Detailed Design Document

for

HP Capital Assets Management

Version 2.0

Prepared by xxxxx

Purdue CS 307 Fall 2007

October 1, 2007

Table of Contents

1. Introduction

1.1 Document Scope

The purpose of this document is to detail in a fine granularity how the project is expected to work from a structural standpoint. This document includes Use Cases for different parts of the application, as well as UML 2.0 diagrams detailing the program's components and static structure. Other information offers Design Principles and Guidelines, as well as object interaction within the program.

1.2 Intended Audience

This document if intended for use by the development staff of the application, as well as any developer who intends to make enhancements or modifications to the overall flow and interaction of the application.

1.3 Project Overview

This program will provide cost center managers with the ability to track numerous operations regarding assets as described in the Specific Requirements Document. (This will be referred as SRS-Ref from this point forward in this document)

When the use cases in this document are referred to, the notation DD-UC-Ref will be used.

As quoted from the Project Requirements document provided to the teams at the beginning of the project:

> "Each engineering team within HP has its own R&D cost center. Each cost center manager is responsible for managing the spending within their cost center against the approved quarterly budget. Each month the cost center manager meets with HP Finance in order to review the current spending in order to determine if spending is running ahead or behind budget. At the end of each quarter, the final spending numbers are compared against that quarter's budget. Spending over budget is just as bad as spending below budget.
>
> "Each cost center budget is composed of the following items; people expense (cost of employee salaries, benefits, etc.), external labor/consulting (short term contractors or consultants), travel, machinery and equipment (depreciation expense and equipment expense), and miscellaneous expenses (training, books, supplies).
>
> "The HP Customer Focused Testing (CFT) team runs three application solution development labs, located in Sophia-Antipolis, France, Marlboro, Massachusetts, and Colorado Springs, Colorado. These labs are staffed with 26 engineers who focus on Microsoft Exchange, Microsoft SQL Server, Oracle, and SAP application solutions.
>
> "As one would expect, supporting these efforts with the necessary equipment takes a considerable amount of money. In fact, within the CFT cost center, the largest budget item is machinery and equipment, which represents more than 50% of the overall CFT budget. Within the machinery and equipment budget item, 65% of the budget is allocated against depreciation expense, and 35% is allocated against equipment expense. Therefore, successful management of machinery and equipment expenses is critical to a successful budget."

2. System Requirements

2.1 Provide an import function for initial data

2.1.1 When "New/Import" is selected from the file menu, ImportWizard panel will display and it will contain 2 buttons and 4 text boxes for file path, and required expense budget limits. File browsing panel will display when "Browse" button is clicked. File browser will look like a typical windows file browser (tree view on left side and file list view on right side). When "Finish" button is clicked, all inputs in text box will be passed to "ImportOperation" class and it will check the existence of file and validation of each column in that file. After the check, all necessary classes will be created. (AssetCollection, AssetReport, Asset and etc).

SRS-REQ 3.1.1
SRS-REQ 3.1.2
SRS-UC 1.1
SRS-UC 1.2

2.2 Provide a view of long term budget impacts for capital acquisitions in the current quarter

2.2.1 When an excel data is imported and "Long Term Budget Impact" is selected from the view menu, "LongTermBudgetImpact" panel which contains a scrollable table list of current capital acquisition on the top and a bar graph showing depreciation run rate (each bar is current or future depreciation stream). The calculation of depreciation run rate will be handled by Monthly or Quarterly Projection class which adds up depreciation expense by assigned period. Analyzer Class will give an alert message dialog if any current or future depreciation run-rate has gone over its budget limit and will mark current capital acquisitions that are the most impacting cause of exceeding. (the most impacting capital acquisition are marked as red in the table view and they are computed in a way that, all the capital acquisition is sorted by hightest to lowest and it will mark each item from the top until no depreciation expense gets exceeded.)

SRS-REQ 3.2.1
SRS-REQ 3.2.2
SRS-REQ 3.2.3
SRS-UC 2.1
SRS-UC 2.2
SRS-UC 2.3

2.3 Provide for creation of persistent asset groups.

2.3.1 Allow grouping of assets using a filtering mechanism. The user must be able to define an asset group as a collection of criteria. This criteria can consist of literal comparisons (e.g. Asset Column A is less than 2000), relative comparisons (e.g. Asset Column A is greater than Column C), and pattern matching comparisons (e.g. Asset

Column B contains 'string'). The collection of criteria can then be run over a collection of assets in order to retrieve the desired group.

SRS-REQ 3.3.1
SRS-UC 3.1

2.4 Provide 'what-if' scenarios.

2.4.1 Allow creation of a scenario. A scenario must be independent of the asset report, so a scenario should be defined simply as a collection of asset obsolescences and acquisitions. Obsolescences may be referred to uniquely by the asset's id, while acquisitions should store the entire asset being added.

SRS-REQ 3.4.1
SRS-UC 4.1

2.4.2 Allow asset groups to be retired. The user will be able to run one of the asset groups defined in section 2.3 over an asset collection. Then, he will be able to check off assets which match the desired criteria, and obsolete those assets in one shot.

SRS-REQ3.4.2
SRS-UC 4.2

2.4.3 Allow scenarios to be saved for future use. As described in 2.3.1, scenarios will be independent of the current asset report. The format in which the scenario will be saved is described in the appendix.

SRS-REQ 3.4.3
SRS-UC 4.3.1
SRS-UC 4.3.2

2.5 Recommend an asset spending limit for the current quarter to maintain a level depreciation expense over the long term.

2.5.1 Assuming a three year depreciation term, calculate the amount that must be spent in order to keep next quarter's depreciation spending the same as the current quarter's depreciation spending.

SRS-REQ 3.5.1
SRS-UC 5.1

2.6 Show all assets which will be retired within the next two quarters

2.6.1 Calculate the total depreciation cost of any not yet retired asset for each month or quarter and display the values on the main depreciation graph. The user can select whether to display monthly or quarterly values in the main depreciation graphs menu.

SRS-REQ 3.6.1
SRS-UC 6.1

2.6.2 List any assets, with a depreciation cost larger than the value defined in the configuration, which will be retired in the next two quarters by soonest to retire.

SRS-REQ 3.6.2
SRS-UC 6.2

2.7 Recommend use of non-asset money to counteract short term increases in depreciation expenses.

2.7.1 If depreciation expenses in the current or near future are higher than the depreciation expenditure budget, the application will recommend program expense money that should be made available to offset the depreciation expenses by subtracting the depreciation expense budget from the total current depreciation cost.

SRS-REQ 3.7.1
SRS-UC 7.1

2.8 Display top 10% of high impact assets

2.8.1 Calculate the depreciation expense per quarter and list the assets with the highest depreciation costs that make up ten percent of the depreciation expenses. The top ten percent will be calculated by ordering the list of assets from largest to smallest monthly depreciation cost and popping the assets off of the list until the total of the assets popped off of the list is at least 10% of the total current depreciation.

SRS-REQ 3.8.1
SRS-UC 8.1

2.9 Recommend assets to retire early in order to balance the future depreciation stream.

2.9.1 After the user has imported the data from the excel files into the program, the user can click the button that calculate the retired asset, and the program would calculate the remaining assets and display the impact in form of currency and graphs.

SRS-REQ 3.9.1
SRS-UC 9.1

2.9.2 After the user has imported the data from the excel files into the program, the user can click the button that calculate the retired asset, and the program would calculate the remaining assets. The program also display the amount of money needed to retire these assets and the remaining amount of money in this quarter's budget.

SRS-REQ 3.9.2
SRS-UC 9.2

2.9.3 After the user has imported the data from the excel files into the program, the user can click the button that calculate the retired asset, and the program would calculate the remaining assets. The program also display the recommended assets to be retired in form of checkboxes and the amount of money needed to retired each of these assets. The user then can choose which assets to retired and click confirm.

SRS-REQ 3.9.3
SRS-UC 9.3

2.10 Allow the information for each view to be printed.

2.10.1 Each view will have either a print button or print option in the menu. Using this option will cause the view to open a print view and print the view.

2.10.2 The print functionality will be used to create a picture export for the graphs so that they may be used outside of the application.

SRS-REQ 3.10.1
SRS-UC 10.1
SRS-UC 10.2

2.11 Allow persistence between sessions.

2.11.1 Any open asset report and scenarios shall be serialized and saved automatically on program close and be unserialized and loaded when the program next starts. The serial objects will be saved in the application folder.

SRS-REQ 3.11.1
SRS-UC 11.1

2.11.2 The visibility and location of each view shall be serialized and saved when the program closes and unserialized and loaded when the program next starts. The serial objects will be saved in the application folder.

SRS-REQ 3.11.2
SRS-UC 11.2

2.11.3 Asset groups shall be automatically saved and maintained until the user chooses to delete them. The asset groups will be saved during the creating of the asset group before applying them to the scenario. The asset group objects will be serialized and saved in a Groups subfolder in the application folder.

SRS-REQ 3.11.3
SRS-UC 11.3

3. Design Specification

3.1 Design Principles/Guidelines

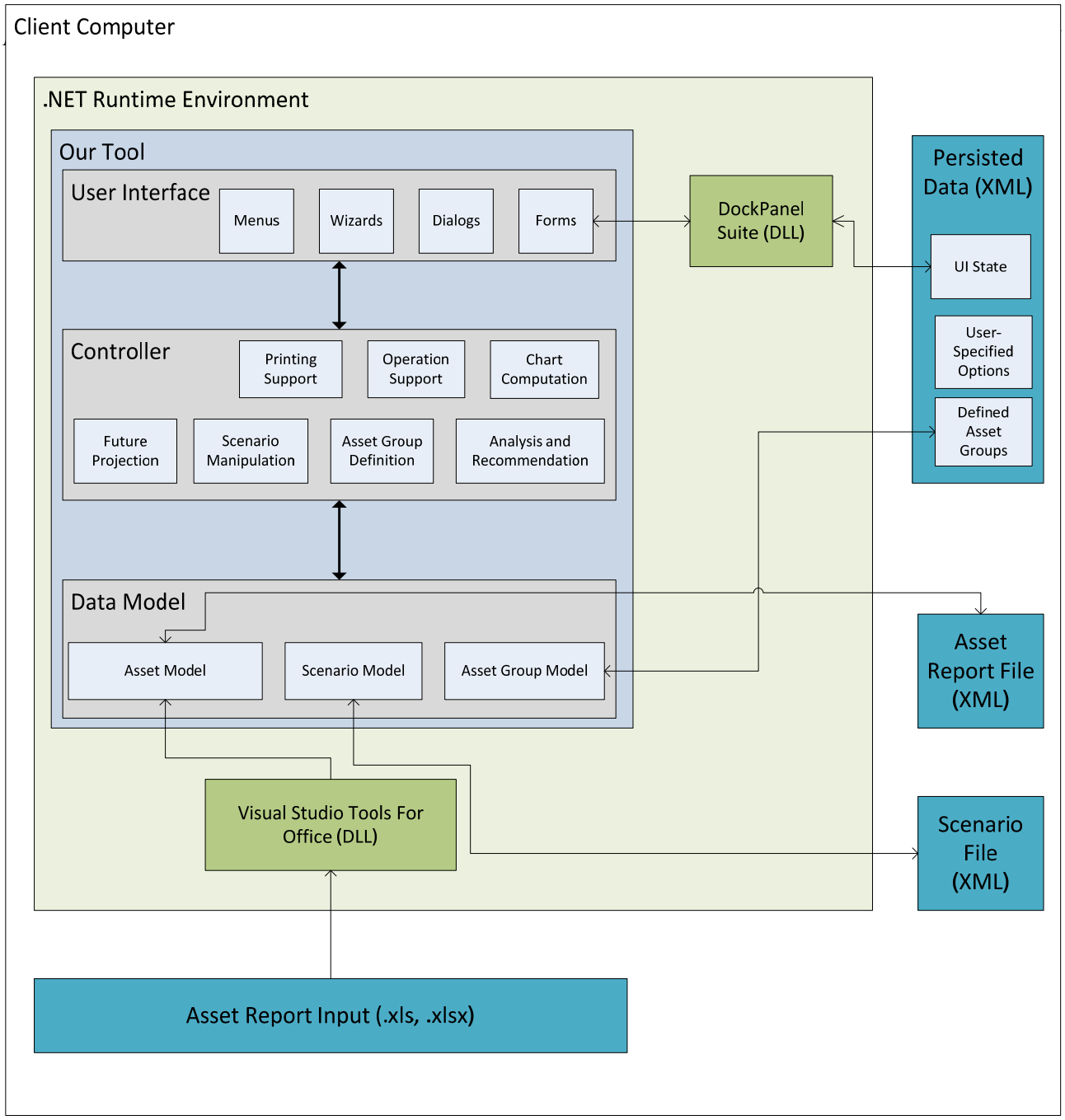
We have decided to take an object oriented approach to the design of our application to keep our design modular. This allows us to easily break up the implementation of the project easily amongst our group. We also look towards using existing libraries to accomplish our goals to minimize the amount of time spent by our group towards recreating efforts already put forth by other people.

3.2 Components

Our application follows the Model-View-Controller design pattern. As such, the software is divided into three distinct layers, and each of these is further divided into subcomponents. A basic overview of these components, as well as their interaction with the host computer, is shown in Figure 1.

3.2.1 Model Layer

The lowest layer of our software is the data model. This layer contains the objects which actually hold the data we are storing, manipulating, and viewing. It consists of three subcomponents.



3.2.1.1 The Asset Model

The most fundamental element of our program is the asset model. An asset is a single identifiable asset of the corporation – a single row on the asset report spreadsheet. The asset model is initially populated through the import spreadsheet process. It is written out in XML format for reuse within our application.

3.2.1.2 The Scenario Model

The asset report, as loaded from the spreadsheet, is used as a reference only and is immutable. To hypothetically add or remove items from the spreadsheet, a scenario must be created. A scenario is basically a collection of asset additions and removals. Stored in this way, the scenarios are not *applied* to the current loaded asset report, but not *dependent* on it. Scenarios are also saved in an XML format.

3.2.1.3 The Asset Group Model

Asset groups are a convenience added in order to construct scenarios in which all assets which match certain criteria are made obsolete. An asset group is essentially a collection of inclusion criteria such as 'Column Description contains "xyz"' or 'Net Value is less than 500'. Asset groups are saved as part of the application's metadata, in a manner transparent to the user.

3.2.2 Controller Layer

The bulk of the functionality of the application lies in the controller layer. It is here that actions are performed which make changes to the model layer. There are seven subsystems of the controller.

3.2.2.1 Operation Support

Probably the most basic component of the controller is the operation framework. This simply defines what an action is, and provides a standard interface through which actions may be created, invoked, and responded to.

3.2.2.2 Future Projection

Another fairly simple component, this module calculates projections for the future based on current data. Primarily, it will compute the depreciation stream for upcoming quarters for the current asset report as well as for any active scenarios.

3.2.2.3 Chart Computation

The chart computation component takes the data as computed by the future projection, and translates it into a format that can be drawn as a chart. Essentially this means translating a series of data points into lines to be drawn on the screen.

3.2.2.4 Scenario Manipulation

The scenario manipulation module is fairly self-explanatory; it allows manipulation of scenarios. Specifically, it provides the interface by which assets are added and removed to scenarios, and is responsible for propagating events which notify observers of these changes.

3.2.2.5 Asset Group Definition

The asset group definition is a mostly transparent interface between the view and the asset group model. It allows asset groups to be created based on input that was received from the user.

3.2.2.6 Analysis and Recommendation

This component handles the recommendation requirements of the software. It is responsible for evaluating current and future budget and expenses and providing reasonable recommendations to the user on how to best plan the budget.

3.2.2.7 Printing Support

For each view, it must be possible for the data from the view to be printed. The function of this component is to take the data that is backing the view, translate it into a printable format, and actually interact with the printer to get the printing job queued (through the .NET framework, of course).

3.2.3 View Layer

The final layer to discuss is the view or UI layer. The purpose of this layer is to expose the functionality of the controller and display the contents of the data model in a user-friendly and intuitive way. There are three main parts of the view layer.

3.2.3.1 Forms

Forms, also referred to in this document as Views, are the core component of the view layer. They provide the application shell and also are responsible for displaying the model's data to the user. Forms also invoke the controller through interactive user input or through use of toolbars and menus.

3.2.3.2 Dialogs

Dialogs are single-purpose forms which interact directly with the user. They are used to either provide information to the user (e.g. give information about an error) or to solicit input from the user (e.g. to customize application settings).

3.2.3.3 Wizards

Wizards are dialogs which walk the user through a process. They are usually related to an action, and provide ways for the user to alter the behavior of that action. Also, it allows the user to confirm the action before it happens.

3.3 Static Structure

3.3.1 Conventions

In the following diagrams, there are a number of conventions used which vary from standard UML. At times these are made to simplify the model, and at others it is due to UML not supporting certain features of the C# language.

3.3.1.1 Properties

C# has the concept of properties as a language element. A property can be thought of as an encapsulated field. To clients, it is addressed identically to a field, but behind the scenes access is controlled by `get` and `set` methods. A property is (typically) defined in the following way:

```
private type field;
public type Property {
     get { return field; }
     set { field = value; }
}
```

The property is then used by clients as if it were a field.

In this document, properties are represented as public attributes beginning with a capital letter (the backing private fields are not shown). Additionally, such attributes with the stereotype «ReadOnly» do not define the `set` portion of the property, and are typically transient (the values are computed on the fly, rather than stored in a field).

3.3.1.2 Events

Events are similar to properties, in that they are present in most languages but are first-class language elements in C#. As the name suggests, events are raised when something important happens that multiple unspecified clients may want to be aware of. The clients mentioned are called event handlers.

Explaining the usage of events is outside the scope of this document (See MSDN) but it is important to understand how events are diagramed. Operations with the stereotype «Event» are event publishers, and those with the stereotype «EventHandler» are subscribers. Events are shown along with the type of arguments they provide.

For example, the diagram element

would be realized as:

```
public delegate void DelName(object sender, ArgType
args);
public event DelName EventName;
```

Event handlers are only identified by the name of the event they subscribe to. The handler itself would be a method by any name which matches the signature of the delegate.

3.3.1.3 Singletons

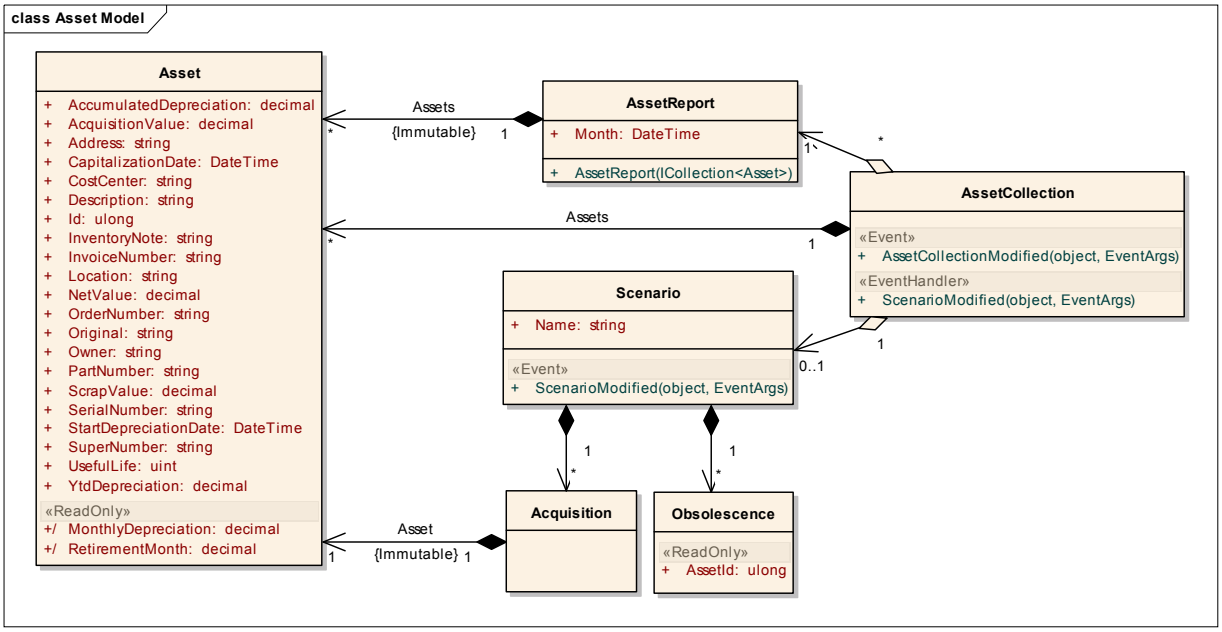
Classes with the stereotype «singleton» may be realized either as a singleton in the traditional sense, or a C# static class. A static class is a class which only permits static members, effectively creating a singleton. The two approaches are equivalent, so which is used does not matter.

3.3.1.4 Collections

For simplicity, operations which modify a aggregation are not shown – unless stated otherwise, it is assumed that clients have the ability to add and remove items of a collection.

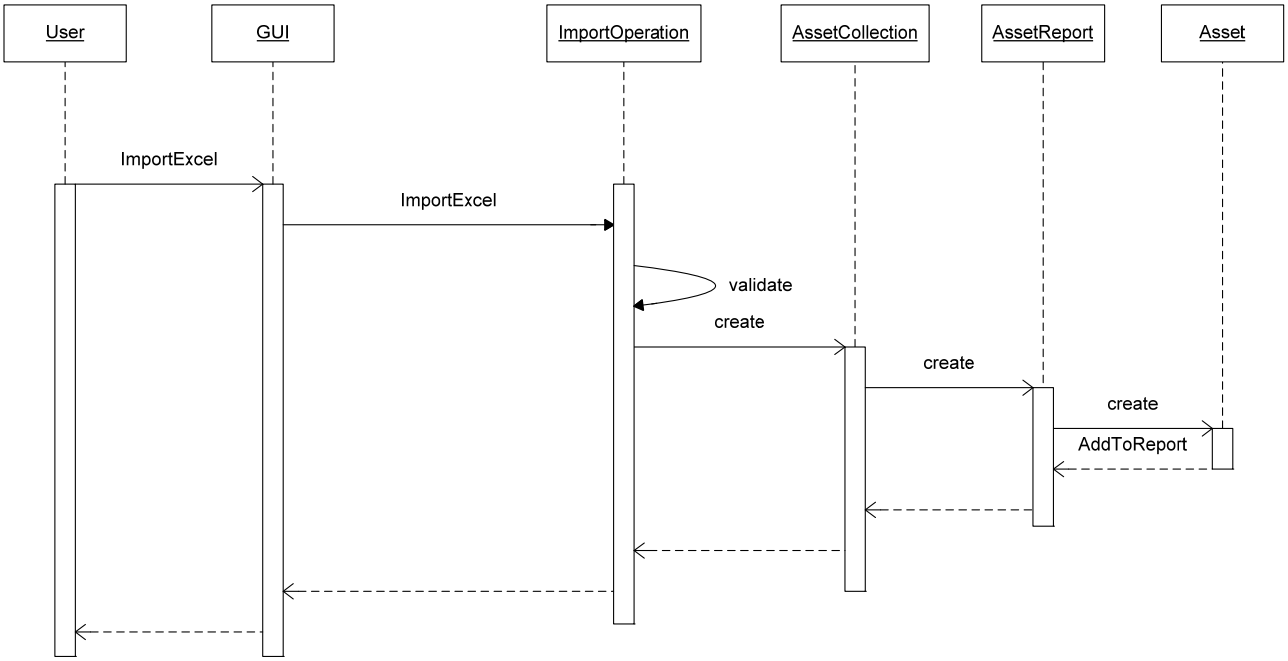
Asset Model

Type: **Package**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Class Model
Detail: *Last modified on 9/27/2007*



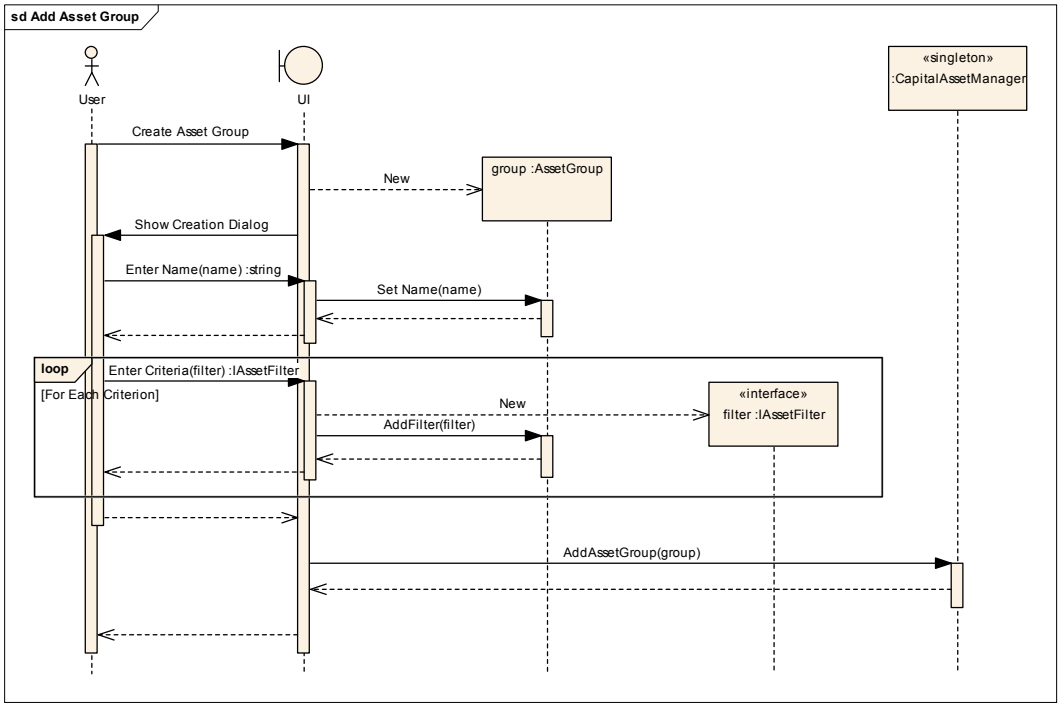
Asset Model - *(Logical diagram)*
Last Modified: 9/28/2007
Version: 1.0.

3.4 Object Interaction

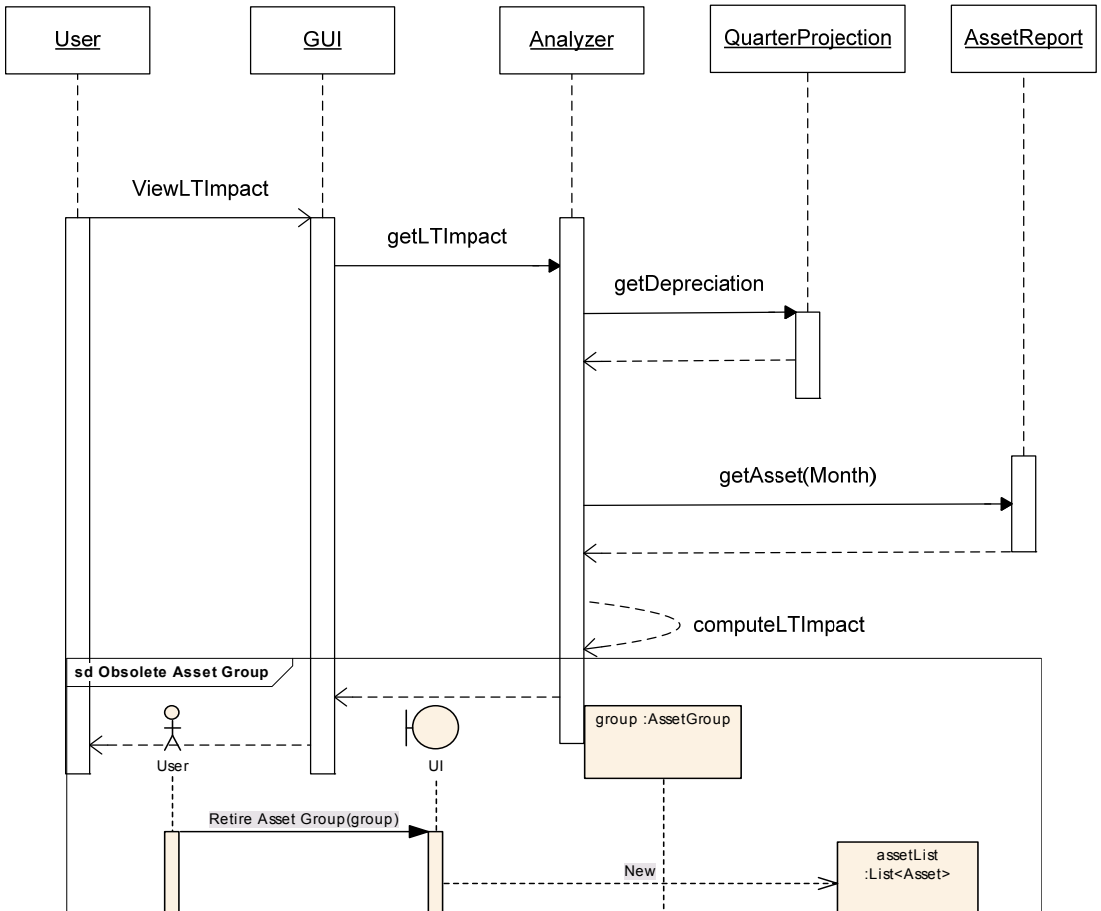


3.4.1 Data import

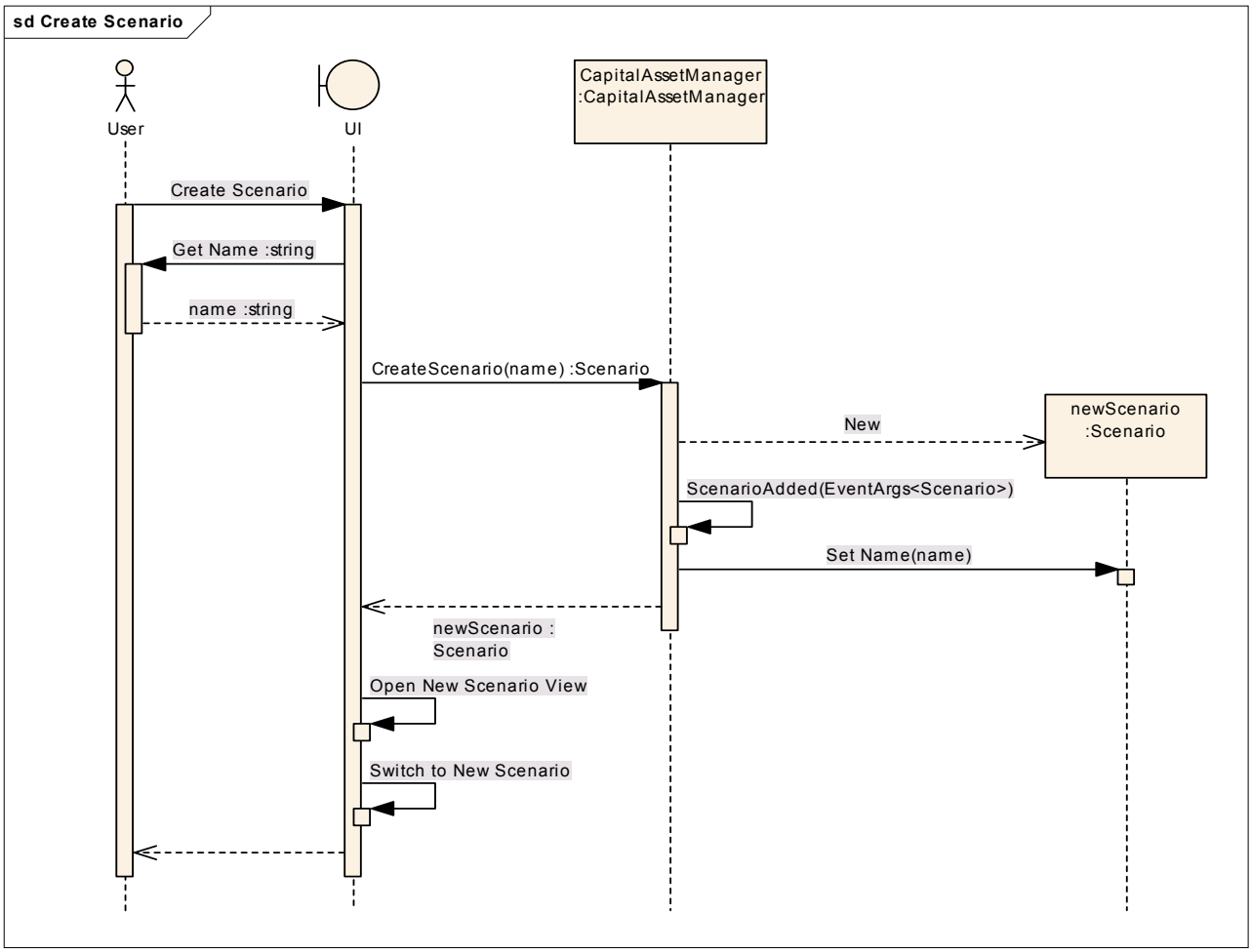
3.4.2 View impact of adding assets



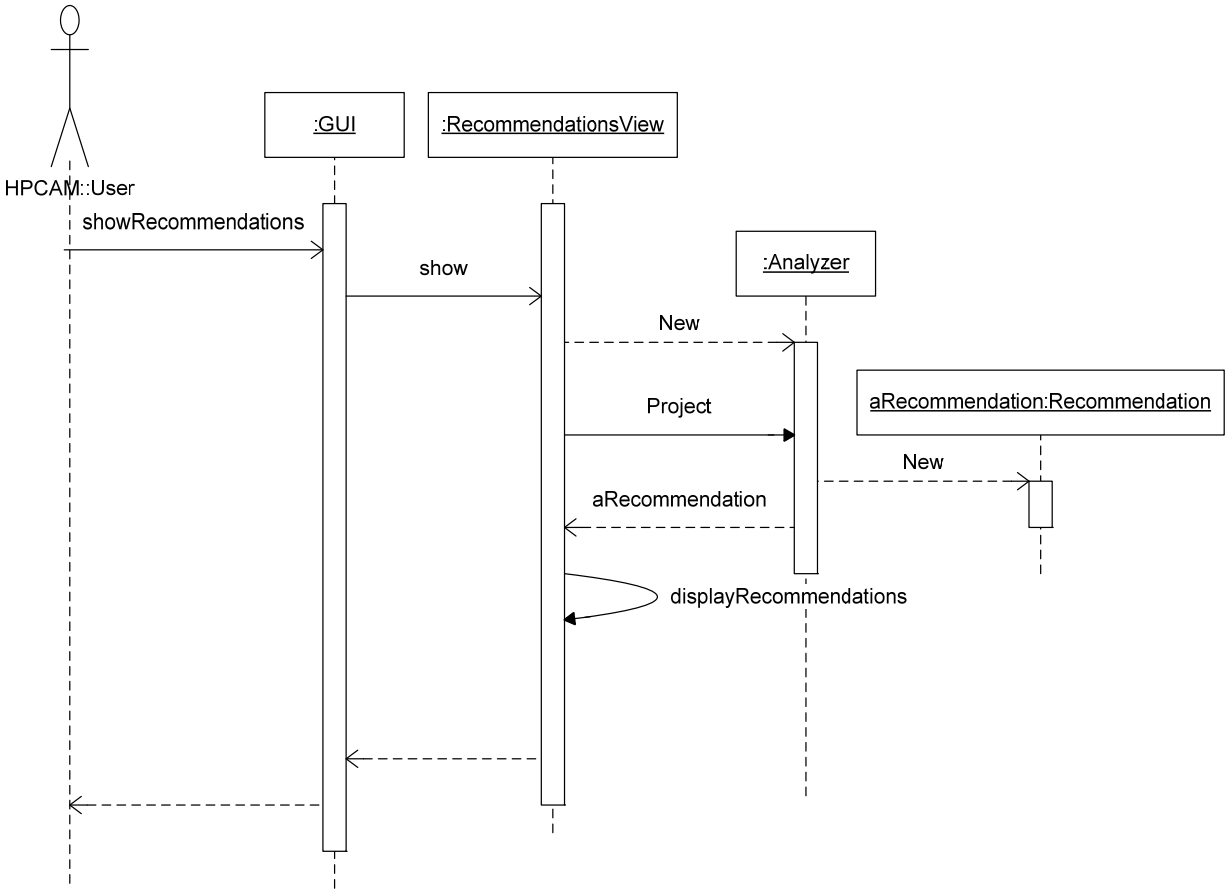
3.4.3 Asset group persistence



3.4.4 What-if scenarios



3.4.5 Recommended asset spending limit

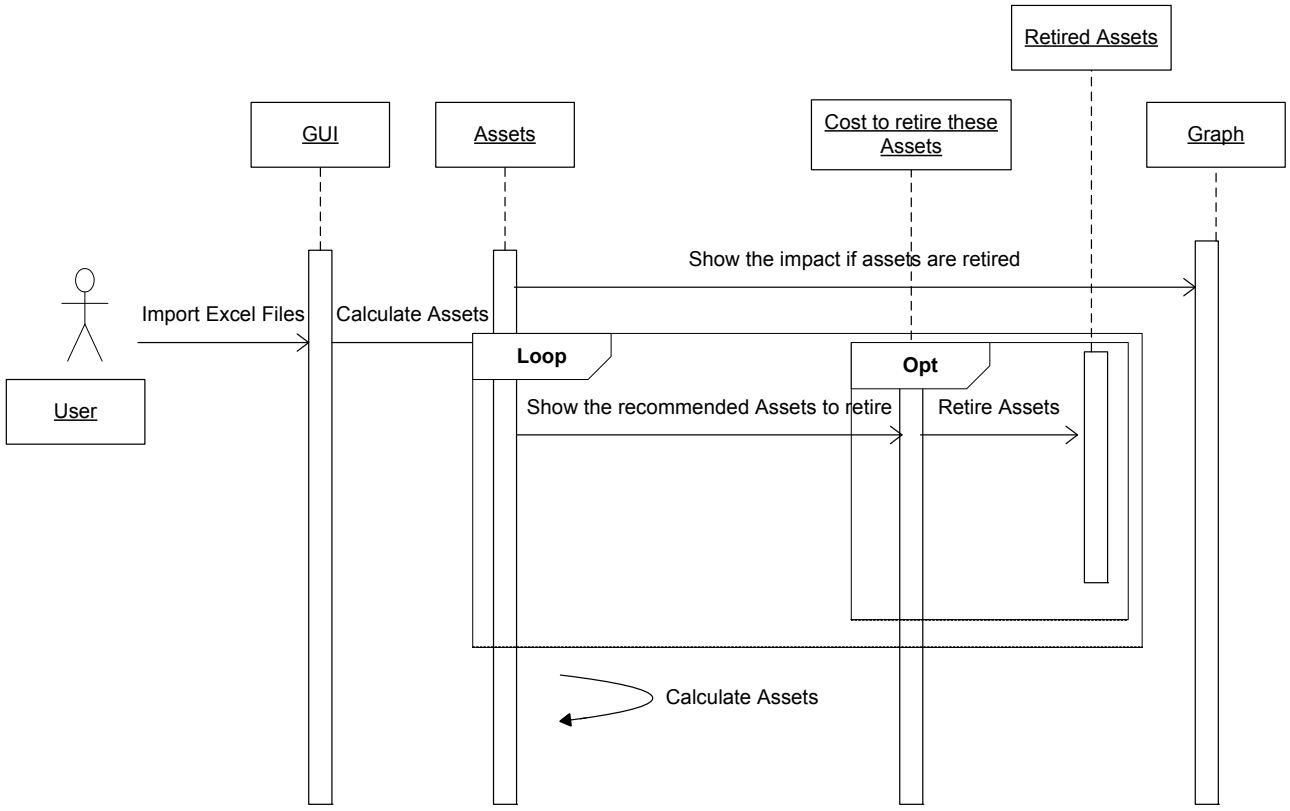


3.4.6 Soon to be retired assets

3.4.7 Recommend using program money to cover depreciation

3.4.8 Top 10% assets

3.4.9 Recommend early retirement of assets



3.4.10 Printing

3.4.11 Saving

4. Appendix

4.1 Depreciation Prediction Algorithm

Define a function PredictDepreciation, which gives the depreciation costs for all assets for a given month:

```
define PredictDepreciation(month) =
  let depreciationSum = 0;
  for each asset
    if asset is not yet retired in given month then
      depreciationSum = depreciationSum + ((value of asset) / (depreciation term of asset))
    end if
  end for
  return depreciationSum
```

4.2 Recommendations Algorithm

Define function Recommend, which given a month returns the recommended capital asset spending and the recommended non asset spending applied to depreciation.

```
define Recommend(month) =
  let assumedDepreciationPeriod = 36
  let capitalSpending, programSpending, depreciation be mappings of months to dollar amounts

  for i = each month from month before current month to given month
    set depreciation[i] = PredictDepreciation(i)
  end for

  for i = each month from current month to month after given month
    let budgetDiff = depreciation budget - depreciation[i]
```

```
    if budgetDiff < 0 then
      if -budgetDiff > program budget then
        signal error to user (not enough program budget to cover depreciation)
      else
        set programSpending[i] = -budgetDiff
        set capitalSpending[month before i] = 0
      end if
    else
      set programSpending[i] = 0
      set capitalSpending[month before i] = budgetDiff * assumedDepreciationPeriod
    end if

    // update future depreciation costs taking into account the spending we just calculated
    for j = each month from i to given month
      depreciation[j] = depreciation[j] + (budgetDiff * assumedDepreciationPeriod)
    end for
  end for
return capitalSpending[month], programSpending[month]
```

4.3 File Formats

Asset groups, asset reports, and scenarios will all be persisted at various points throughout the application's lifetime. All three types of files will be persisted as XML. The .NET framework has a built in mechanism for serializing and deserializing objects using XML. Thus, the actual format used is handled by the framework, and is irrelevant from a design perspective. It is, however, described somewhat at MSDN and elsewhere.

4.4 Graph Description

One of the principal uses of the application is the graphing utility. As such, it is important to provide as much functionality as possible within the graph. The graph will be based on the current AssetCollection, and will update whenever a change is made.

The first option for the graph is whether it is a bar or line chart. While bar charting may be preferred for discreet data sets such as this, there are times when it can be too cluttered. If viewing the graph in a small window at monthly granularity over a long period of time, it would be more beneficially to see a line chart. So, both options are allowed.

The principal data set is the depreciation stream, obviously. By default, the only bar (or line) which will be drawn is the monthly depreciation cost. The user will have the option of toggling on and off horizontal guidelines which represent the program, depreciation, and acquisition budgets.

The granularity of the graph may be specified as months or quarters. The starting point of the graph is always the current month (or quarter) due to a lack of previous data. The span of the graph is customizable, supporting at least two years into the future (going further would not be difficult and may be allowed, but it serves little purpose).

Finally, if the asset collection is based on a scenario, the user may select to show the basis asset report as well as the current scenario to visually compare the impact of acquisitions and retirements.

5. Glossary