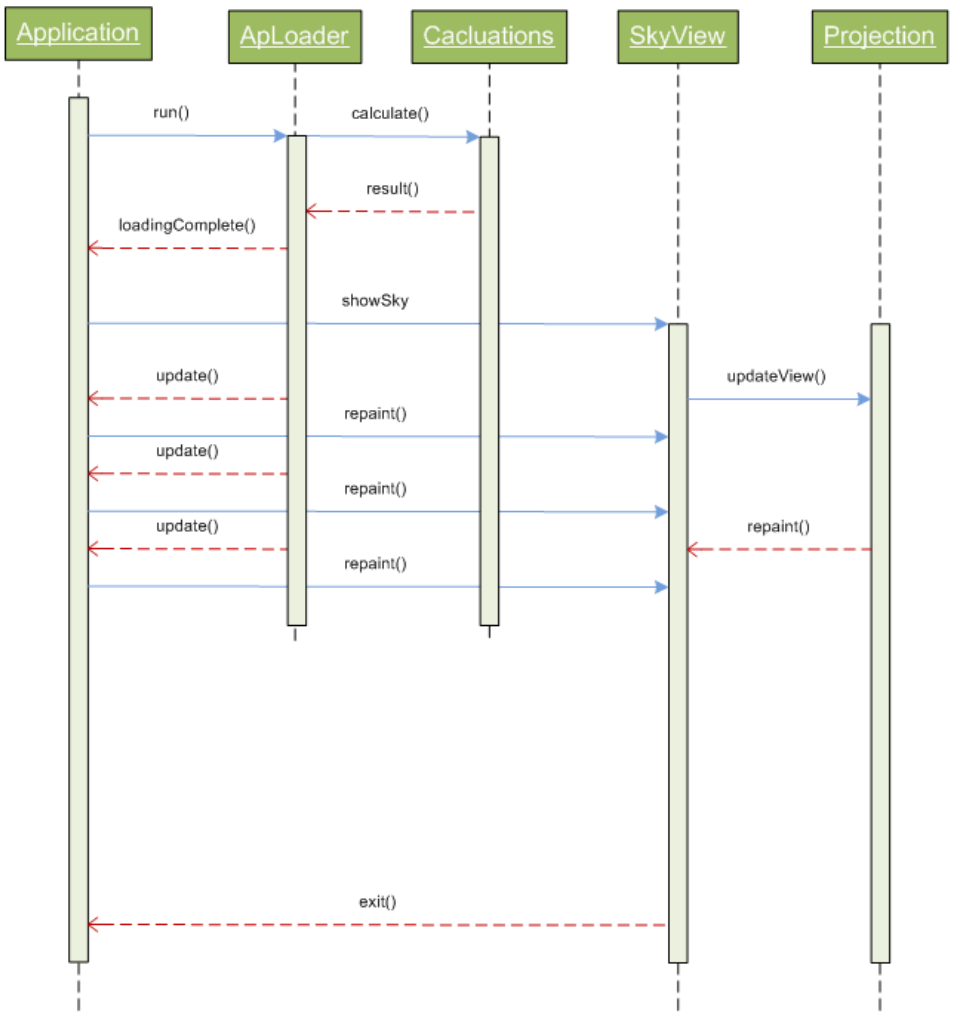


Figure 6.19: Overall structure sequence diagram

- **Application** will generate its own *ApLoader* (responsible for loading all data) and its own *Calculations* object through a *Config* object.
- **ApLoader** will operate in its own Thread once *Application* calls its *run()* method. This method will load all of the static data, including the brightest stars and planets, and also essential system data such as constellation lines. Having calculated the correct positions of these objects, *Application* will be called back. While this initial loading is taking place, a loading screen will be displayed to the user.
- **SkyView** is now created and displays the various objects that have already been loaded and are in main memory. A user may now interact with the sky model using rotate, zoom and navigation options.

- **Projection** is used by *SkyView* to convert to the correct x,y position for the currently selected projection.
- **ApLoader** will continue to work in the background loading all of the stars from the main data file.
- **ApLoader** is finally destroyed when all of the stars have been successfully loaded.

6.10 Client/Server Interaction Sequence Diagram

A representation of the client/server interactions can be seen in the sequence diagram in Figure 6.20 and a description can be seen below it.

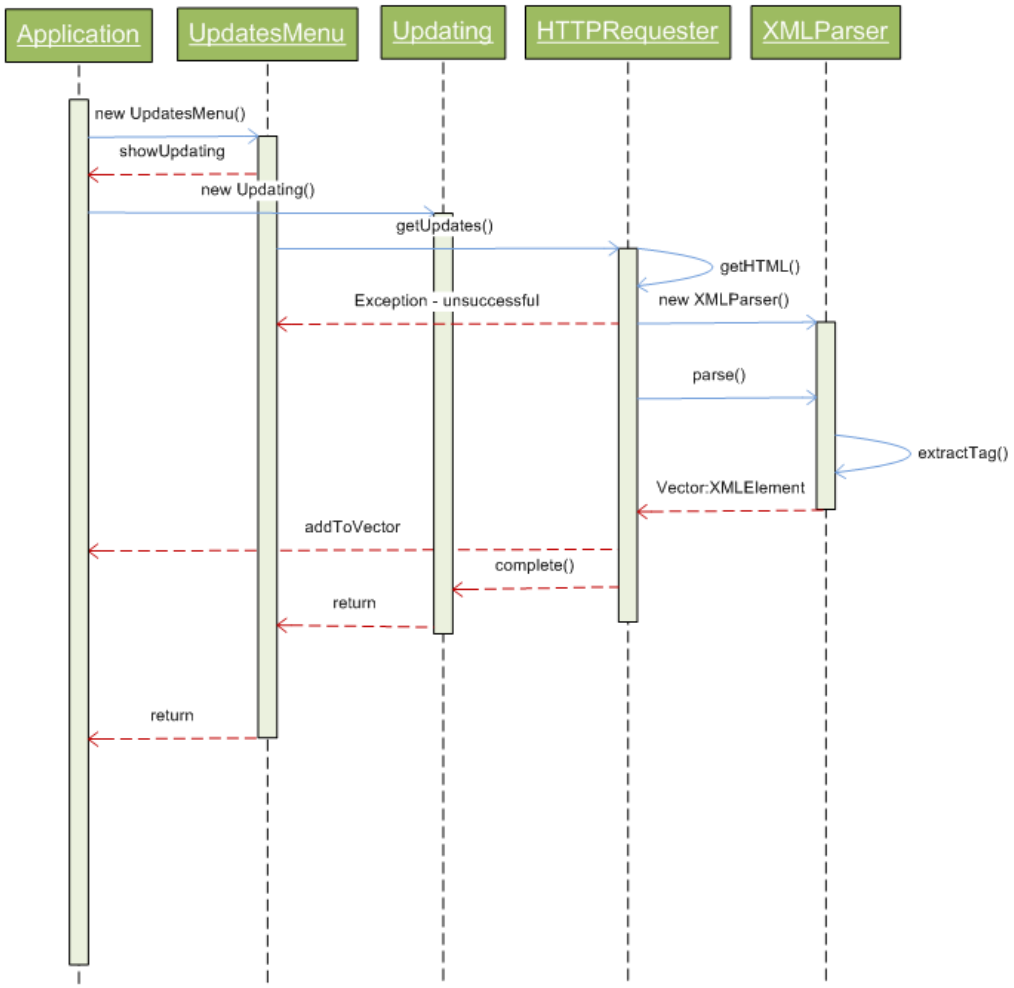


Figure 6.20: Client/Server sequence diagram

- **Application** sets the current display to the updates menu.
- **UpdatesMenu** receives a request from the user to retrieve an update from the server.

- **UpdatesMenu** calls back to the application to set the updating screen as current.
- **UpdatesMenu** creates a new *HTTPRequester* Thread, invoking the required request.
- **HTTPRequester** attempts to getHTML from a remote address, either an exception is generated or an *XMLParser* is generated with the remote data.
- **XMLParser** parses the remote data creating a Vector of XMLElements from the data source. This vector is passed back to *HTTPRequester* and in turn *Application*.
- **Updating** is alerted that the update is complete and the current display is return to *UpdatesMenu*.

6.11 Extensions

The following extensions from those identified during the requirements analysis phase in section 4.1.2 will be attempted and included in the application if at all possible :

1. **Location Environment Update**
   Remote weather information will be included within the application to provide a basic set of information regarding a users local weather conditions.
2. **Events in the Night Sky**
   By loading remote celestial objects and news feeds, the application will provide additional up-to-date information to roaming users.
3. **Planets, Moon and Sun Positions**
   The most logical and appropriate addition to the application is to include the sun, moon and planet positions in the application. This will require additional calculations from Meeus [19] and the correct implementation of the CelestialObject abstract class. This extension will increase the functionality of the application.
4. **Introduce GPS/Compass Mapping**
   Investigation into the possible use of the Java Location API for additional system functionality and usability - specifically regarding orientation will be included.

Chapter 7

Low-Level Design

7.1 Key System Activities

7.1.1 Loading

The application must load a large amount of data; it is vital that an efficient and effective loading policy is adopted. A number of different data sources are loaded and plotted within the system (please see Figure 7.1). The objects are added to a vector of *CelestialObjects* within System.Application; following this the horizon coordinates (altitude/azimuth) are calculated.

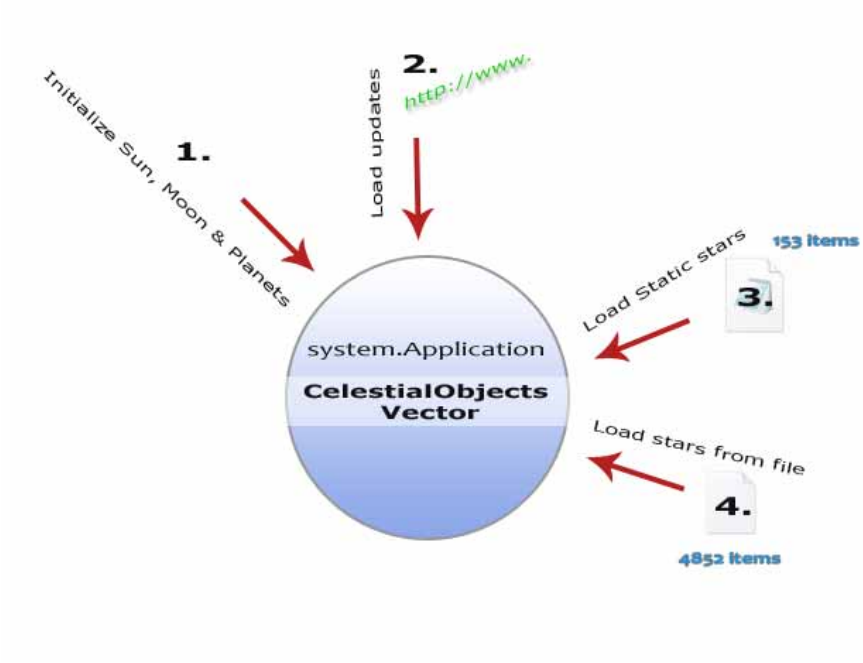


Figure 7.1: Loading to the CelestialObjects Vector (steps 1 to 4)

The ***system.AppLoader*** class will be responsible for all of the loading activities within the client application. For example, loading of updates takes place by iterating through the updates Vector and adding the elements directly into the CelestialObjects vector (see Figure 7.2).

```
// 2. load updates (if any)
Vector v = this.application.getUpdatesVector();
for(int i=0; i<v.size(); i++) {
    application.getVector().addElement(v.elementAt(i));
}
```

Figure 7.2: Loading of Updates

Once a stars position is calculated a check is made to see if it's visible from the current location. Only if a star is visible will it be added to the main CelestialObjects Vector (see Figure 7.3).

```
// create new star and assign colour/size
Star s1 = new Star(ID, info, RA, DEC, spectrum, magnitude, bayer);
starColour(s1);

// calculate star position (alt/azi)
s1.calculatePosition(application, calc);

// add star to vector if visible :)
if(s1.isVisible()) {
    application.getVector().addElement(s1);
}
```

Figure 7.3: Visible Stars Only

Additionally, the ApLoader will load all of the constellation lines from */resources/constlines.txt*. Lines are parsed and added, defining the total collection of lines which make up an entire constellation's figure. Lines are stored with their constellation name in the resource file, allowing for lines to be correctly assigned.

While the loading is taking place, the user is informed visually by a loading screen (see Figure 7.4).

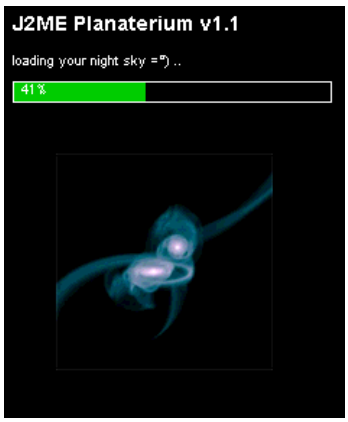


Figure 7.4: Loading Screen

The Loading screen (*ui.Loading*) has a method which allows for a local variable to be updated with the current loading progress. The *updateProgress()* method is called by *system.AppLoader* each time a significant proportion of the data has been successfully loaded (see Figure 7.5), adjusted

the progress bar accordingly.

```
/**
 * Update the progress percentage and repaint
 * @param progress
 */
public void updateProgress(int progress) {
    this.progress=progress;
    // repaint screen, progress has been updated.
    repaint();
}
```

Figure 7.5: The updateProgress() method

7.1.2 Displaying

The sky is represented as a circle within the application. This circle represents the Celestial Sphere, on which *objects.CelestialObjects* are plotted and displayed by means of a concrete Projection (for example *resources.EqualArea* or *resources.Sterographic*). Initially the Celestial Sphere circle is sized to the smaller of the screens height or width, allowing the complete sky to be visible (please see Figure 7.6). This circle is positioned in the centre of the screen and the circle size and position is recorded within *resources.Projection*. This initial size is also used as an absolute minimum; a user will not be able to zoom out any further.

Figure 7.6: complete sky

The night sky is updated when a user requests the view be adjusted by one of the input keys. For example pressing the *9* button will select and highlight a visible constellation. *ui.SkyView* implements *javax.microedition.lcdui.CommandListener*, this means that it can detect and react to user key presses.

View Manipulation

The circle size is adjusted when a user requests to zoom in or out; the *ui.SkyView* provides a case statement which filters commands and calls the appropriate method in the *resources.Projection*. This will result in the celestial sphere circle becoming larger (much larger than the display in most cases) or smaller (to the minimum size) please see Figure 7.7.

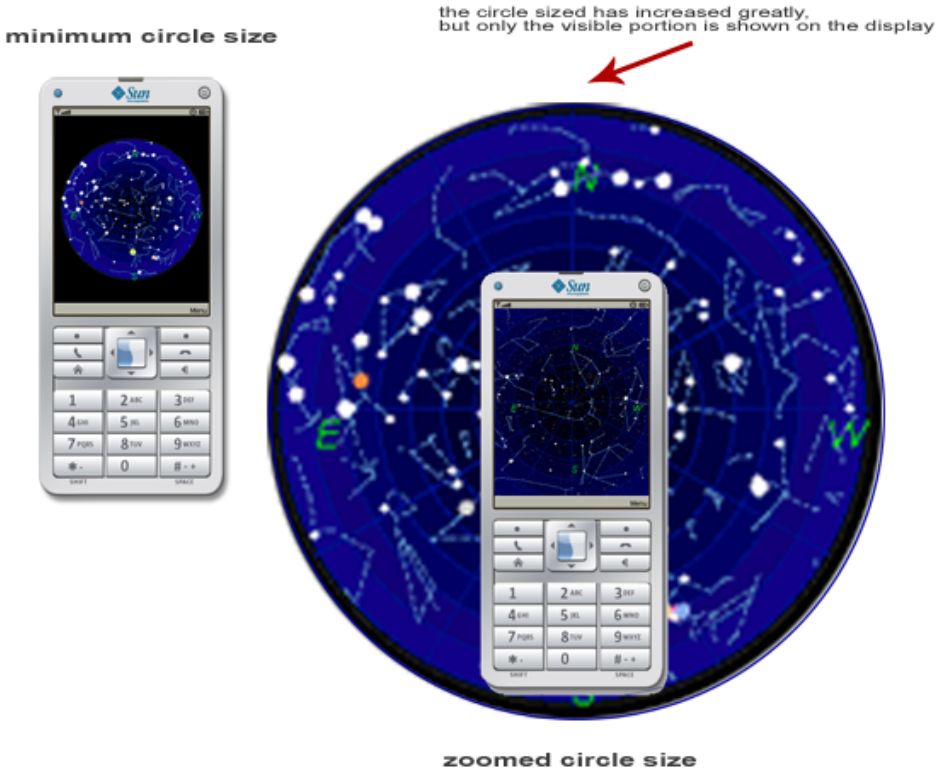


Figure 7.7: Celestial Sphere Circle Size

The circle size will be used to determine the number of visible stars displayed on the screen. The smaller the circle size, the more easily it appears overcrowded. So smaller circle sizes will only display the brightest stars. As a user zooms in a more detailed depiction will appear. Additionally a **X** and a **Y** offset variable are utilised to allow a user to move the circle left, right, up and down. This functionality works in exactly the same manner as the circle size but adjusts the position of the centre of the circle. Finally a variable **rotate** is used and managed by *resources.Projection* for allowing the circle to rotate by spinning all of the points by an additional degree.

SkyView display breakdown

The *ui.SkyView* has specific areas of the interface for displaying information, please see Figure 7.8 for more details.

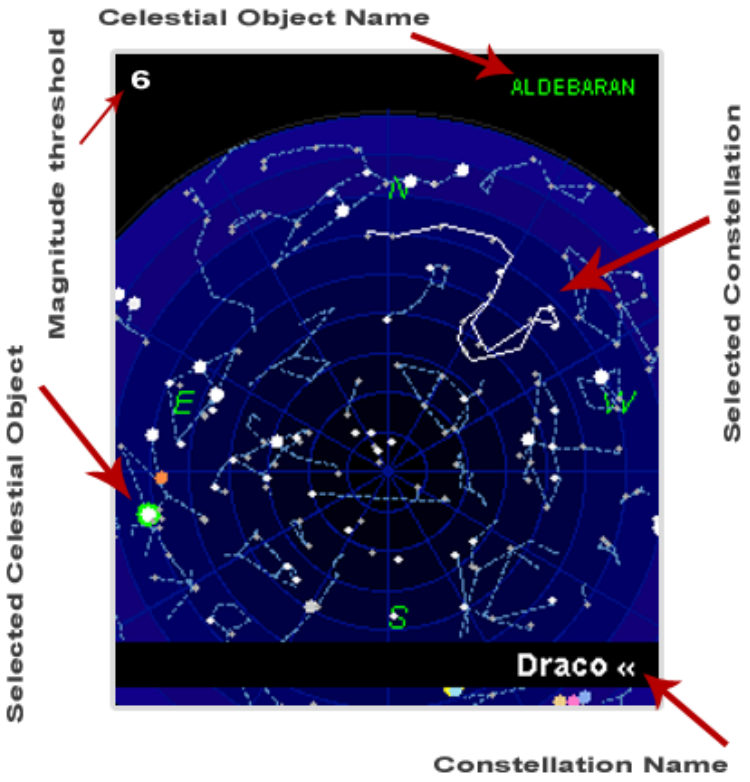


Figure 7.8: SkyView display breakdown

This consistent layout provides a user with an interface that will always remain the same no matter which objects / constellations are selected.

Drawing a Celestial Object

All *objects.CelestialObject* implement an abstract *draw* method, when the CelestialObjects Vector is being examined the draw method of each object will be called, passing with it the SyViews current *javax.microedition.lcdui.Graphics* object. Each *objects.CelestialObject* holds its own X and Y coordinates and as such can draw itself appropriately onto the display. A Star will represent itself as a circle, with its size depending on its magnitude and its colour depending on the current user colour setting. The colours of the planets does not change: these have been pre-assigned.

Drawing Constellation Figures

Constellation figures consist of a Vector of Line objects. Each line is drawn, and together they create the Constellation lines that make up a particular Figure. As constellations can be highlighted by a user, it was decided that constellation lines will be made a neutral colour initially, similar to

the background (please see Figure 7.9). When selected, solid bright lines will be used to alert a user of their selection.

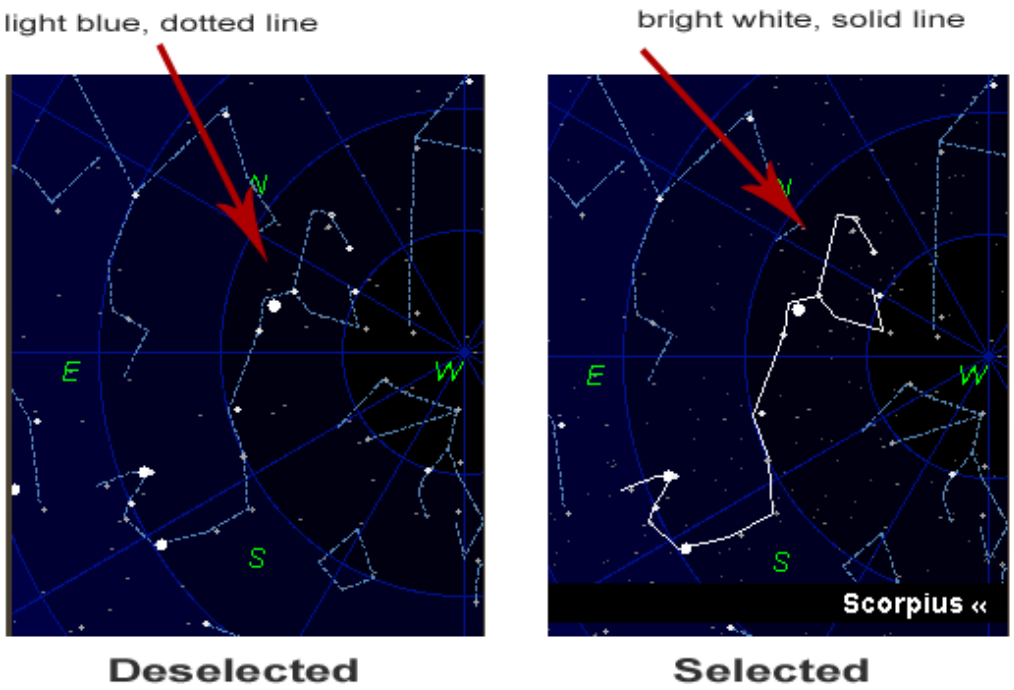


Figure 7.9: Drawing Constellations

Complex geometry could be used to make sure that lines do not appear outside of the main Celestial Sphere circle. It was decided that a much more simplistic approach would be adopted. As lines are represented as Right Ascension and Declination and are converted to Altitude and Azimuth any line (consisting of $(x_1, y_1) \rightarrow (x_2, y_2)$) with any X or Y variable that has an **altitude** $<$ **0** i.e. not above the horizon, will not be drawn. This saves on the additional calculations required to correctly crop the constellation lines.

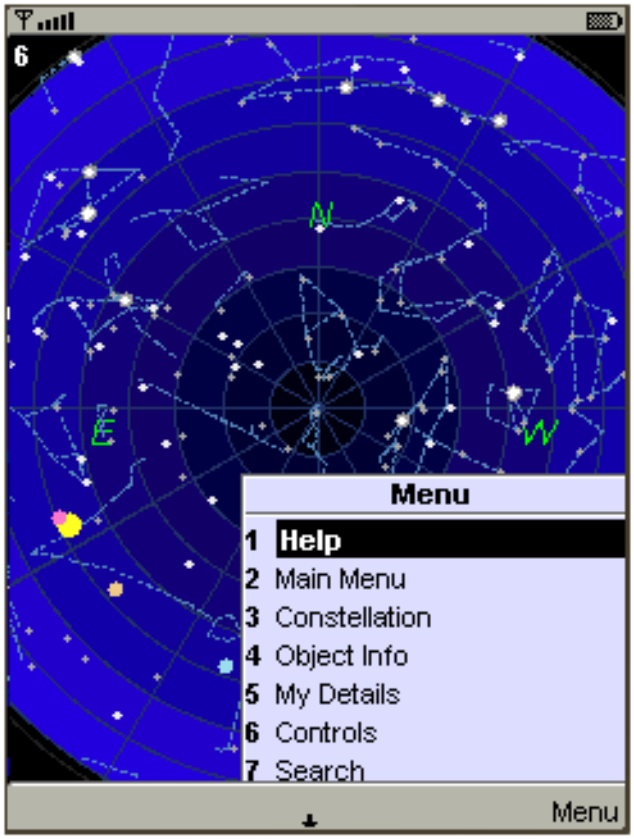


Figure 7.10: SkyView menu options

SkyView Menu Options

The *ui.SkyView* provides additional menu options, see Figure 7.10 for more details. These options allow a user to gain further information or return to the main menu at any time. Each menu option will allow the currently displayed screen or form to change or update. Additional menu items can be added very easily by additional case statements and command definitions.

Application Menus

Standardized menus are used throughout the system. The menus allow a user to make a choice between a list of possible options, (see Figure 7.11) clearly indicating the currently selected item, the list of possible options and additional information regarding recent actions.

Program Flows

The application switches between various Screens and Forms displaying some sort of interface to the user. Available screens and forms are defined within arrays by system.Application (see Figure 7.12). system.Application provides suitable methods for allowing the currently displayed user interface to be updated at any time (see Figure 7.13). The currently displayed user interface is assigned via the devices *Display*, which is attained simply via a call to *System.getDisplay()* and assigned by *Display.setCurrent()*.

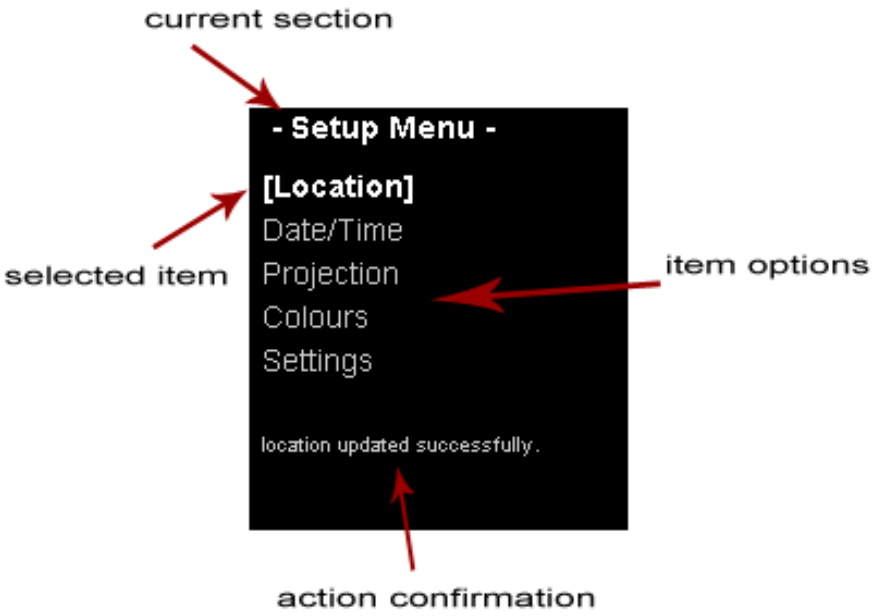


Figure 7.11: System Menus

```
// define screens
private Screen[] screens = {
    new MainMenu(this),
    new Loading(this),
    new SetupMenu(this),
    new UpdatesMenu(this),
    new Updating(this)
};

// define forms
private Form[] forms = {
    new LocationForm(this),
    new ProjectionForm(this),
    new ColourForm(this),
    new HelpForm(this),
    new SettingsForm(this),
    new SearchForm(this),
    new AboutForm(this)
};
```

Figure 7.12: System Screens and Forms

```
/**
 * Update the current screen
 * @param screenIndex
 */
public void updateScreen(int screenIndex) {
    System.out.println(screenIndex);
    current_screen = screenIndex;
    if(current_screen==-1) {
        // the sky view screen
        phoneDisplay.setCurrent(skyview);
    } else {
        // a normal screen
        phoneDisplay.setCurrent(screens[current_screen]);
    }
}

/**
 * Update current form to display
 * @param formIndex
 */
public void updateForm(int formIndex) {
    phoneDisplay.setCurrent(forms[formIndex]);
}
```

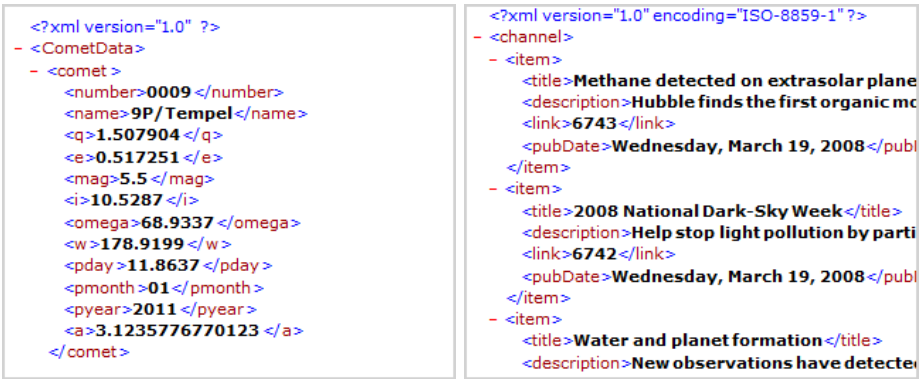
Figure 7.13: Updating currently UI

7.1.3 Updating

The Server will generate *XML documents* using the Document Object Model (DOM) in *PHP* [25]. *XML documents* will feed back to the roaming clients for parsing. The parsed document elements will then be added to local Vectors and used appropriately.

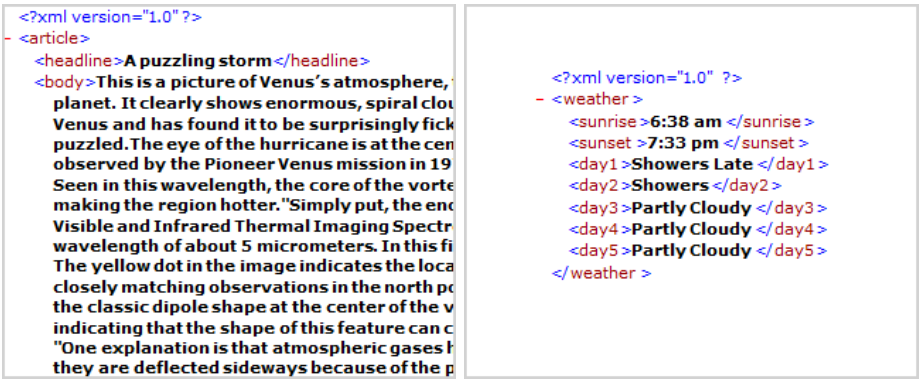
Four types of XML documents are available from the server:

- **Comets** - consisting of zero or more comet elements, each of which contain the key orbital elements required for calculation of a comets position. Comets included in this file must be visible by the naked eye and hence are required to be loaded directly from the server (see Figure 7.14a).
- **News Listing** - An up-to-date listing of available news articles, including a title, date and news identification number (see Figure 7.14b).
- **News Article** - An article title and body are available from a given news identification number (see Figure 7.15a).
- **Weather** - A single forecast for the next 5 days for a given location, contained within a Weather element (see Figure 7.15b).



(a) Comets XML (b) News Listing XML

Figure 7.14: Sample XML Documents



(a) News Article XML (b) Weather XML

Figure 7.15: Sample XML Documents

7.2 Parsing Data Files

A number of resource files must be successfully parsed to extract system data, this includes:

- **stardata.txt** All naked eye visible stars which are greater than first order magnitude. Each star contains a possible name, right ascension, declination, spectrum and magnitude. This file contains **4852** stars.
- **constlines.txt** All of the individual lines which make up the complete figure must be loaded and assigned to the correct constellation. Each line consists of a start and end point, recorded as right ascension and declination values. This file contains **854** lines.
- **places.txt** Possible geographical locations are stored within this file. A location consists of a Country, City, Longitude and Latitude value. This file contains **123** locations.

These files can be separated into two categories:

- **stardata.txt** and **constlines.txt** are both required to be present by the SkyView.
- **places.txt** is only required when a user wishes to select or update their current location.

System.ApLoader provides a method for collecting a resource file and adding each line to a Vector:

```
public Vector getFile(String fileName)
```

Once a given file has been successfully loaded into a Vector, this Vector can be enumerated and parsed to instantiate objects. A specific delimiter character is used in each data file to separate the variables in each line. The *java.lang.String substring* and *indexOf* are then used to systematically parse each variable on each line.

7.3 Threading Policies

7.3.1 Real Delay

Opening and loading a file into memory is the main cause of delay in the application, to resolve this issue, specific Thread policies are to be adopted.

7.3.2 Loading data

system.ApLoader will work as a Thread. The brightest objects will be loaded and then control will be passed directly to *ui.SkyView*, *system.ApLoader* will continue running in the background adding fainter stars to the Celestial Objects Vector. Each time 800 stars are successfully loaded a callback to *syatem.Application* and in turn the *ui.SkyView* will be made; this will update the display, incorporating the new objects. An interval of approximately 3-4 seconds is generated from refreshing the display every 800 stars, including a pause of 1 second following each update. This interval allows for the complete star dataset to be loaded within 25 seconds without using all of the devices processing power.

Due to the large amount of delay caused from loading the star data resource file into memory, its initial loading time is also included into the *system.ApLoader* workload. To enable the user to be informed of this progress, additional updates are included within the actual loading of the resource file into main memory before the *ui.SkyView* is displayed to the user and background loading begins. Without this additional progress update, the loading bar presented to the user would freeze at around 97% for a number of seconds, this resolved this issue. To improve this threading policy further, stars are ordered in ascending magnitude in the star data file, hiding the absence of the star data from the user.

7.3.3 Calculation updates

When *ui.SkyView* is displayed a background Thread will reload all of the celestial objects at a specific interval through *System.ReCalc*. This interval can be set via the settings menu and is specified in a period of minutes. The entire night sky is reloaded when this interval is reached, recalculating visible objects positions in the sky.

To reload the night sky, *system.reCalc* calls *system.Application* in order to save the current state of the interface and to reload it once the recalculation has taken place. The state includes the size of the celestial sphere circle, the x and y offsets, the rotate value and the currently selected constellation and celestial object.

7.3.4 Location API - Orientation

A Thread (*resources.LocOrientation*) utilizing the Location API's Orientation classes: (*javax.microedition.location.Orientation*) is included. This Thread systematically checks the devices Azimuth and updates the *ui.SkyView*'s rotation variable accordingly, locking out the users

ability to adjust the orientation of the screen manually. In order for this Thread to activate when the Location API is available, all Orientation calls were wrapped in this Stub class. The *ui.SkyView* creates an instance of this class upon loading, if any Exceptions are generated it is assumed that the device does not support the Location API and the application continues to function as normal. This addition to the application was interesting, however it was only tested using Emulation as no supporting devices were available.

When deploying the application on a device which does not support the Location API, a warning is generated:

```
TBCC Warning: prepare javax.microedition.location.Orientation: missing class
```

7.4 Server Requests

Clients will call the Server via a Universal Resource Locator (*URL*) including an additional request variable to specify the type of request being made.

For example, the following *URL* specifies to send a list of News items to the remote device:

```
http://www.-----.co.uk/finalyearproject/server/controller.php?request=2
```

Additional information can also be passed to the server via the *parmas* request variable. For example: obtaining a particular news article from the server will require a news identification number and weather information will require a reference to the users location.

The Client will implement a very simple XML parser which will read elements from a document. An XML Element class will be used to store the parsed elements and will be collected together in a Vector for easy access (see Figure 7.16). XML Parsers are available for J2ME [23], however, here a hand built parser will suffice and save on system resources.

7.4.1 Exceptions

The application will manage the result of a remote HTTP request via use of Exceptions. The *resources.HTTPRequester* class has a single method which collects remote data:
getHTML(String url, String params). This method has been declared as having the possibility of throwing an *Exception*, which indicates that an error has occurred while attempting to access a remote HTTP address. Either the String representation of the remote data is returned or an *Exception* is thrown.

As there are no guarantees that a mobile device will be able to connect to the Internet, this is the most simplistic and straightforward means of managing remote connections in the application. An additional timer Thread could be used to prevent unacceptable return times from collecting remote data, however it was decided that this overhead will not be managed in the application.

7.4.2 Caching

To improve performance, caching will ensure that fresh data is only collected once a day. When a request is received at the server a check is made for a local copy, if not present a fresh copy will be collected and saved. Subsequent requests will return the local version of the data.

cron, a time-based scheduling service for unix-like operating systems [6] is available on the server hosting platform. *cron* will be used to purge the server's local cache directory once a day.

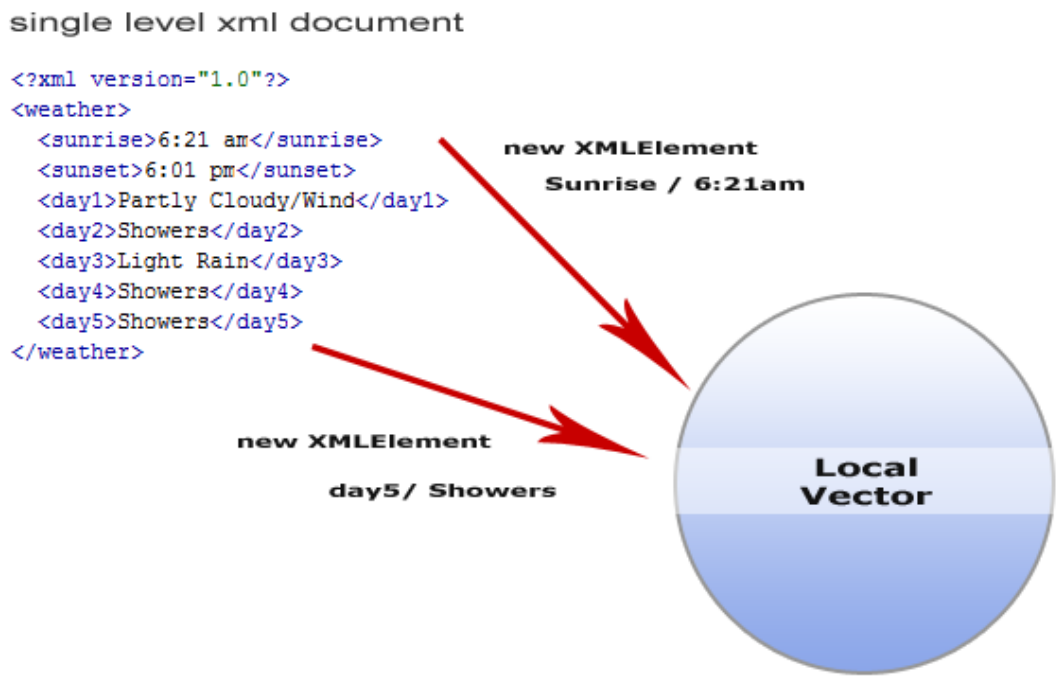


Figure 7.16: Single Level XML Parsing

7.5 Data Structures

7.5.1 Vectors

Vectors are used extensively throughout the application. They provide a dynamic size storage for any object types and provide simplistic iteration and management. All of the Celestial Objects used in the system are stored in a central Vector, providing a means for iterative calculation of positions and plotting, ensuring that every object is always examined. The use of a centralized vector also allows for *search* functionalities to be implemented. Each element of the Vector can be examined and checked for a matching search term and results can easily be pinpointed.

7.5.2 Static Arrays

Static arrays are used in the application for storing the official Constellations and the brightest stars. As they are static it reduces the runtime loading overhead as they already exist in memory as soon as the application is loaded. They are simple to evaluate as arrays have a fixed length and are easily referenced. Static data has been used sparingly to avoid excess file size as discussed in section 6.2.

Chapter 8

Implementation

8.1 Error Checking

To debug and evaluate methods a new boolean variable was introduced to each class called **verbose**. This variable if set to true, allow for various

```
System.out.println(useful debugging information here)
```

statements to be invoked. This proved to be an invaluable addition to the application.

8.2 Difficulties

A vast amount of the development time was spent implementing the various calculations and algorithms presented by Meeus [19]. Carefully entering constants in vast quantities and the implementation of trigonometry based calculations was required to calculate the positional information required by the applications.

8.3 TimeZones and Daylight Savings

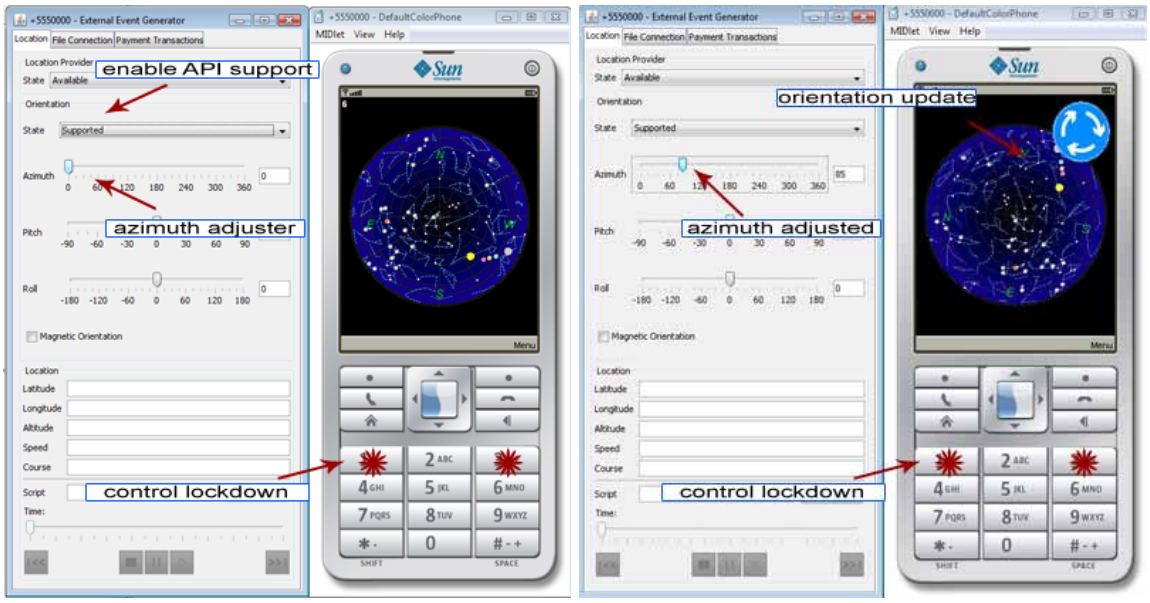
The correct time and GMT offset are required for positional calculations. It would appear that not only do Daylight Savings Times (DST) change [8] but also J2ME's TimeZones class, responsible for collecting GMT and DST offset values, have some known bugs [20] for specific time periods. This could cause problems in regions with updated DST time offsets; unfortunately there was no time available to investigate this issue further.

8.4 Emulation

The emulation software (part of the Sony Ericsson SDK) allowed for the J2ME application to be run through eclipse on a desktop computer environment. This was invaluable in the development of the system. Although slower and more awkward in its operation, the emulator provided a reliable medium for the developments. Ideas could be tried and tested in J2ME without deploying the application to an actual device.

8.4.1 External Events

To use the Location API's Orientation classes, the Emulator was used extensively due to the lack of a supporting physical device. The Emulator provides additional tools for causing *external events* such as adjusting the devices azimuth orientation (see Figure 8.1).



(a) (b)

Figure 8.1: Emulator External Events

8.5 Deployment

At various key points during the developments the application was deployed onto my mobile phone. Having identified the correct procedure for porting to the actual device at the requirements analysis phase this was straight forward.

To correctly deploy the application, a specific JAD file configuration needed to be assigned. The JAD file is a simple text file which accompanies the applications compiled JAR file (see Figure 8.2).

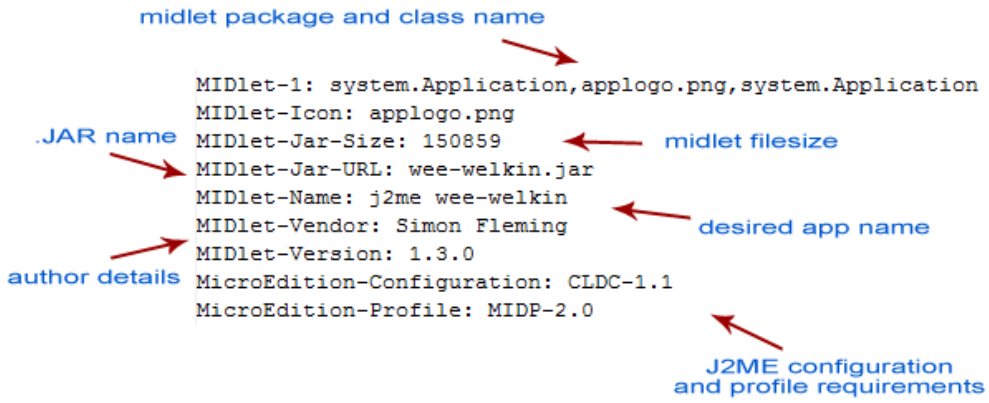


Figure 8.2: Application JAD file settings

8.5.1 Over the Air

Over the Air (OTA) distribution has been allowed by the creation of a dedicated website for the application:

`http://www.-----.co.uk/wee-welkin/`

This website allows for the application to be downloaded directly and includes links to the user manual and supported device listings.

8.6 User Manual

An in depth user manual was created to aid users understanding of the controls and functionalities of the system (see appendix **??**).

Chapter 9

System Testing

9.1 Requirements Specification

The application was checked against the requirements specification.

9.1.1 Core Features

- **Create a view of the night sky from a given position on earth on mobile phone screen.** This has been achieved via the SkyView and the astronomy calculations implemented. This was checked by means of a comparison test with existing desktop applications (see section 9.3). Unit testing (see section 9.2) was also used to ensure that acceptable celestial object positions are computed.
- **Draw constellation lines between stars.** The 88 official constellations are included in the application, and appropriate figure lines have been used.
- **Provide the ability to navigate this information by using zoom and rotation options.** The SkyView provides a user the ability to manipulate the view via zoom, rotate and navigation controls.
- **Introduce a server application that will provide the framework to feed data to roaming users.** The client/server interaction is made via the communication of data in the form of XML documents, via *resourecs.HTTPrequester*; *resources.XMLParser* and *resources.XMLElement*.
- **Ability for the date and time to be set.**
  This setting and associated mechanics have been implemented.

9.1.2 Non-Functional Requirements

- **The application must allow a new user to easily navigate around a current night sky view.** With the inclusion of a user manual and a help section in the application, users are able to successfully navigate the night sky.
- **The application must have reasonable loading times.** Appropriate Threading policies have been adopted to ensure the best possible loading times.
- **Aid perception by adding realism to the application, for example a horizon.** An atmosphere has been included in the application to aid perception for the users.

- **Storage and file size of the final application must not be excessive and should be in line with similar applications e.g. J2ME games.** Throughout the development of the application the file size has been monitored to prevent excessive storage requirements. The application restricts its use of static data and instead promote resource files to minimize storage requirements.
- **Experienced users must be able to use the application effectively.** Additional settings and a search feature have been implemented to allow advanced users to find what they are looking for for effectively.
- **Suitable colours, font sizes, layout, menus and program flows should be used.** Selected objects are clearly highlighted with appropriate clear and bold lines and fonts to present a user with a clear indication of their current selections and options.
- **Data transfer volume must be kept to a minimum.** HTTP requests are only made to collect remote data, these requests only add to the application and are not required for the application to function correctly.

9.2 Unit Testing

Testing of the specific methods for celestial object positions was conducted against the provided examples by Meeus [19]. These tests were made during developments to ensure that acceptable outputs are generated. Details of these tests can be seen in appendix 11.9.1.

9.3 Comparison Testing

The desktop application *Stellarium* [36] and `www.heavens-above.com` was used in order to compare the J2ME rendered night sky to a known correct solution. The details of this testing can be seen in Appendix 11.9.2.

9.4 Form Validation

During testing it became apparent that a number of user input forms could supply invalid data which would cause problems to the systems integrity. The date/time form was able to take invalid dates, which in turn broke the positional calculations. To resolve this issue, *javax.microedition.lcdui.Alert* was included on the form, when an invalid date is entered the application date/time is not updated and a simple alert style message prompt is presented to the user.

9.5 Performance

9.5.1 File Size

Variable name length

It states in the Sun Microsystems Tech Tips for *Optimizing J2ME Application Size* [22] that the length of the names for packages, variables, classes and methods all contribute to the total file size of the application. As the application contains meaningful names and identifiers, an *Obfuscation* process will be included in the final compilation using ProGuard [16]. This Obfuscation process will

shorten all names within the application from meaningful names such as *objects.CelestialObject* → A.B and *system.Application.getVector()* → *B.C.D()* , a significant reduction. The final application is *146* KB in file size, which is less than all of the Java games found on my mobile phone (see appendix 11.4).

9.6 Different Phones

The application was tested on the following mobile phones, by over the air deployment (see appendix 11.11).

1. Sony Ericsson K800i (main development platform)
2. Samsung G600
3. Sony Ericsson S500i

The main difference between the default platform and the Samsung 600 is that it has a significantly larger display and seems to render circles much more smoothly. On the downside, the Samsung is noticeably slower at loading resource files and also there is no joystick, only a four way directional button. The Sony Ericsson S500i's main difference was that this phone was much faster at loading and downloading as it is a more modern phone.

9.7 Field Testing

The finished application was tested by four users and they each completed a questionnaire designed to collect user feedback. Although conditions were bitterly cold and we did not have the best visibility this was an interesting and useful activity. Users were very interested by the application and were happy to spend time exploring and were getting to grips with the controls. It does appear that there is an unavoidable learning curve and finding the correct orientation took some time. Users were able to clearly identify some of the more recognizable constellations and bright objects fairly quickly and seemed pleased with their findings.

Generally, users consensus was that they would be far happier testing and exploring the application in the summer months. Please see Appendix 11.12 for the completed questionnaires.

The main point highlighted during the field tests was that visibility of the various stars is not perfect, as such the brightest stars were made larger within the final application to help aid perception.

Additionally it was decided that the application should not show the 3000 magnitude 5 stars. They increased the loading time and file size and they were too faint to see during the tests.

Chapter 10

Conclusion

I feel that the project has been a success. A working J2ME application has been successfully created that fulfills all the projects core requirements. This application can be used to guide a user through the visible constellations in their night sky, as well as other bright objects of interest. Descriptive information regarding the constellations is fully available, providing background knowledge of the 88 official constellations.

I have successfully implemented several of my proposed extensions and have had positive feedback from testers. What's more, I have learnt a great deal and throughout enjoyed the process involved in arriving at the end result.

The only downfall I perceive is concerning perception, although the application does present the night sky it is still a challenge to easily and quickly match what you are looking at in the sky with the view on the application. There are thousands of visible stars in the night sky, spotting specific objects of interest on such a small display is challenging at times. The application requires some time and patience to explore the sky effectively, find objects of interest and manipulate the display to focus and match perception correctly.

Personally I have been using the application on my phone in various forms for some months now and I find spotting the larger and brightest objects straight forward.

I have spent a great deal of time researching, designing and testing this application since the summer of 2007 and am proud of the array of new skills I have learnt and of the problems that I have overcome.

The Times newspaper released an image of the night sky (see appendix 11.25) towards the end of the project; encouragingly this image is very similar to the interactive night sky created in my application and I feel that it indicates the level of the success I have achieved.

10.1 Reflections

If I were to attempt this project again I would focus far more on the HCI issues of matching the perception of a user with the interface. This project involved a great deal of research and testing to ensure that the astronomy calculations were implemented correctly and efficiently and this turned out to be the biggest time sink in regards to the project. Now that I have a good overall understanding of the required calculations and the J2ME programming platform, this would be achievable.

10.2 Future Work

If I had another month to work on this application I would definitely focus my efforts in a horizonal projection (see appendix 11.26), as used by the Sky at Night Newsletter, to help aid perception. I feel that the design of the system would allow for a new projection such as this to be implemented fairly easily, following careful identification of the various formula associated with it. The addition of such a projection would allow for further realism to be implemented by means of trees and buildings which would seem appropriate.

Additional information could also be presented to the user by means of a bespoke news feed, instead of utilizing the information made available on a general astronomy feed a custom one, specifically for star gazing could be created with ease. Furthermore, new calculations could be implemented to allow for the identification of further celestial events such as meteor storms and satellite identification.

I would also look to undertake testing on a wider selection of mobile phones allowing for the identification of any outstanding issues and gaining further user feedback.

Bibliography

[1] Astronomy.com. Rss news feed. http://www.astronomy.com/asy/rss/?sid=26 (accessed on 09/11/2007).

[2] Prof John D. Barrow. *Cosmic Imagery key Images in the History of Science*. Bodley Head April 2008, 2008.

[3] Digital Chocolate. Fotoquest fishing. http://www.digitalchocolate.com/games/mobile/fotoquest-fishing.html (accessed on 18/04/2008).

[4] Wikipedia Community. Apparent magnitude. http://en.wikipedia.org/wiki/Apparent_magnitude (accessed on 14/11/2007).

[5] Wikipedia Community. Bayer designation. http://en.wikipedia.org/wiki/Bayer_designation (accessed on 12/11/2007).

[6] Wikipedia Community. cron. http://en.wikipedia.org/wiki/Crontab (accessed on 01/03/2008).

[7] Wikipedia Community. Stellar classification. http://en.wikipedia.org/wiki/Stellar_classification (accessed on 01/10/2007).

[8] CommunityDispatch.com. New rules for daylight savings time. http://communitydispatch.com/Announcements_9/New_Rules_for_Daylight_Savings_Time.shtml (accessed on 01/04/2008).

[9] Centre de Donnes astronomiques de Strasbourg. Vizier service. http://apm20.ast.cam.ac.uk/viz-bin/VizieR (accessed on 01/09/2007).

[10] Storm Dunlop. *Atlas of the Night Sky*. Harper CollinsPublishers Ltd, 2002.

[11] Sony Ericsson. Mobile developer support - sony ericsson developer world. http://developer.sonyericsson.com/ (accessed on 01/09/2007).

[12] C. Fabricius+. Dataset: Iv/25 – henry draper catalogue identifications for tycho-2 stars. http://archive.astro.umd.edu/archive/catalogs/4/4025/ (accessed on 15/11/2007).

[13] Simon Fleming. Mobile phone planaterium questionnaire. http://www.-----.co.uk/finalyearproject/questionnaire.php (accessed on 01/11/2007).

[14] James B. Kaller. *The Ever-Changing Sky : A Guide to the Celestial Sphere*. Cambridge University Press, 1996.

[15] Nikolay Klimchuk. J2me tools - float point calculations for cldc 1.1. http://henson.newmail.ru/j2me/Float11.htm (accessed on 05/12/2007).

[16] Eric Lafortune. Proguard : Java class file shrinker, optimizer, obfuscator, and preverifier. http://proguard.sourceforge.net/ (accessed on 10/12/2007).

[17] James Mainwaring. *An Introduction to The Study of Map Projection*. Cambridge University Press, 1996.

[18] Kevin McGrath. Bubba's bits (picosky). `http://www.bubbasbits.com/picoSky/` (accessed on 30/09/2007).

[19] Jean Meeus. *Astronomical Algorithms 2nd Ed.* Willmann-Bell Inc, 1998.

[20] Sun Microsystems. Class timezone. `http://java.sun.com/javame/reference/apis/jsr139/java/util/TimeZone.html` (accessed on 01/04/2008).

[21] Sun Microsystems. Jsr 75 - pda optional packages. `http://java.sun.com/javame/technology/msa/jsr75.jsp` (accessed on 03/12/2007).

[22] Sun Microsystems. Optimizing j2me application size. `http://java.sun.com/developer/J2METechTips/2002/tt0226.html` (accessed on 10/11/2007).

[23] Sun Microsystems. Parsing xml in j2me. `http://developers.sun.com/mobility/midp/articles/parsingxml/` (accessed on 05/03/2008).

[24] Patrick Moore. *Teach yourself Astronomy*. Hodder Headline PLC, 1995.

[25] PHP.net. Dom xml functions. `http://uk.php.net/manual/en/ref.domxml.php` (accessed on 06/03/2008).

[26] Jennifer Preece. *Interaction Design: beyond human-computer interation*. Jon Wiley and Sons Inc, 2002.

[27] Roger S. Pressman. *Software Engineering - A Practitioner's Approach*. MCGraw-Hill, 2005.

[28] SkyMap Pro. Constellation figures. `http://www.skymap.com/constellations.htm` (accessed on 15/11/2007).

[29] HighBeam Research. Latitude and longitude of world cities. `http://www.infoplease.com/ipa/A0001769.html` (accessed 20/12/2007).

[30] H. A. Rey. Constellation figures from: A field guide to the stars and planets by donald h. menzel collins 1966. `http://www.skymap.com/files/constellations/dhmenzv8.zip` (accessed on 15/11/2007).

[31] Steve McRobb Simon Bennett and Ray Farmer. *Object-Oriented System Analysis and Design 3rd Ed.* McGraw-Hill Education 2002, 1998.

[32] The British Computer Society. Code of conduct. `http://www.bcs.org/upload/pdf/conduct.pdf` (accessed on 15/10/2007).

[33] The British Computer Society. Code of good practice. `http://www.bcs.org/upload/pdf/cop.pdf` (accessed on 15/10/2007).

[34] SourceForce.net. Mobile planetarium - for java-enabled mobile phone. `http://mobilestarchart.sourceforge.net/` (accessed on 20/10/2007).

[35] SourceForce.net. Mobile planetarium - short manual. `http://mobilestarchart.sourceforge.net/manual.html` (accessed on 20/10/2007).

[36] SourceForce.net. Stellarium. `http://www.stellarium.org` (accessed on 21/10/2007).

[37] Sun Mobile Device Technology. A survey of j2me today. `http://developers.sun.com/mobility/getstart/articles/survey/` (accessed on 25/10/2007).

[38] Kim Topley. *J2ME in a nutshell*. O'reilly Media, 2002.

[39] `htt://cfa-www.harvard.edu`. Orbital elements: Comets in mct format. `http://cfa-www.harvard.edu/iau/Ephemerides/Comets/Soft00Cmt.txt` (accessed on 03/11/2007).

[40] Marco Voegeli. class yahoo weather v2. `http://www2.voegeli.li/no_cache/code-tutorials/php-scripts/class-weather-v2.html` (accessed on 07/11/2007).

[41] Yahoo! Yahoo! weather rss feed. `http://developer.yahoo.com/weather/` (accessed on 05/11/2007).

[42] Yahoo! Zonetag : Cell location api. `http://developer.yahoo.com/yrb/zonetag/locatecell.html` (accessed on 19/11/2007).

[43] Georg Zotti and Meister Eduard Groller. A sky dome visualisation for identification of astronomical orientations. `http://www.cg.tuwien.ac.at/research/publications/2005/Zotti-2005-vis/Zotti-2005-vis-.pdf` (accessed on 10/10/2007).

Chapter 11

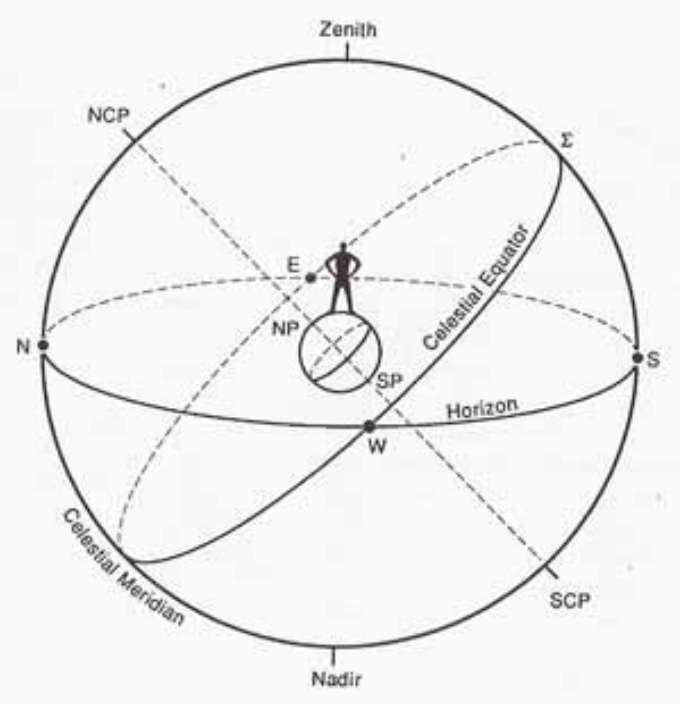
Appendix

11.1 Astronomical Research

11.1.1 Celestial Sphere

Figure 11.1 shows a representation of the Celestial Sphere, including the *Zenith* - the point directly above an observer, the *Nadir* - the point directly beneath an observer, the *Celestial Equator* which is an extension of Earths equator onto the Celestial Sphere, the *North and South Celestial Poles* - NCP / SCP and an observers horizon.

Figure 11.1: Illustration of the Celestial Sphere



11.1.2 Plotting the position of a star

From the stars apparent equatorial coordinates: Right Ascension (α) and Declination (δ).

We first calculate the ***mean* sidereal** time at Greenwich for a given date, by means of the following algorithm.

First we calculate the **mean sidereal** for the data at 0h universal time (UT):

The Julian Day (JD) with reference to a Year (Y), Month (M), Day(D) is obtained as follows - please note D is represented as a floating point representation of a day and time.

if M *ge* 2, leave Y and M unchanged if M =1 or 2, replace Y by Y-1, and M by M+12 A = INT(Y/100)

B = 2 - A + INT(A/4) (Gregorian Calendar) or B = 0 (Julian Calendar)

The Julian Day is then

JD = INT(365.25(Y+4716)) + INT (30.6001(M+1)) + D + B - 1524.5

next find T by

T = JD - 2451545.0 / 36525

The *mean* sidereal time at Greenwich at 0h UT is then given by the following expression which was adopted in 1982 by the International Astronomical Union: θ_0 = 100.46061837 + 36000.7700053608 T + 0.000387933 T^2 - T^3/38710000

Hour Angle is calculated by:

$H = \theta_0$ - φ - α

Finally, the calculation for the altitude and azimuth of the star is as follows

$$tan\ \ A = \frac{sin\ \ H}{cos\ \ H\ sin\ \ \varphi\ \ - tan\ \ \delta\ \ cos\ \ \varphi}$$

$$sin\ \ h = sin\ \ \varphi\ \ sin\ \ \delta\ \ +\ \ cos\ \ \varphi\ \ cos\ \ \delta\ \ cos\ \ H$$

altitude (A), azimuth (h), Hour Angle (H), Observer's latitude (φ)

Algorithm taken from a combination of descriptions from Astronomical Algorithms, Meeus [19].

11.1.3 Elliptic Motion

The following calculation is used to calculate the position of the planets and also comets [19] (p. 227-232).

The following orbital elements are supposed to be known:

a = semimajor axis, in AU
e = eccentricity
i = inclination
ω = argument of perihelion
Ω = longitude of ascending node
n = mean motion, in degrees/day

where i, ω and Ω are referred to a standard equinox.

Let ϵ be the obliquity of the ecliptic. For the standard equinox of 2000.0, use the value $\epsilon_2000 = 23^\circ\ 26'\ 21''.448$, from which:

$\sin \epsilon = 0.397\ 777\ 156 \quad \cos \epsilon = 0.917\ 482\ 062$

Then calculate:

F = $\cos \Omega$ G = $\sin \Omega \cos \epsilon$ H = $\sin \Omega \sin \epsilon$
P = - $\sin \Omega \cos i$
Q = $\cos \Omega \cos i \cos \epsilon$ - $\sin i \sin \epsilon$
R = $\cos \Omega \cos i \sin \epsilon + \sin i \cos \epsilon$

Then the quantities a, b, c, A, B, C are given by:

$\tan A = \frac{F}{P}$ $\tan B = \frac{G}{Q}$ $\tan C = \frac{H}{R}$
$a = \sqrt{F^2 + P^2}$ $b = \sqrt{G^2 + Q^2}$ $c = \sqrt{H^2 + R^2}$

Calculate the body's mean anomaly *M*. (omitted) Then the eccentric anomaly *E*. (omitted) The true anomaly *v*. (omitted) The radius vector *r*. (omitted)

Then the heliocentric rectangular equatorial coordinates of the body are given by:

$x = r\ a \sin (A + \Omega + v)$
$y = r\ b \sin (B + \Omega + v)$
$z = r\ c \sin (C + \Omega + v)$

For the same instant calculate the Sun's rectangular coordinates X, Y, Z. (omitted)

The geocentric right ascension α and declination δ of the planet or comet are then found from:

$\xi = X + w \quad \eta = Y + y \quad \zeta = Z + z$

$\mathbf{tan}\ \alpha = \eta\ /\ \xi$
$\mathbf{tan}\ \delta = \frac{\zeta}{\sqrt{\xi^2+\eta^2}}$

11.1.4 The Stars

Slowly over hundreds of years the stars will begin to move around as they travel further away from our solar system. Each star has its own magnitude (brightness), ranging from minus values (brightest) to positive integers (less bright). Today, magnitudes are calculated using electronic equipment and each successive magnitude value is considered to be half that of its predecessor. [14](section 4.9)

Without the use of special equipment only those stars with a magnitude less than 6 will be visible with the naked eye [4].

Stars also emit a colour ranging from deep red to orange. This colour is based on the surface temperature of the star. Magnitude is also colour sensitive with yellow light being the most sensitive to the human eye. A list of the star colour spectrum types can be seen in appendix 11.1.9.

Star positions are defined for a specific epoch: "A particular fixed instant used as a reference point on a time scale, such as B1950.0 or J2000.0" [19](p. 410). *Right ascension* is awkwardly measured in units of time (from 0 to 24 hours) from the vernal equinox, positive to the east, along the celestial equator. *Declination* is also measured in units of time from the equator, positive to the north, negative to the south, represented as degrees [19](p. 409).

11.1.5 Constellations

The brightest of the stars in a constellation are defined by a Bayer Designation [5] in which stars are ordered by brightness by the Greek alphabet, please see appendix 11.1.10.

11.1.6 The 88 Official Constellations

Andromeda	Circinus	Lacerta	Piscis Austrinus
Antlia	Columba	Leo	Puppis
Apus	Coma Berenices	Leo Minor	Pyxis
Aquarius	Corona Austrina	Lepus	Reticulum
Aquila	Corona Borealis	Libra	Sagitta
Ara	Corvus	Lupus	Sagittarius
Aries	Crater	Lynx	Scorpius
Auriga	Crux	Lyra	Sculptor
Botes	Cygnus	Mensa	Scutum
Caelum	Delphinus	Microscopium	Serpens
Camelopardalis	Dorado	Monoceros	Sextans
Cancer	Draco	Musca	Taurus
Canes Venatici	Equuleus	Norma	Telescopium
Canis Major	Eridanus	Octans	Triangulum
Canis Minor	Fornax	Ophiuchus	Triangulum Australe
Capricornus	Gemini	Orion	Tucana
Carina	Grus	Pavo	Ursa Major
Cassiopeia	Hercules	Pegasus	Ursa Minor
Centaurus	Horologium	Perseus	Vela
Cepheus	Hydra	Phoenix	Virgo
Cetus	Hydrus	Pictor	Volans
Chamaeleon	Indus	Pisces	Vulpecula

Figure 11.2: The 88 Official Constellations [14](p. 94)

11.1.7 Projections

"Map Projection is the representation, on a plane surface, of the meridians of longitude and parallels of latitude, reference to which defines any point on the earth's surface. Mathematically, the term 'projection', when restricted to the representation of points on a plane surface, means perspective projection; i.e the determining of the position on a plane of points in space as they would appear on the plane from a fixed view point. [17](p. 2)"

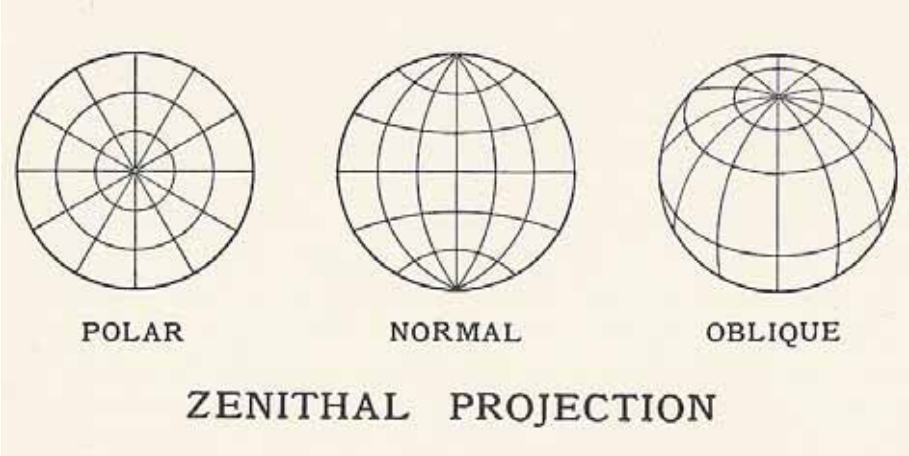
Conical or *Cylindrical* projections are not going to match any kind of perception of what is seen in the night sky as they are so distant in form.

Zenithal projections can be of a number of different forms that I have identified [17](p. 27-29):

1. *Polar*; in which the plane is tangent to one of the spheres poles.
2. *Normal*; in which the plane is tangent to any point at the equator of the sphere.
3. *Oblique*; in which the plane is tangent to any other point on the sphere.

Figure 11.3 [17](p. 28) provides visual representations of these types of Zenithal Projections.

Figure 11.3: Zenithal Projections



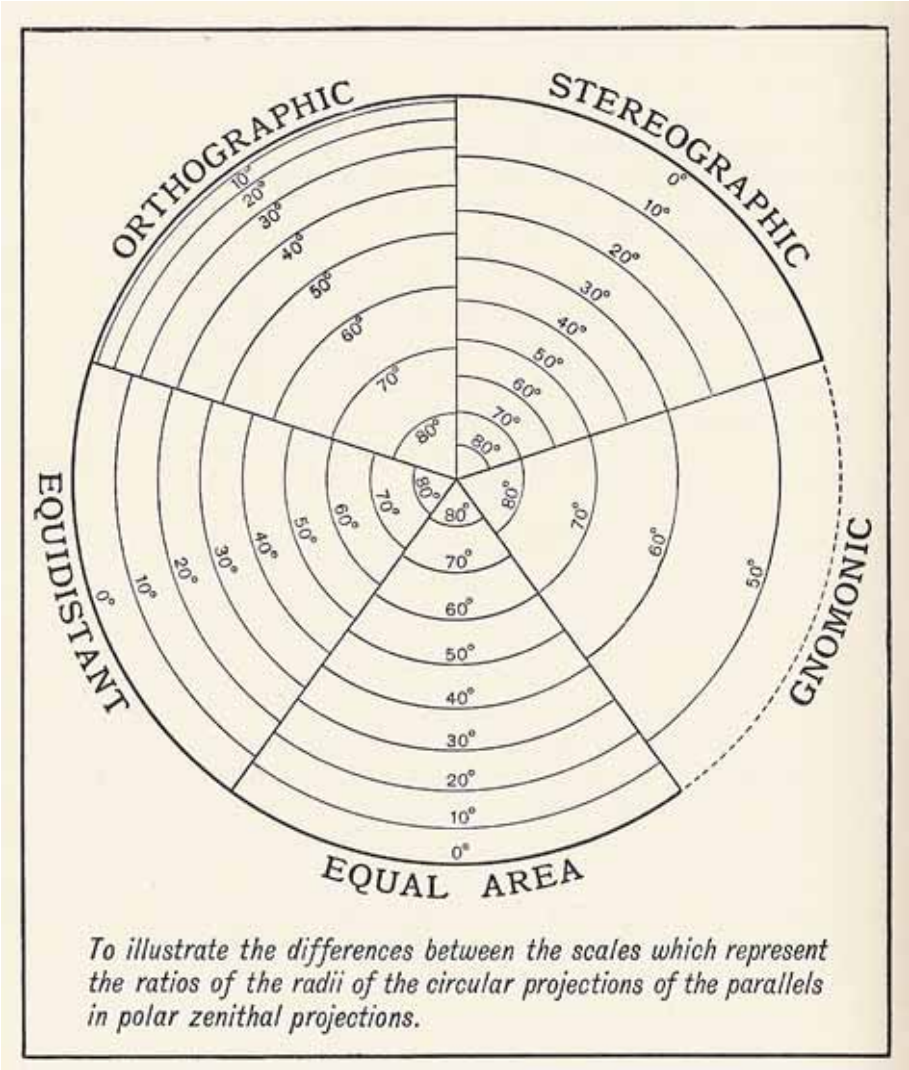
All categories on Zenithal Projections can be broken down yet again into further specification, where the meridians of longitude and parallels of latitude are represented with different proportions. An example of the different types of *Polar* Projections can be seen in figure 11.4 [17](p. 52).

11.1.8 Orbital Elements

Orbital elements are objects which have an orbit. These include the Planets in our solar system, the moon, comets and satellites. After conducting some serious research into orbital elements it seems there are many accurate methods of calculating the position based on a system of interpolation. This system provides great accuracy, however it requires a vast array of tabulated data as well as demanding computational power to formulate a result [19](p. 223).

Fortunately a less accurate (over time) method can be used to calculate positions relying on vastly more simplistic computation and calculations [19](p. 227). If orbital elements are to be included in the proposed system as suggested by the possible extensions, the only option would be to use this less accurate method. The positions of the orbital elements are calculated using the apparent position of the sun at a given time. Fortunately, the target environment for the proposed system

Figure 11.4: Polar Zenithal Projections



has a very limited display so extreme accuracy is not required and close enough calculations will suffice.

11.1.9 Star Colour Spectrum types [7]

Spectral Type	Colour	Surface Temperature(C)
O	Blue	>30,000
B	Blue-White	20,000
A	White	10,000
F	Yellow-White	7,000
G	Yellow	6,000
K	Orange	4,500
M	Red	3,000

11.1.10 The Greek alphabet.

Greek	*English*	*Greek*	*English*
α	alpha	ν	nu
β	beta	ξ	xi
γ	gamma	o	omicron
δ	delta	π	pi
ϵ	epsilon	ρ	rho
ζ	zeta	σ	sigma
η	eta	τ	tau
θ	theta	υ	upsilon
ι	iota	ϕ	phi
κ	kappa	χ	chi
λ	lambda	ψ	psi
μ	mu	ω	omega

11.2 Requirements Analysis

11.2.1 Domain Analysis

picoSky

"picoSky 'A Lot of Sky in a Little Space' is a tiny Java application that harnesses the computational power of your cell phone to display a graphical representation of the night sky, for your location and time" [18]. picoSky is available to buy directly from the Internet. Several screen shots of picoSky can be seen in Figure 11.5.

picoSky appears to be the main J2ME application available to buy over the Internet, boasting an impressive list of desirable features; an attractive and informative user manual is supplied; star spectrum's are represented using an effective use of colour; proportional star sizes are based on star magnitudes. however, the initial setup process is aimed directly at the American market, making it difficult for users in another areas of the world to correctly setup the application. Location selection will have to be carefully considered to avoid this problem.

Mobile Planetarium

The Mobile Planetarium for Java-Enabled Mobile Phones is an open source J2ME application presenting a user with a view of the night sky based on their location. It is part of `sourceforge.net` and is fully available in open source manner [34].

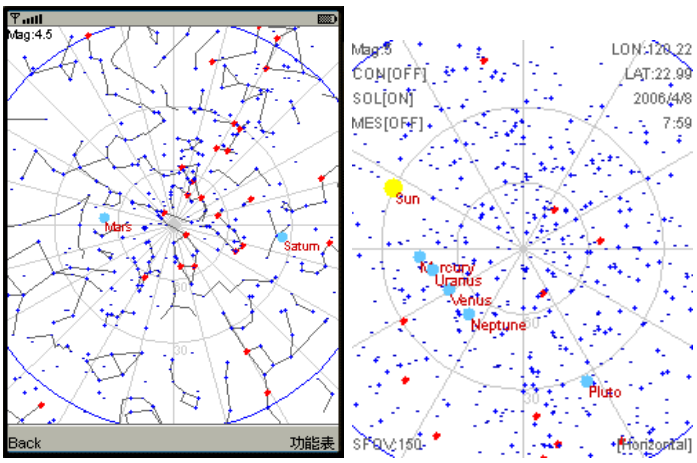
Figure 11.5: Images presented on the picoSky website.

The Mobile Planetarium boasts many very impressive features to aid star gazers. Most interestingly that the designer has decided to split the controls of the system into two clear key modes, *Normal* and *Time*. In *Normal mode* a user may toggle the numerous available objects of interest for example; constellation names and lines, grids and star names. While in *Time mode* a user may control the time intervals and the direction of time - to the future or past. Universal keys are also used for common tasks, for example zooming in and out. Below is the control mode graphical representation given on the product website within the manual section [35], see Figure 11.7.

Although the source code of the application is provided, no compiled version is available that is ready to install directly onto a mobile phone. Having downloaded the source code in full, I was unable to compile and run the application on either my real or emulated mobile phone. It would seem that although many of the calculations and program flows are provided the application is mainly used as a point of reference.

Fortunately, the website offers a list of the functionalities provided and also some screen shots, see Figure 11.6.

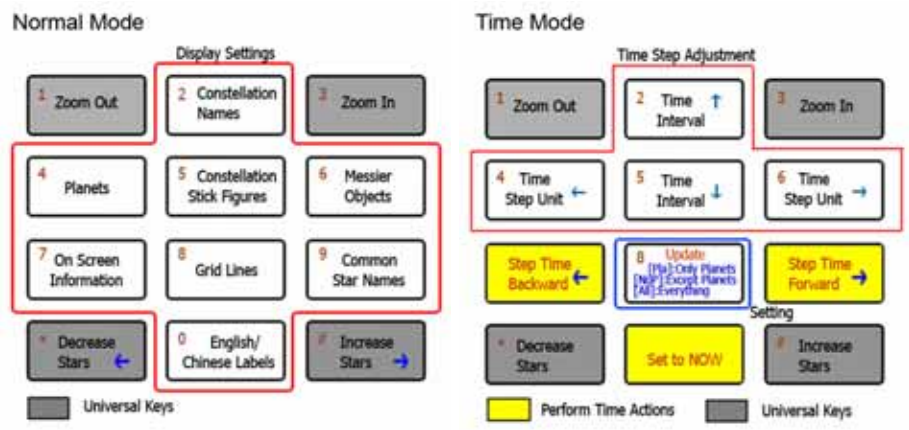
Figure 11.6: Images presented on the Mobile Planetarium website.



Stellarium

Stellarium is an open source planetarium for your computer. It is a fully featured desktop application available for most operating systems including Windows, Mac OSX and Linux. For a free

Figure 11.7: Mobile Planetarium control modes.



open source application, Stellarium is a useful tool for aiding stargazing, unfortunately the key elements which create this application are mostly unobtainable by a mobile phone [36].

Figure 11.8: Stellarium screen shots.

11.2.2 picoSky application Mapping

- Show Sky
  - shows a circle containing the stars and constellations?
  - two always visible options:
    - ∗ : Info
      - · no celestial object selected. Use the * or hash Keys to select an object
      - · Constellation: 'Cepheus', Circumpolar, RA: 22h 25m, DEC: 72 34', Alt: 47 37', Az: 25 59'
    - ∗ : More
      - · Zoom in
      - · Zoom Out
      - · Back
      - · five levels of zoom from fully zoomed out to fully zoomed in
- Current Time

  - date 17-Oct-2007
  - Observing DST
  - Local: 4:24:56 PM
  - UT: 11:24:56PM
  - GMST: 1:09:16AM
  - LMST: 5:00:44PM
  - Julian Day: 2454391.47
- Solar Systems
  - Sun
  - Moon
  - Mercury
  - Venus
  - Mars
  - Jupiter
  - Saturn
  - Uranus
  - Neptune
  - Pluto
- Search
  - box saying name defaulted with M31
  - successful search displays constellation name of object
  - additional information provides:
    - rise: e.g. 12:52PM
    - transit: 12:06PM
    - Set: 11:21AM
    - RA: 0h 43M
    - DEC: 41 16'
    - alt: 15 19'
    - az: 44 21'
- Options
  - Rendering
    - constellations (show/dont show)
    - Deep Sky (show/dont show)
    - font size - small, medium or large
    - alt/az grid (show/dont show)
  - set date and time

    - time (present / fixed)
    - displays the fixed time
- time format
    - 24 military or 12 civil
- units
    - miles or kilometers
- recalc rate
    - ability to enter number of minutes
- Site Information (details about location!)
    - by Area Code (US only)
    - Latitude (degrees, minutes and seconds)
    - Longitude (degrees, minutes and seconds)
    - Time Zone
- Help
    - gives a small image of the controls, which are
        - joypad is up, down, left, right
        - 1 = NE
        - 2 = N
        - 3 = NW
        - 4 = E
        - 5 = LOOK UP?
        - 6 = W
        - 7 = SE
        - 8 = S
        - 9 = SW
        - 0 = next screen
        - * = next label
        - hash = previous label
    - Latitude (degrees, minutes and seconds)
    - Longitude (degrees, minutes and seconds)
    - Time Zone
- About
    - gives product name
    - current platform - including CLDC version, and MIDP
- Register
    - asks for a registration code.

11.2.3 Questionnaire Results from 40 responses

1. Have you ever identified a constellation or star in the night sky? 73%.
2. Have you ever used an application which creates a view of the night sky? 18%.
3. If asked would you know your approximate longitute and latitude position on earth? 20%
4. Longitude and latitude information is needed for accurate star-gazing, how would you most like an application to provide you with these details?
   (a) 48% A point and click map of the world
   (b) 28% A searchable List of locations
   (c) 43% Automatically Approximated
5. If you have ever looked up at the night sky did you use any of the following?
   (a) 50% Telescope
   (b) 53% Binoculars
   (c) 15% Compass
   (d) 3% Planisphere
   (e) 15% Guide Book
   (f) 3% Astrolabe
6. Which pieces of information would you be interested about when looking at a particular star?
   (a) 90% The stars name
   (b) 33% Brightness of the star
   (c) 70% The distance of the star
   (d) 65% The constellation of the star
   (e) 18% Celestrial Coordinates
7. Which of the following would you be most interested to know in terms of upcoming night sky events?
   (a) 88% Comets
   (b) 40% Satellites
   (c) 80% Lunar Eclipses
   (d) 78% Meteor showers
   (e) 78% Planets
8. Does your mobile phone have a colour screen? 88%
9. Does your mobile phone have games? 93%
10. Does your mobile phone have an internet connection? Can you surf the web? 63%
11. If you had an application on your mobile phone, how often would you be tempted to use it to aid looking at the night sky?
    (a) 0% daily
    (b) 3% weekly

(c) 0% fortnightly

(d) 0% monthly

(e) 0% annually

(f) 97% unknown

12. In which situations would you most be inclined to use such an application?

(a) 60% holiday

(b) 60% walks

(c) 35% garden

(d) 18% parties

13. Please take a look at the following very cut down basic frames. A representation of the night sky is required. Which of the following is your personal favorite and could potentially aid star gazing. To explain, imagine when looking at the night sky you are seeing it as a great sphere (the celestial sphere) - which image matches your perception?

(a) 48% Polar

(b) 35% Normal

(c) 10% Oblique

14. If you were to buy such an application, how much would you be prepared to pay?

(a) 43% nothing

(b) 3% 0.50

(c) 13% 1.50

(d) 25% 5.00

(e) 13% 10.00

(f) 3% 20.00

15. Which of the following date/time formats would you be most happy to use?

(a) 57% DD/MM/YYYY HH:MM (24 hour)

(b) 0% DD/MM/YYYY HH:MM (12 hour)

(c) 40% DD/MM/YY HH:MM (24 hour)

(d) 3% DD/MM/YY HH:MM (12 hour)

16. If this application existed right now, how would you most like to receive the application for use with your phone?

(a) 40% Direct Download via Internet on Phone

(b) 40% Download and install via desktop computer

(c) 18% Recieve download link via text message

(d) 0% CD / DISK via Post

17. In order to learn the controls and features of the application would you most like...

(a) 23% A detailed user manual

(b) 35% A graphical image displaying controls

(c) 35% An animated movie guide

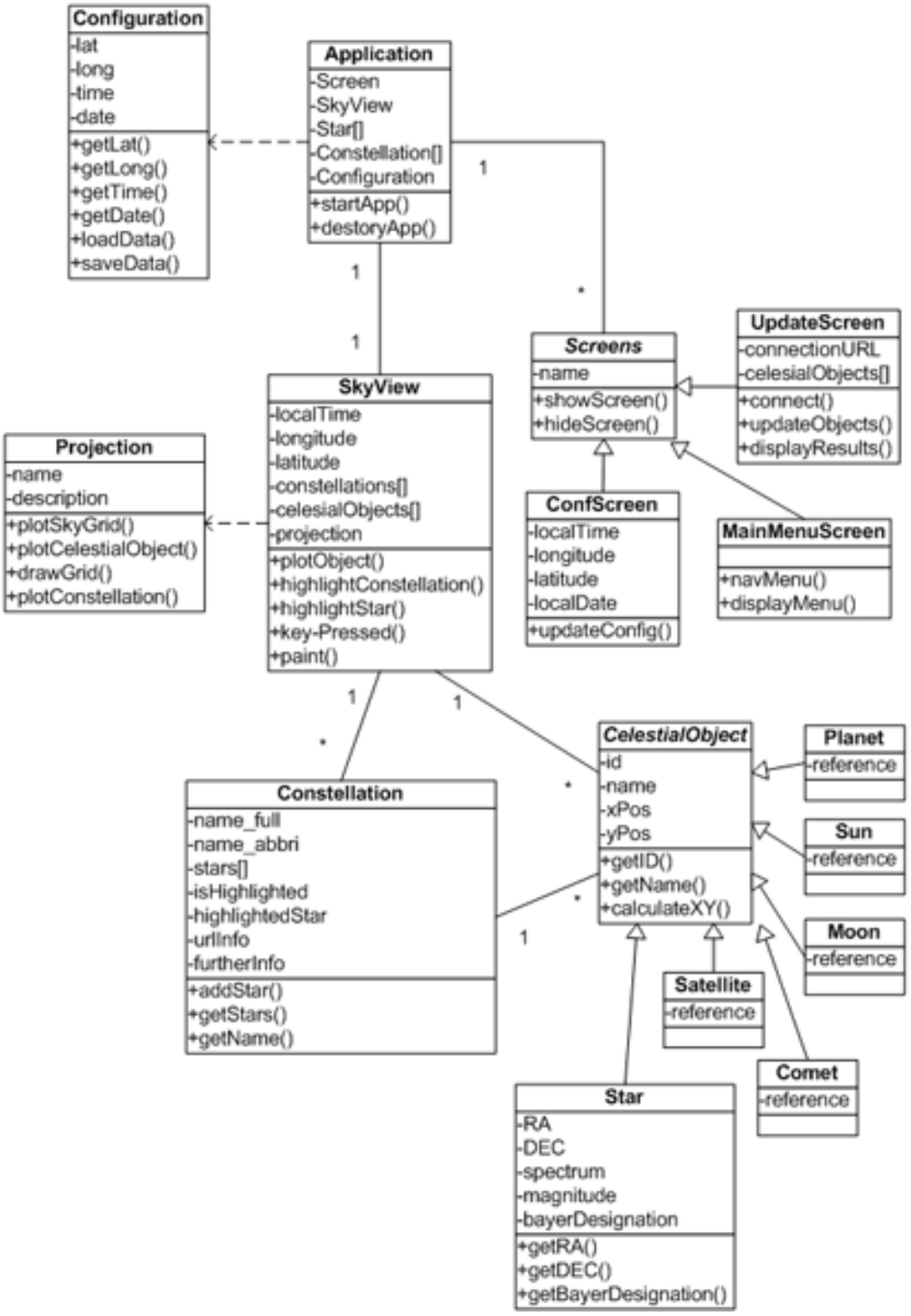
11.2.4 Questionnaire Results : Comments

- Red light mode, to save night vision
- Picture sharing community with photos taken of the stars by users of the application
- Possibly the historical reasons why the constellations are named as they are, along with any other relevant information.
- The colour of the star you are looking at. -Discovery and information updates. i.e. star of the week!.. today BBC news reports Berkley university have discovered a 5th planet around a star cephi55 i think.. i would like the app to tell me that and where... even if i cant see because its too dim... don't underestimate the imagination of the viewer. Some successful way of converting the complex numbering, indexing and information system of storing star info into easy to understand information.. e.g. converting magnitude info, to "just visible in town" "just visible in the country" etc. Best of luck!
- A built-in compass! (Presumably using GPS)
- Little stories behind the name of stars and maybe who found them.
- Automatic determining of position on Earth on phones equipped with GPS

11.2.5 Key Codes

Key	Code	Key	Code
1	49	2	50
3	51	4	52
5	53	6	54
7	55	8	56
star	42	0	48
hash	35	call	-10
end call	-11	menu left	-6
menu right	-7	back	-8
JoypadUp	-1	JoypadDown	-2
JoypadLeft	-3	JoypadRight	-4
JoypadSelect	-5		

11.3 System Overview



11.4 J2ME Game File Size

Game	File Size
FotoQuest Fishing [3]	503 KB
sony ericsson WTA tour	351 KB

11.5 User Details

The following three users were used throughout the user evaluation.

1. **User 1**
   - Gender: Male
   - Age: 27
   - Computer Literacy: Average
   - Mobile phone Literacy: Average
   - Played mobile phone games? YES
   - Used mobile Internet? YES
   - Used mobile Applications? YES
2. **User 2**
   - Gender: Female
   - Age: 54
   - Computer Literacy: Average
   - Mobile phone Literacy: Below Average
   - Played mobile phone games? NO
   - Used mobile Internet? NO
   - Used mobile Applications? YES
3. **User 3**
   - Gender: Female
   - Age: 24
   - Computer Literacy: Above Average
   - Mobile phone Literacy: Above Average
   - Played mobile phone games? YES
   - Used mobile Internet? YES
   - Used mobile Applications? YES

11.6 Prototype 1: User Testing

The initial prototype was developed from the system overview presented in the requirements analysis phase of the project.

11.6.1 User 1

Key Findings:

1. Menus need to have consistent and obvious options to go forward and backwards.
2. User felt that the 15 second loading time was acceptable when presented with a visual indication of progress.
3. Wanted to know what buttons/controls to use. This is not currently obvious.
4. Didn't like the large amount of text regarding the number of visible constellations.
5. Felt the display was too 'busy', wanted to see less on the initial screen and more when zooming in.
6. Felt strongly that the loading time for collecting constellation information from the Internet was too slow.

11.6.2 User 2

Key Findings:

1. Wanted more helps understanding what they are looking at (in terms of how to match with the sky).
2. Also felt the display was too 'busy'.
3. Felt that the loading time was acceptable, however was getting bored.

11.6.3 User 3

Key Findings:

1. Wanted a demo to quickly show how to use the application.
2. Wanted a search feature to help quickly locate objects of interest.
3. Unhappy with the slow speed to get constellation information from the Internet.

11.7 Loading Time vs Application Size

Test Details
- up to mag 6 stars.
- 9110 stars in total.

Static star data prototype

app installed filesize: 1354KB // 1.32226562 megabytes
initial loading time: 4 second "please wait menu displayed" - preparing satic data

FIRST LOAD
103 ms to plot and draw all stars.
96 ms to plot and draw all stars.
94 ms to plot and draw all stars.

Loading data from files

app installed filesize: 382KB // 0.373046875 megabytes
initial loading time: 1 second "please wait menu displayed" - NO preparing of satic data

FIRST LOAD
20 seconds to plot and draw all stars.
24 seconds to plot and draw all stars.
17 ms to plot and draw all stars.

both performed equally as well following first load. when both have all stars in memory.

however, with up to Mag 6, there was simply too many stars, display was far to "busy"

program adjusted to mag 5 or brighter stars

up to mag 5 stars.5057 stars in total.

Static star data prototype

app installed filesize: 845KB // 0.825195312 megabytes megabytes
initial loading time: 2 second "please wait menu displayed" - preparing satic data

FIRST LOAD
98 ms to plot and draw all stars.
95 ms to plot and draw all stars.
91 ms to plot and draw all stars.

Loading data from files

app installed filesize: 319KB // 0.311523438 megabytes
initial loading time: 0.5 second "please wait menu displayed" - NO preparing of satic data

FIRST LOAD
10 seconds to plot and draw all stars.
11 seconds to plot and draw all stars.
8 seconds to plot and draw all stars.

11.8 Record Store Prefixing

Prefix	Data
*	Location Option Index
@	Country Name
#	City Name
%	Longitude
$	Latitude
&	Projection Option Index
~	Colour Option Index
£	Show Grid
-	Recalculation Interval
=	Show Bayer Designations

11.9 Testing

11.9.1 Unit Testing

Julian Date (Calculations.julianDate())

Calculate the JD corresponding to 1957 October 4.81, the time of launch of Sputnik 1. [19](p. 61)

Correct Answer: 2436116.31
System Answer: 2436116.31

This is an essential for the calculation of celestial object positions.

Mean sidereal Time (Calculations.meanSiderealTime())

Find the mean sidereal time at Greenwich on 1987 April 10 at $19^h21^m00^s$ [19](p. 89)

Correct Answer: $\theta_0 = -1677831.262$
System Answer: $\theta_0 = -1677831.262$

This is an essential for the calculation of celestial object positions.

Sun's Position (Sun.calculatePosition())

Calculate the Sun's position on 1992 October 13 at 0^h TD. [19](p. 165)

Correct Answer:

$\boldsymbol{T}$ = **-0.072183436**
L_0 = **-201.80720**
M = **278.99397**
e = **0.013711668**
C = **-1.89732**
$\odot$ = **199.90988**
R = **0.99766**
ω = **264.65**
λ = **199.90895**
ϵ_0 = **23.44023**
ϵ = **23.43999**
α = **198.38093**
δ = **-7.78507**

System Answer:

$\boldsymbol{T}$ = **-0.07218343600273786**
L_0 = **-2318.1928034932926**
M = **278.9939664315975**
e = **0.01671166771493543**
C = **-1.897323843371984**
$\odot$ = **199.9098726633356**
R = **0.9976619500056293**
ω = ***omitted**
λ = ***omitted**
ϵ_0 = **23.44023**
ϵ = ***omitted**
α = **198.381662821727937**
δ = **-7.78549982837678**

Rectangular Coordinates of the Sun (Calculations.orbital_to_ra_dec())

For 1992 October 13.0 TD = JDE 2448908.5, calculate the equatorial rectangular coordinates of the Sun referred to the standard equinox J2000 [19](p. 175)

Correct Answer:

***X* = -0.93739590**
***Y* = -0.31316793**
***Z* = -0.13577924**

System Answer:

***X* = -0.9373668244336052**
***Y* = -0.3133864487796232**
***Z* = -0.13587397345226737**

This calculation is essential for plotting the position of planets in the application

Equation of Kepler (Calculations.kepler())

Solve the equation of Kepler for $e = 0.100$ and M = 5, to an accuracy of 0.000001 degree [19](p. 196).

Correct Answer: **E = 5.554589**
System Answer: **E = 5.554589253872315**

Kepler equation is required for calculation of ecliptic orbits.

Mean Orbital Elements (Planets.calculatePosition())

Calculate the mean orbital elements of Mercury on 2065 June 24 at 0^h TD [19](p. 211).

Correct Answer:

$\boldsymbol{T}$ **= +0.654770704997**
L **= 203.494701**
a **= 0.387098310**
e **= 0.205645108**
i **= 7.006171**
ω **= 49.107650**
π **= 78.475382**
M **= L -** π **= 125.019319**
$w = \pi - \omega$ **= 29.367732**

System Answer:

$\boldsymbol{T}$ **= +0.6547707049965776**
L **= 202.57945270944037**
a **= 0.38709831**
e **= 0.20564509972233957**
i **= 7.001089421722686**
ω **= 48.248731964904515**
π **= 77.56013293425961**
M **= L -** π **= 125.01931977518076**
$w = \pi - \omega$ **= 29.3114009693551**

Mean orbital elements are required for calculating the position of a planet.

Ecliptic Orbits (Calculations.orbital_to_ra_dec())

Calculate the geocentric position of the periodic comet Encke for 1990 October 6.0 Dynamic Time [19](p. 232).

Correct Answer:

$\alpha = 158.558965$
$\delta = +19.158496$

System Answer:

$\alpha = 158.57451789877803$
$\delta = +19.150955081306474$

This calculation is used to plot all of the planets (from mean orbital elements) and potentially comets with ecliptic orbits.

Position of the Moon (Moon.calculatePosition())

Calculate the geocentric longitude, latitude, distance, and equatorial horizontal parallax of the Moon for 1992 April 12, at 0^h TD [19](p. 342).

Correct Answer:

$\alpha = 134.688470$
$\delta = +13.768368$

System Answer:

$\alpha = 134.68765607050497$
$\delta = +13.767372454574257$

The position of the moon is almost identical.

11.9.2 Comparison Testing

Three different locations and dates will be used to compare the night sky at a single time of day between the finished application and the Sky Charts available from `zwww.heavensabove.com`. Heavens above was selected as it provided the clearest, most simplistic and easily available charts providing the ability to customize date, time and location. Stellarium was also used as a second comparison.

Please note that there are no official standards for the constellation figure lines, so some or all of the constellations may not appear identical in form.

Locations:

1. Reigate, Surrey, UK (lat: 51.24, long: 0.22)
2. Nairobi, Kenya, Africa (lat: -1.26 long: 36.8)
3. Funafuti, Tuvalu, Pacific Ocean (lat: -8.5200 long: 179.2200)

Dates:

1. 4th November 1982
2. 31st December 1999
3. 22nd March 2015

Time:

1. 4:00AM

I am happy with the outcomes of the comparison tests and feel that the sky is being correctly rendered in a useful manner for aiding star gazing. I am confident that a fairly accurate representation in line with existing applications is being produced.

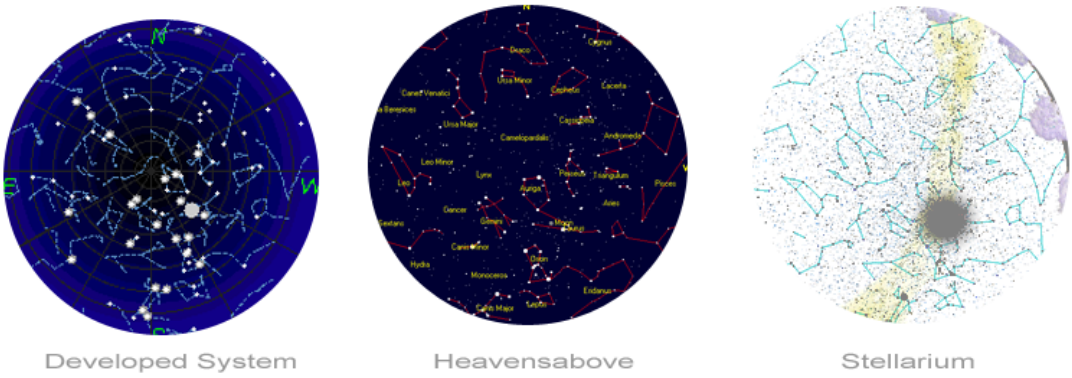


Figure 11.9: Reigate, Surrey, UK : 4th November 1982 @ 4:00

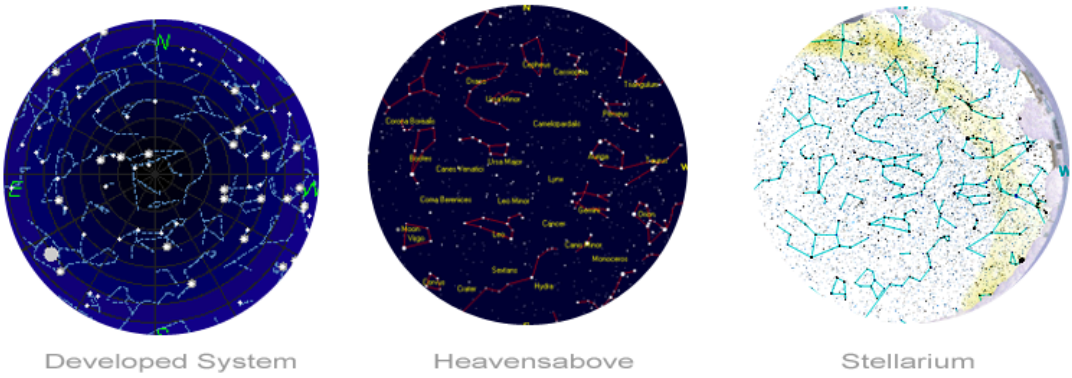


Figure 11.10: Reigate, Surrey, UK : 31st December 1999 @ 4:00

Figure 11.11: Reigate, Surrey, UK : 22nd March 2015 @ 4:00

Figure 11.12: Nairobi, Kenya, Africa : 4th November 1982 @ 4:00

Figure 11.13: Nairobi, Kenya, Africa : 31st December 1999 @ 4:00

Figure 11.14: Nairobi, Kenya, Africa : 22nd March 2015 @ 4:00

Figure 11.15: Funafuti, Tuvalu, Pacific Ocean : 4th November 1982 @ 4:00

Figure 11.16: Funafuti, Tuvalu, Pacific Ocean : 31st December 1999 @ 4:00

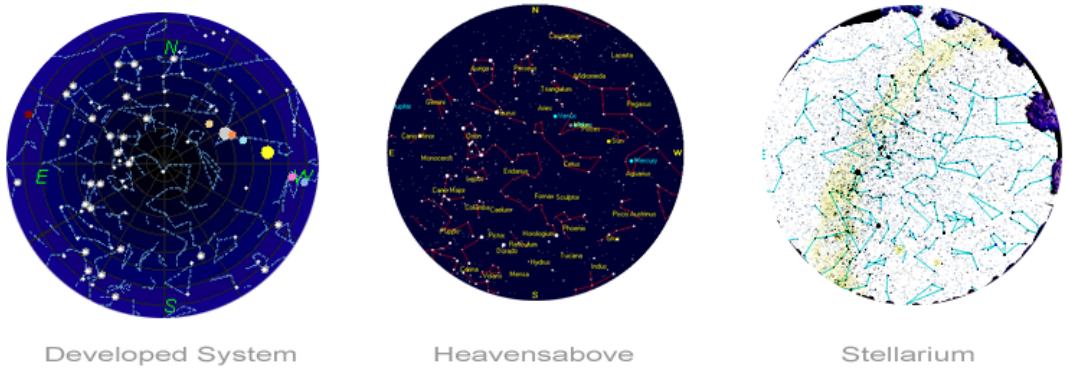


Figure 11.17: Funafuti, Tuvalu, Pacific Ocean : 22nd March 2015 @ 4:00

11.10 Project Logs

11.10.1 Autumn Term

Wednesday 22nd August 2007
Initial meeting with Dan Chalmers to discuss a shortlist of possible project ideas that was compiled over the summer holidays. The project involving creating a star chart application for mobile phones was decided on.
Work completed: J2ME testing started and astronomy research.

Tuesday 9th October 2007
Discussion of project progress and meeting other students who are also being supervised by Dan Chalmers. It was decided that we should complete our project proposal and begin thinking about time management and a project plan.
Work completed: Project plan created and proposal document.

Tuesday 12th October 2007
Starting Domain analysis was advised at this point. Looking into similar products will allow certain functionalities to emerge and also where other products on the market currently lack.
Work completed: Questionnaire created and domain analysis begun.

Tuesday 23rd of October 2007
Recommended to begin looking into sources of data for the project. This includes star data, update information - such as upcoming events - comets, satellites etc. Beginning investigation of possible projections of the sky was also suggested.
Work completed: Use cases, state and sequence diagrams created.

Tuesday 30th October 2007
Discussing relevant HCI issues, prioritizing and update about report progress.
Work completed: Professional considerations reviewed and LOW-FI prototype created.

Tuesday 6th November 2007
Clear that further investigation into astronomy is required. Thoughts about possible difficulties with the vast array of different algorithms and data sets to be prioritized.
Work completed: User evaluations and examining of questionnaire results.

Tuesday 13th November 2007 Process models and testing approaches were considered topics of interest. Draft report to be handed in at least one week before deadline. **Work completed:** Class and method identification.

Tuesday 20th November 2007
Discussions into the correct use of references to be used within the interim report.
Work completed: Writing up research and results.

Tuesday 27th November 2007
Work completed: Submission of draft interim report

Tuesday 4th December 2007
Work completed: Submission of interim report.

11.10.2 Christmas Holiday

- Creation of first prototype from system overview.
- Resource file conclusions
- Creation of second prototype.

11.10.3 Spring Term

Monday 7th January 2008
First term meeting with supervisor, design and implementation is to begin.
Work completed: 1st J2ME prototype completed over Christmas holiday.

Monday 14th January 2008
Suggestions on possible design document elements.
Work completed: User evaluation of first prototype conducted.

Monday 21st January 2008
Thinking about possible testing strategies.
Work completed: Following feedback and suggestions, second prototype began constructed.

Monday 28th January 2008
Possible design issues and strategies discussed.
Work completed: High level design draft started, moon position implemented.

Monday 4th February 2008
Discussed Threading policies and ideas for improving the loading speed of the application.
Work completed: High level design finished and further development of application made.

Monday 11th February 2008
Possible server platforms and PHP design considerations for UML.
Work completed: Implementation of search functionalities to application

Monday 18th February 2008
Work completed: Sun and Planet positions research and user interface design work.

Monday 25th February 2008
Work completed: Sun and Planet positions implemented.

Monday 3rd March 2008
More discussion on possible testing techniques.
Work completed: report structure review and further writeup.

Monday 10th March 2008
No meeting this week.
Work completed: draft report almost complete.

11.10.4 Easter Holiday

- completed of application client / server mechanics.
- Final user testing conducted.
- System comparison testing.
- Location API implementation for Orientation.
- Testing on a second mobile phone device.
- Application website constructed.
- Final report completed.

11.11 Different Phones

Figure 11.18: Sony Ericsson K800i

Figure 11.19: Samsung G600

Figure 11.20: Sony Ericsson S500i

11.12 Field Testing

User Questionnaire

1 – age: 21 gender: MALE

2 – How did you find the application menus?

[ ] difficult [✓] ok [ ] easy

Comments:

3 – What did you think about the Location selection?

[ ] difficult [✓] ok [ ] easy

Comments:

4 – What did you think about the Loading Screen?

[ ] difficult [ ] ok [✓] easy

Comments:

5 – How did you find the Skyview?

[ ] difficult [✓] ok [ ] easy

Comments:

6 – How did you find the controls?

[ ] difficult [ ] ok [✓] easy

Comments:

7 – What did you think about the SkyView Layout?

[ ] difficult [✓] ok [ ] easy

Comments:

8 – What did you think about the colours/fonts?

[ ] difficult [ ] ok [✓] easy

Comments:

9 – How did you find spotting a constellation? (perception)

[ ] difficult [✓] ok [ ] easy

Comments:

10 – How did you find spotting a bright object? (perception)

[ ] difficult [ ] ok [✓] easy

Comments:

11 – What were your preferred settings?

Comments:

12 – Would you use this application again?

[✓] Yes [ ] No

Comments:

Definitely. In the summer when it is warm at night I could spend ages using it.

13 – Did you find the User Manual useful?

[✓] Yes [ ] No

Comments:

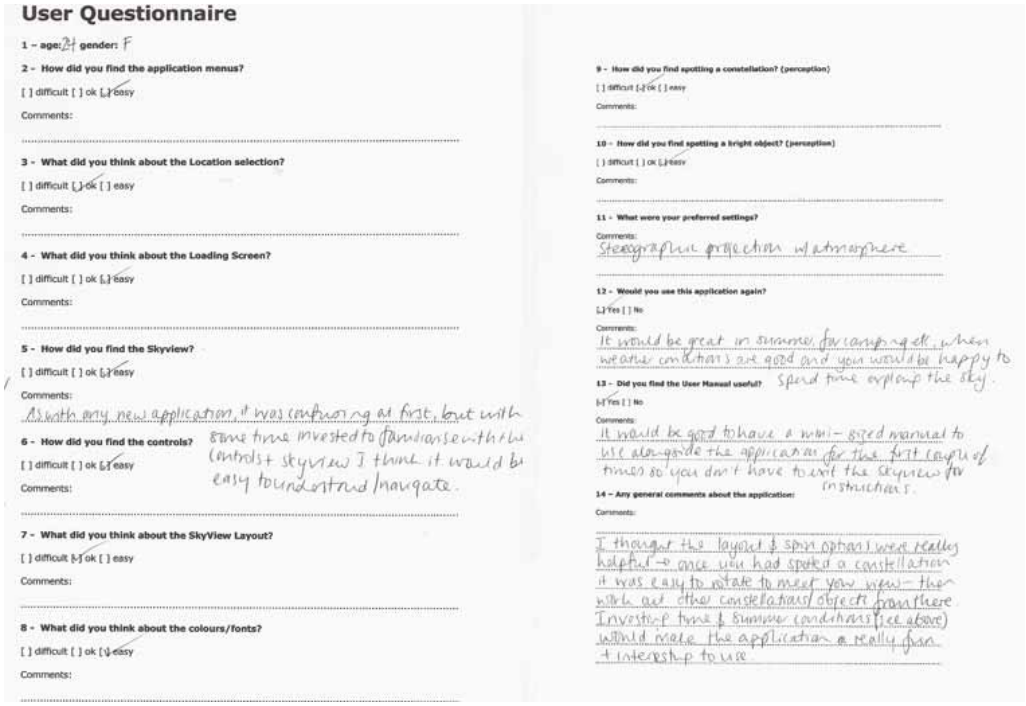
Without having read how to use the controls I wouldn't have felt as confident using the application

14 – Any general comments about the application:

Comments:

Very good

Figure 11.21: Questionnaire one



User Questionnaire

1 – age: 21 gender: F

2 – How did you find the application menus?

[ ] difficult [ ] ok [✓] easy

Comments:

3 – What did you think about the Location selection?

[ ] difficult [✓] ok [ ] easy

Comments:

4 – What did you think about the Loading Screen?

[ ] difficult [ ] ok [✓] easy

Comments:

5 – How did you find the Skyview?

[ ] difficult [ ] ok [✓] easy

Comments:

As with any new application, it was confusing at first, but with

6 – How did you find the controls? some time invested to familiarise with the

[ ] difficult [ ] ok [✓] easy controls + skyview I think it would be

Comments: easy to understand/navigate.

7 – What did you think about the SkyView Layout?

[ ] difficult [✓] ok [ ] easy

Comments:

8 – What did you think about the colours/fonts?

[ ] difficult [ ] ok [✓] easy

Comments:

9 – How did you find spotting a constellation? (perception)

[ ] difficult [✓] ok [ ] easy

Comments:

10 – How did you find spotting a bright object? (perception)

[ ] difficult [ ] ok [✓] easy

Comments:

11 – What were your preferred settings?

Comments:

Stereographic projection w atmosphere

12 – Would you use this application again?

[✓] Yes [ ] No

Comments:

It would be great in summer, for camping etc, when weather conditions are good and you would be happy to spend time exploring the sky.

13 – Did you find the User Manual useful? [✓] Yes [ ] No

Comments:

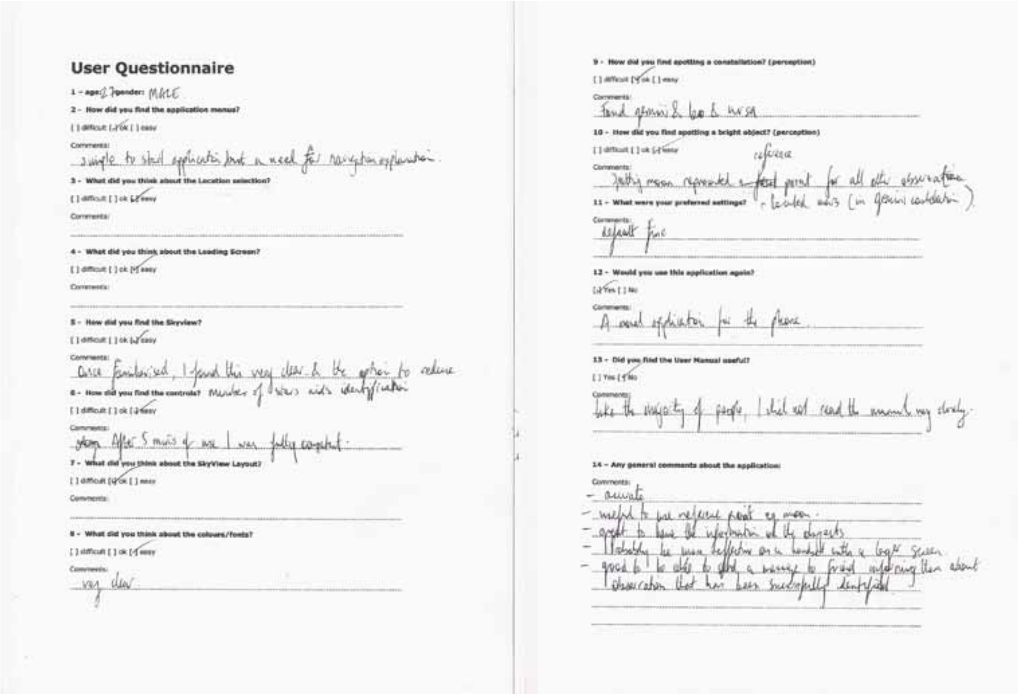
It would be good to have a mini-sized manual to use alongside the application for the first couple of times so you don't have to exit the skyview for instructions.

14 – Any general comments about the application:

Comments:

I thought the layout & spin options were really helpful - once you had spotted a constellation it was easy to rotate to meet your own - then work out other constellations/objects from there. Investing time & summer conditions (see above) would make the application a really fun + interesting to use

Figure 11.22: Questionnaire Two



User Questionnaire

1 – age: 2? gender: MALE

2 – How did you find the application menus?

[ ] difficult [✓] ok [ ] easy

Comments: simple to start application but a need for navigation explanation.

3 – What did you think about the Location selection?

[ ] difficult [ ] ok [✓] easy

Comments:

4 – What did you think about the Loading Screen?

[ ] difficult [ ] ok [✓] easy

Comments:

5 – How did you find the Skyview?

[ ] difficult [ ] ok [✓] easy

Comments: Once familiarised, I found this very clear & the option to reduce number of stars aids identification

6 – How did you find the controls?

[ ] difficult [ ] ok [✓] easy

Comments: After 5 mins of use I was fully competent.

7 – What did you think about the SkyView Layout?

[ ] difficult [✓] ok [ ] easy

Comments:

8 – What did you think about the colours/fonts?

[ ] difficult [ ] ok [✓] easy

Comments: very clear

9 – How did you find spotting a constellation? (perception)

[ ] difficult [✓] ok [ ] easy

Comments: found gemini & leo & ursa

10 – How did you find spotting a bright object? (perception)

[ ] difficult [ ] ok [✓] easy

Comments: Getting moon represented a fixed reference point for all other observations - located stars (in gemini constellation)

11 – What were your preferred settings?

Comments: default fine

12 – Would you use this application again?

[✓] Yes [ ] No

Comments: A neat application for the phone.

13 – Did you find the User Manual useful?

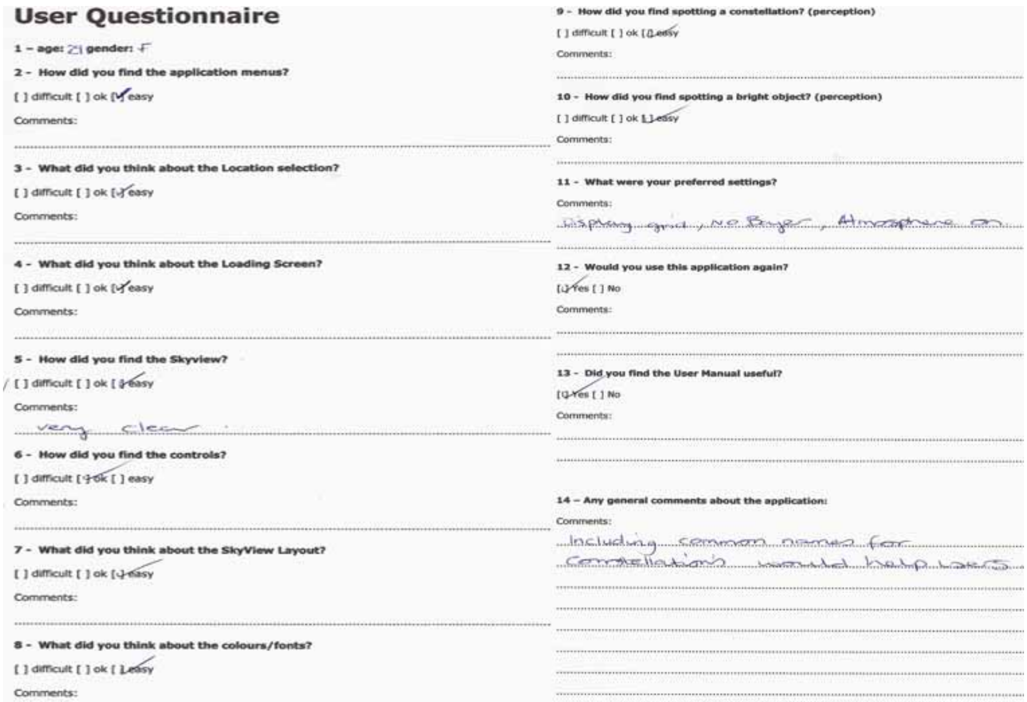
[ ] Yes [✓] No

Comments: Like the majority of people, I did not read the manual very closely.

14 – Any general comments about the application:

Comments:
- accurate
- useful to use reference point eg moon.
- great to have the information on the objects
- Probably be more effective on a handset with a larger screen
- good to be able to send a message to friend informing them about observation that has been successfully identified

Figure 11.23: Questionnaire Three



User Questionnaire

1 – age: 21 gender: F

2 – How did you find the application menus?

[ ] difficult [ ] ok [✓] easy

Comments:

3 – What did you think about the Location selection?

[ ] difficult [ ] ok [✓] easy

Comments:

4 – What did you think about the Loading Screen?

[ ] difficult [ ] ok [✓] easy

Comments:

5 – How did you find the Skyview?

[ ] difficult [ ] ok [✓] easy

Comments: very clear

6 – How did you find the controls?

[ ] difficult [✓] ok [ ] easy

Comments:

7 – What did you think about the SkyView Layout?

[ ] difficult [ ] ok [✓] easy

Comments:

8 – What did you think about the colours/fonts?

[ ] difficult [ ] ok [✓] easy

Comments:

9 – How did you find spotting a constellation? (perception)

[ ] difficult [ ] ok [✓] easy

Comments:

10 – How did you find spotting a bright object? (perception)

[ ] difficult [ ] ok [✓] easy

Comments:

11 – What were your preferred settings?

Comments: Display grid, No Boxes, Atmosphere on

12 – Would you use this application again?

[✓] Yes [ ] No

Comments:

13 – Did you find the User Manual useful?

[✓] Yes [ ] No

Comments:

14 – Any general comments about the application:

Comments: Including common names for constellations would help users

Figure 11.24: Questionnaire Four

11.13 Physical Media

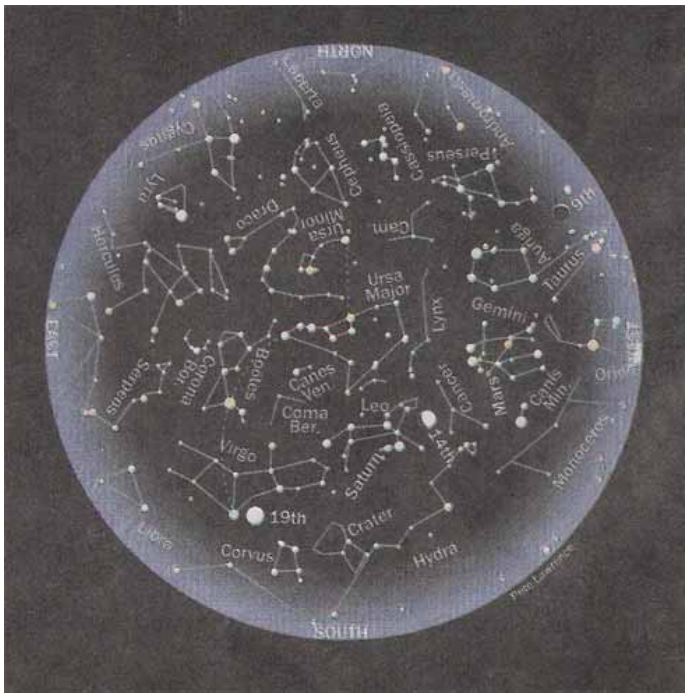


Figure 11.25: Times Sky Map

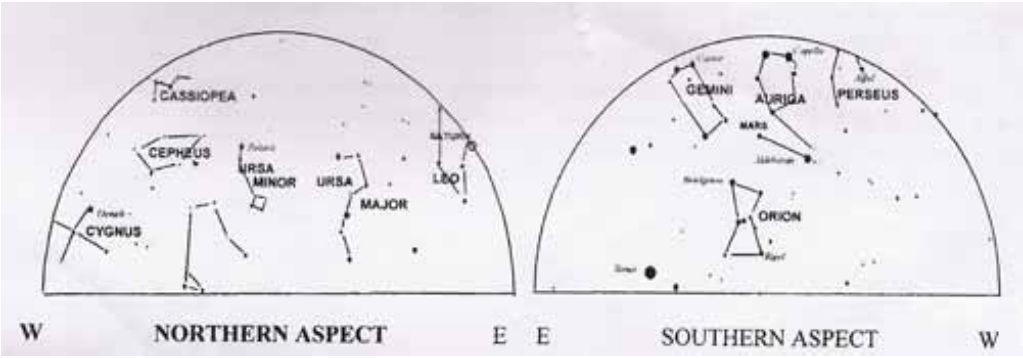


Figure 11.26: Sky at Night NewsLetter