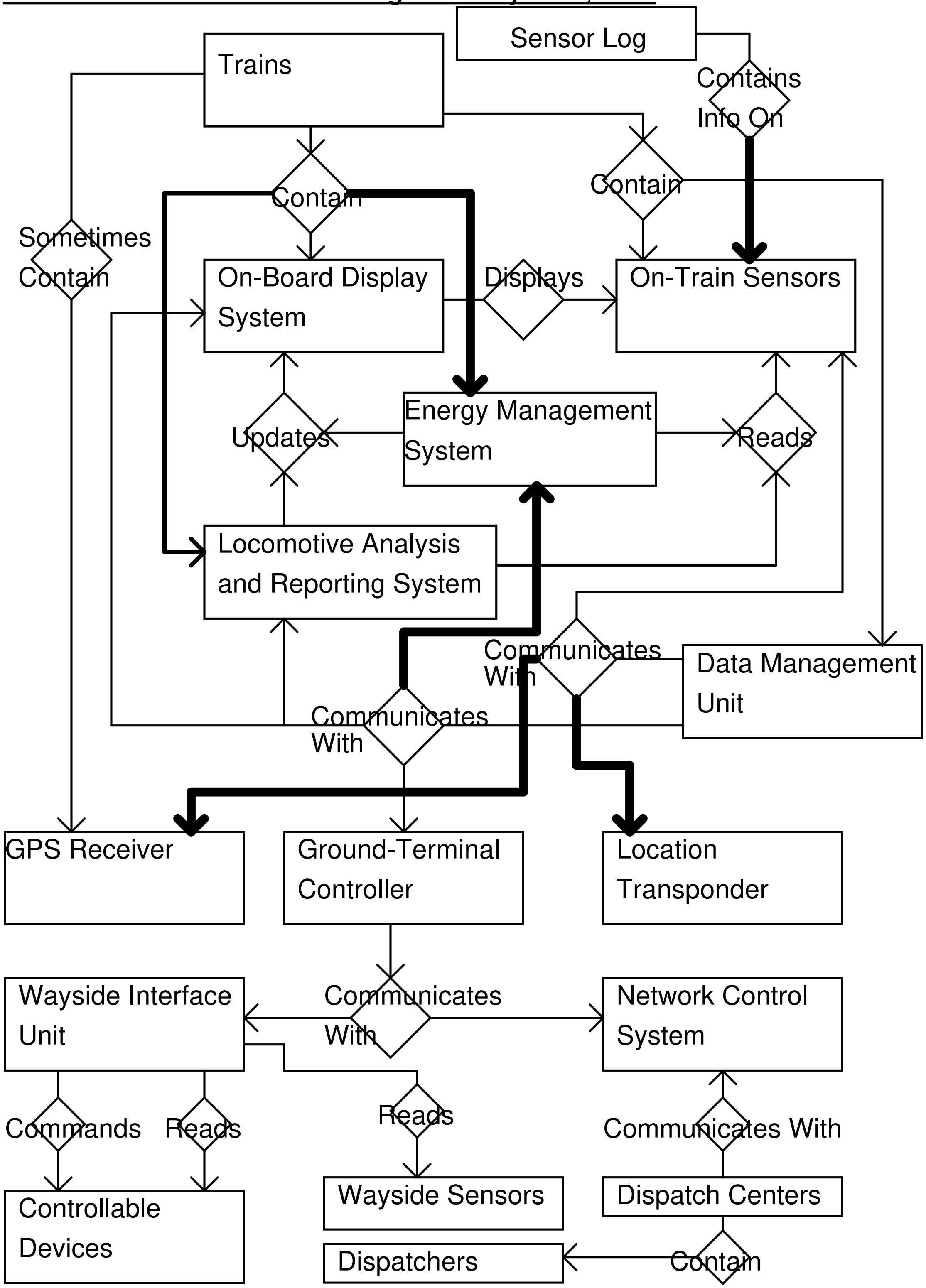
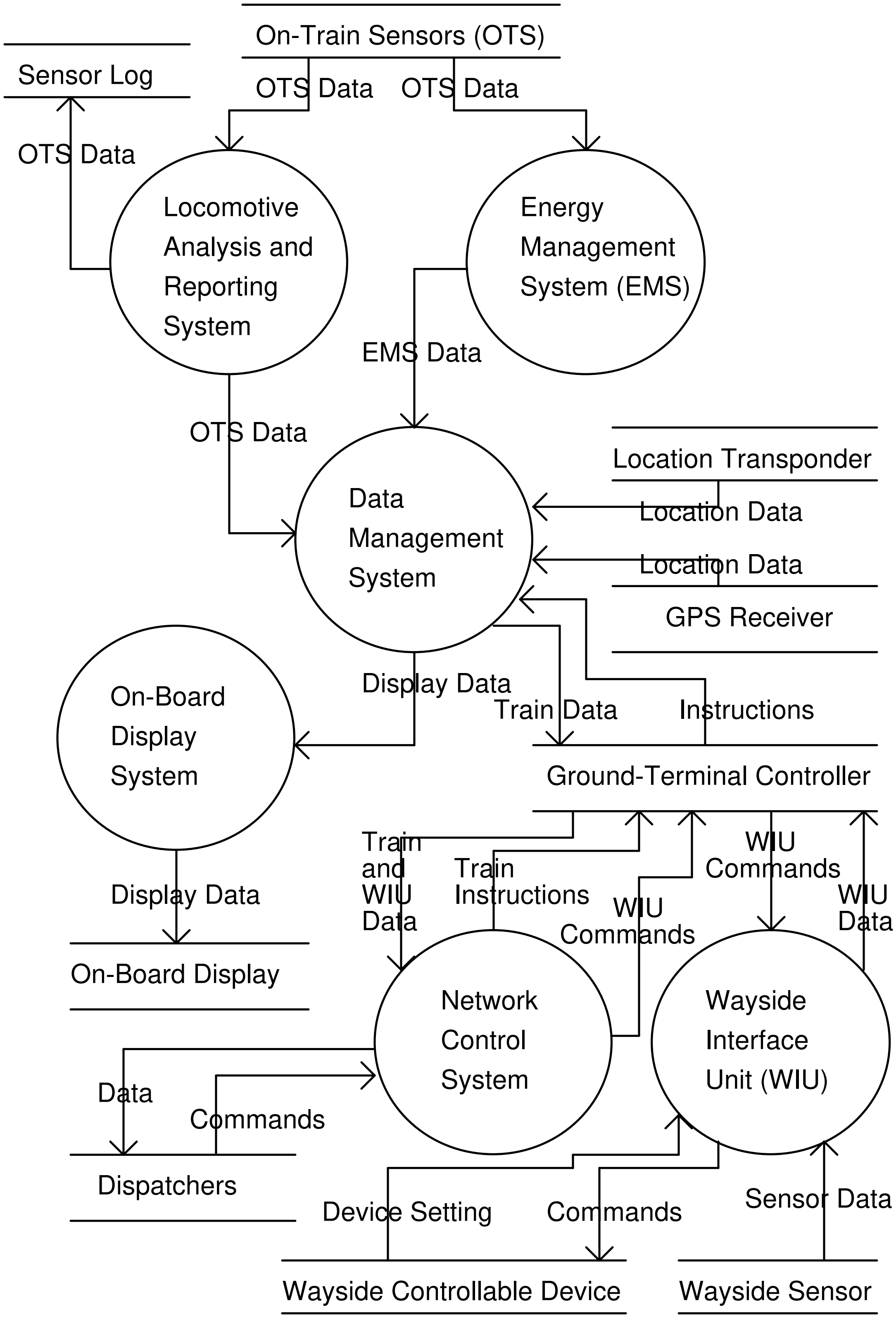
Problem 2-1: Ada Traffic Management System, ERD



Problem 2-1: Ada Traffic Management System, DFD



3-1. Paper on Ada Tasking

Acceptable Rating. The following information is required to achieve a rating of *acceptable*:

1. A discussion of the Ada tasking mechanism. How an Ada task is declared (defined as a class) and how instances of a task are created.

2. A discussion of the Ada rendezvous mechanism. This discussion should mention the *communicating sequential process* model and how it works. The idea that tasks go through states during the rendezvous is important.

3. A discussion of how interrupts are tied to task entry points. How an address is set to a task entry point should be included.

Reference: Grady Booch, *Software Engineering with Ada, 2nd Edition*, Benjamin-Cummings, 1987, Chapter 16.

Exceptional Rating. All information expected for the Acceptable Rating must be included. Additionally, one or both of the following items must be included:

1. The state model for an Ada task. Upon encountering an *accept* statement or a task entry invocation, a given task moves from the *running* state to the *blocked* state. The task stays in the *blocked* state until the rendezvous is achieved (the other task has issued a task entry invocation or encountered the corresponding *accept* statement. The task executing the *accept* statement moves from the *blocked* state to the *ready* state, at which point the Ada runtime system's task scheduler determines if (and when) memory is available to restart the task. When the task is restarted, it is moved from the *ready* state to the *running* state, and the *accept* block is executed. Upon completion of the *accept* block, the invoking task moves from the *blocked* state to the *ready* state, and the Ada runtime system determines if the task can then load. When loaded, the invoking task is moved to the *running* state, and both tasks continue on their way.

2. The fact that tasks can have attributes, such as 'TERMINATED. Just like any object, task objects in Ada have attributes which may be queried by other tasks in the Ada system.

Reference: Grady Booch, *Software Engineering with Ada, 2nd Edition*, Benjamin-Cummings, 1987, Chapter 16.

3-2. Class Book

For this problem, I was looking for a class definition of a book that is complete enough to distinguish the book class from other classes. I was also looking for a sufficient level of abstraction to make this distinction. A reasonable C++ declaration may be:

```
class book {
  char **toc;                // contents string array
  char **index;              // index string array
  char **chapters;           // array of chapters
  char **appendices;         // array of appendices
  char **front_cover;
  char **back_cover;
  char **pictures;
public:
  book();                    // constructor
  ~book();                   // destructor
  int open();                // open a book
  void close();              // close a book
  int read_chapter();        // read a chapter of a book
  int scan_index(char *);    // scan index of a book
  void goto_page(int);       // goto page in a book
};
```

Midterm 1. Data Dictionary

Price : amount spent for something
value range: 0.00 to large positive value (fixed point, 2 digits after decimal)
units: dollar

Seat : item whose use is purchased by the customer

- Single-ticket price category based on location (front row, middle, rear, or balcony)
- Full Subscription Series ticket price category based on location and nature of customer (adult, student)
- Mini Subscription Series ticket price category based on location and nature of customer

Customer : a person who purchases the use of a seat

Administrator/Sponsor : the person using the Rev-Pro System

- Source of all data fed into the system
- Sink of all data produced by the system

Location : the position of a seat
value range: enumeration (front row, middle, rear, balcony)
units: none

Ticket : a printed form given to a customer when he purchases the use of a seat

- Indicates the location of the seat whose use is purchased
- Indicates the amount of money involved in the purchase (Price)

Ticket Price : the amount of money involved in the purchase of a Ticket
subtype of Price

Single Ticket : a ticket for a single seat during a non-series concert
subtype of Ticket

Series Ticket : a ticket for a single seat during a series concert
subtype of Ticket

Full Subscription : a Series Ticket which is priced for a full series concert
subtype of Series Ticket

Mini Subscription : a Series Ticket which is priced for a part of a series concert
subtype of Series Ticket

Demand : a requirement by the Customer for a certain kind of Ticket (Single, Series, etc.)
value range: 0 on up
units: number of tickets

Revenue : an amount of income
subtype of Price

Price Category : an association between the price of a ticket, its location, and its customer
aggregate value: based on Price, Location, Customer, and Series Ticket

Pricing Level : same as Price Category

Discount Level : the percentage by which the Price of a subclass of Ticket is reduced based on the Price Category

Discount : the particular percentage by which the Price of a particular Ticket is reduced based on the Price Category

Pricing Grid : a 2-D table of ticket prices at each discount level versus price category

- In ticket revenue estimation, a level of demand is estimated for each cell
- Used to determine estimated ticket revenue (subtype of Price)

Estimated Demand per Cell : the estimated demand by the customer for a cell in the Pricing Grid
value range: 0 on up
units: number of seats

Cost Amount : the cost of a particular item associated with a concert
subtype of Price

Total Cost : aggregate cost of an event
subtype of Price

- Computed as a sum of a number of given cost amounts

Non-Ticket Revenues : aggregate income from outside sources, such as grants or gifts
subtype of Price

Estimated Revenue from Ticket Sales : estimated aggregate income from sales of tickets
subtype of Price

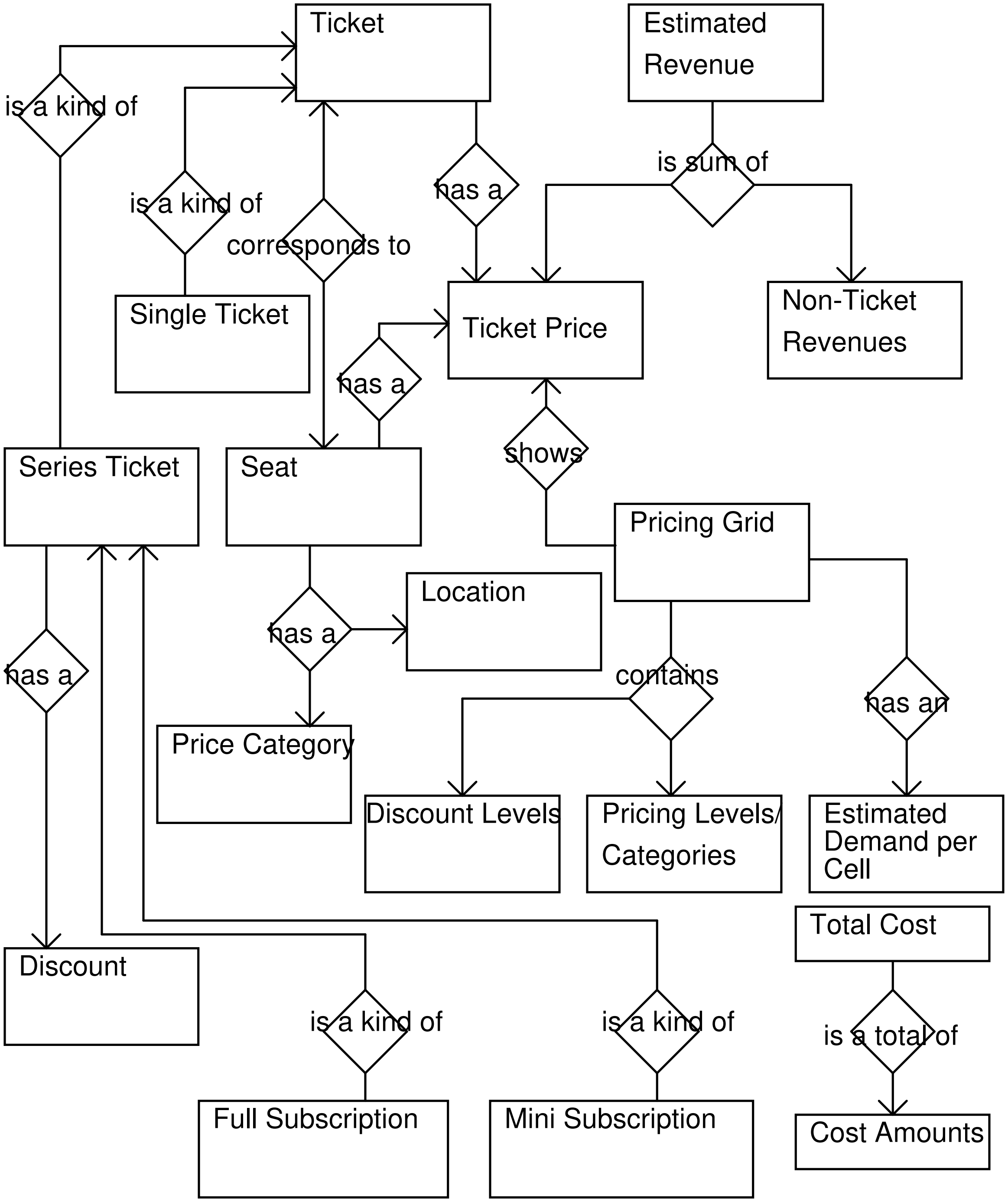
- Computed as the sum of products for each cell in the Pricing Grid; associated with each cell is the product of the estimated demand and the price of the cell, and these are the products which are summed

Estimated Revenue : total revenue estimated
subtype of Price

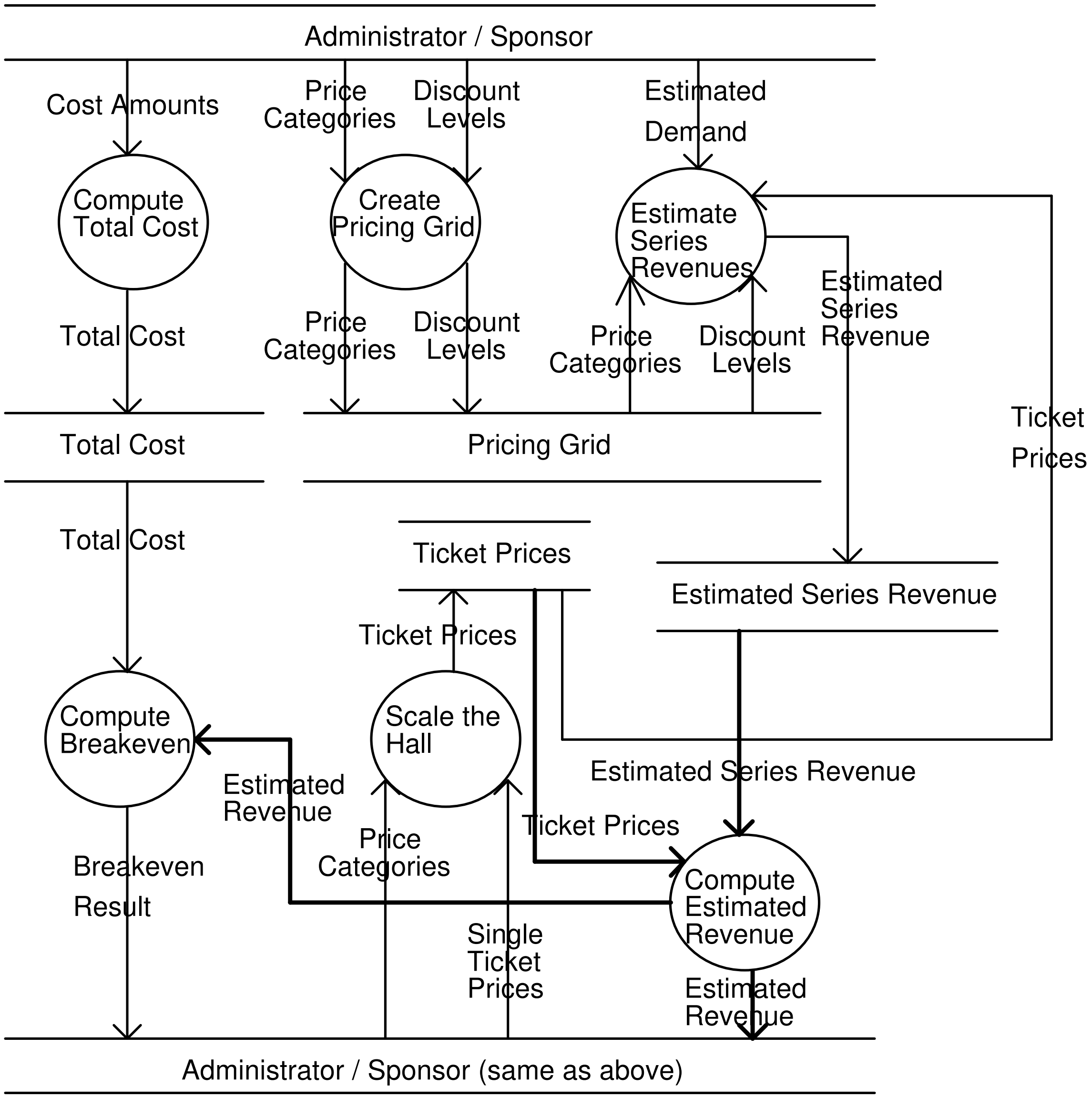
- Sum of Estimated Revenue from Ticket Sales and Non-Ticket Revenues

Breakeven Point : the situation in which Total Cost = Estimated Revenue
dimensionless

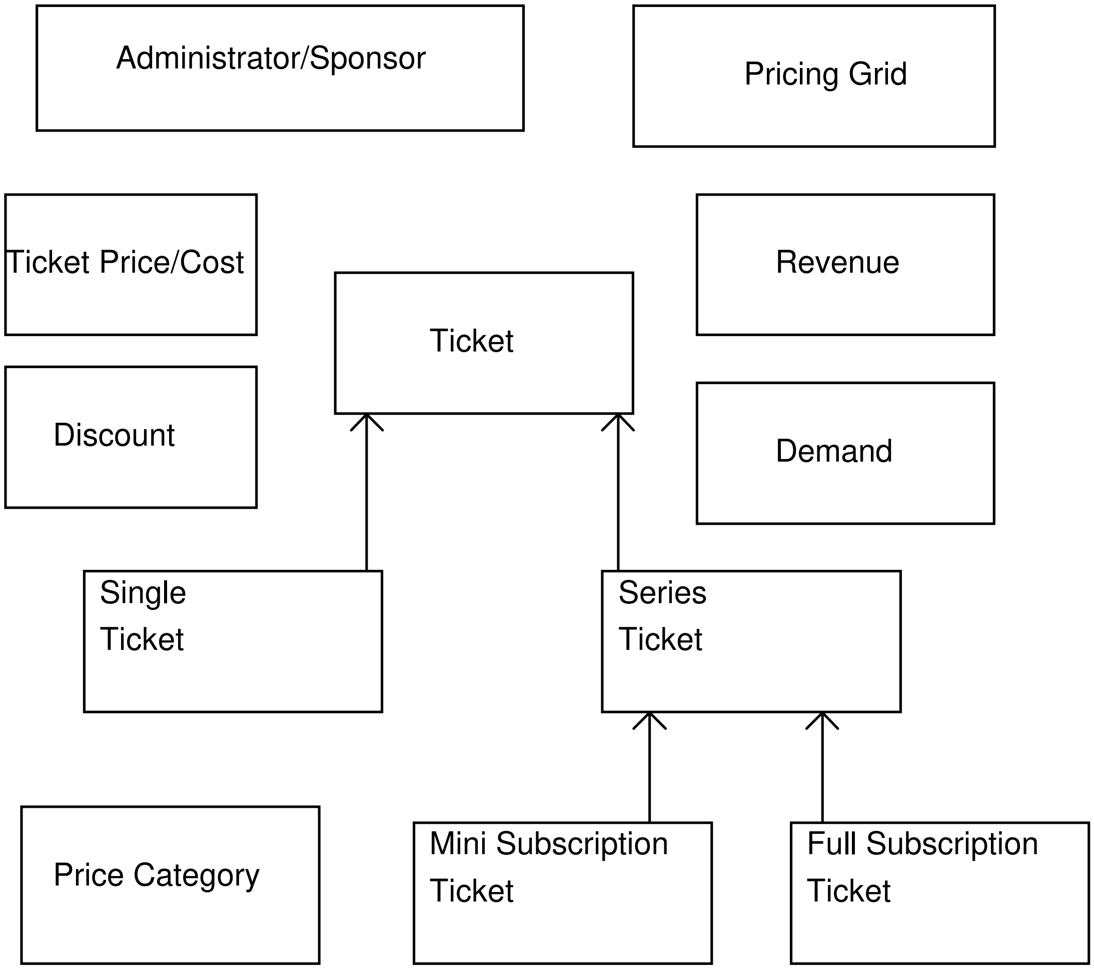
Midterm 2. Entity Relationship Diagram



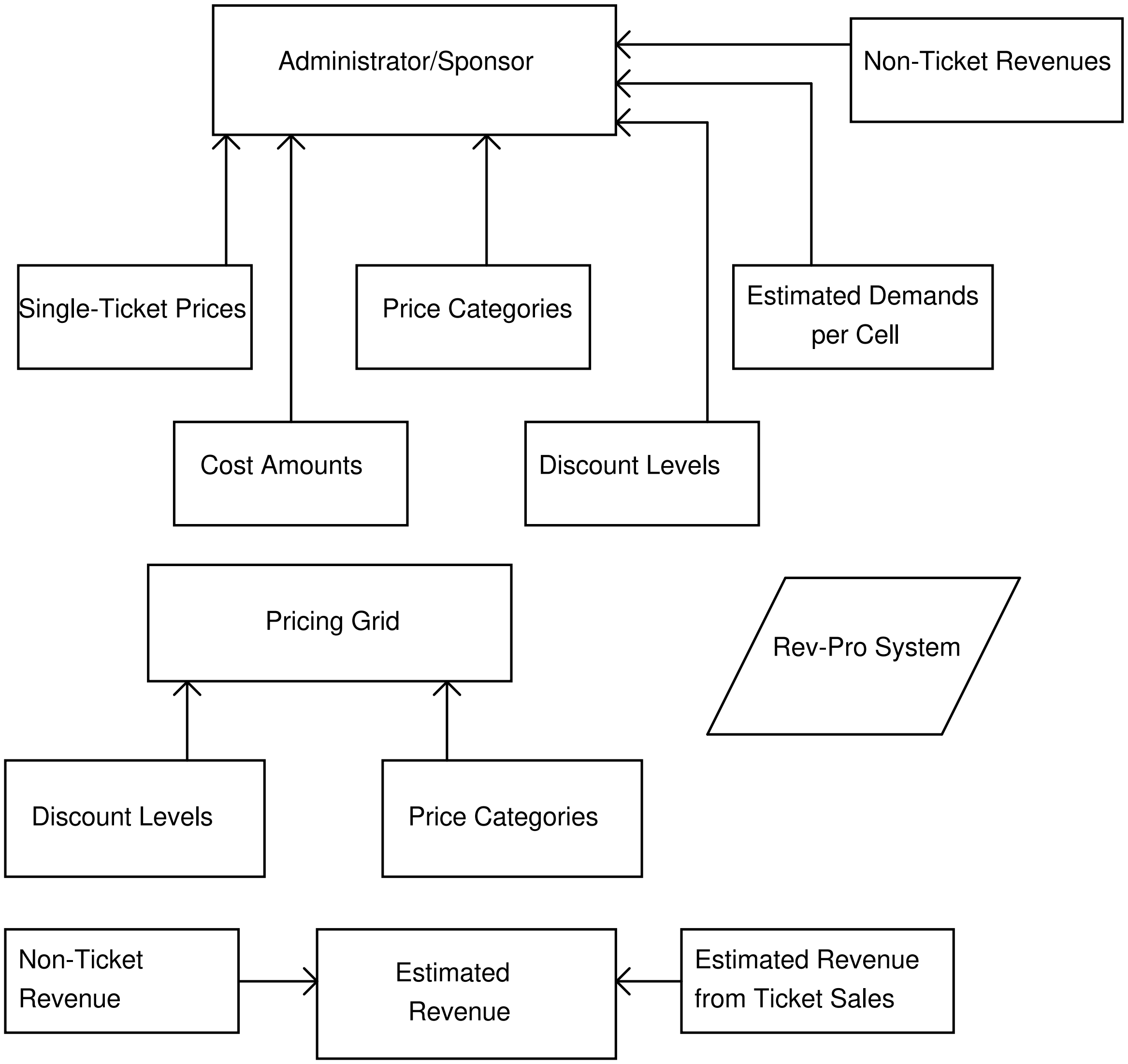
Midterm 3. Data Flow Diagram



Midterm 4. Class Hierarchy



Midterm 5. Object Structure



4-1. Ada Traffic Management System Classes

A wide variety of solutions would be acceptable for this problem, but it is interesting to note the book's solution in particular. Reading Chapter 12 of the book is a very good way to learn about an object-oriented design technique. The book's top-level class selection is as follows:

A. Train Class

Attributes: contains on-board display system, energy management system, locomotive analysis and reporting system, on-train sensors, data management unit, and (sometimes) GPS receiver [see my ERD from problem 2-1]

Behaviors: move on tracks (so the current location is of interest)

B. Track Class

Attributes: contains wayside devices (wayside interface unit, controllable devices, and wayside sensors)

Behaviors: report train location and status information

C. Plan Class

Attributes: contains schedules, orders, clearances, authority, and crew assignments

Behaviors: is read and updated

Lower-level classes suggested by the book are:

A. Message Class
B. Train-plan Class
C. Display Class
D. Sensor Class

The book goes into a little detail, but not all detail, on the attributes and behaviors of these classes.

4-2. Method Invocation

An acceptable paper must include the following information:

1. A discussion of static method invocation, including issues on resolving unresolved references to data and functions. Linking with object libraries should be addressed.

2. A discussion of dynamic method invocation, including issues on using pointer tables to functions (VPTR tables in the C++ vernacular, altho mention of C++ VPTRs is not required). How these pointer tables are addressed at runtime to locate the desired functions.

An exceptional paper may include any of the following additional topics:

1. Linking in a static sense to shared libraries. How these libraries do not need to be linked in to the actual executable but are memory-resident and accessed at runtime.

2. Very intimate details on the VPTR mechanism in C++. I'm afraid it has to be my call as to if the details are adequate to be called "very intimate."

5-1. Paper on Classification

An acceptable paper must include some discussion of the following topics:

1. The Importance of Proper Classification - includes a statement of the incremental and iterative nature of classification
2. Identifying Classes and Objects - includes statements about three general approaches to classification: classical categorization, conceptual clustering, and prototype theory
3. Key Abstractions and Mechanisms - includes statements of how key abstractions are identified (this must include statements about two processes: discovery and invention) and how mechanisms are determined

An exceptional paper is one which goes beyond the book, bringing out concepts from other sources (*not* including my course notes).

6-1. Ada Specification for Text File Objects

Note: This solution is not complete, but it gives a fair idea of the concept.

```
with TEXT_IO; -- Ada system support for files
package File is

  type OBJECT is limited private;
    -- defer the details of the attributes (member data)

  type OPEN_MODE is (READ, WRITE, APPEND);
    -- the modes by which a file may be opened

  procedure Open (File_ID : in out OBJECT;  -- handle on the file
                  Name    : in     STRING;  -- name of file
                  Mode    : in     OPEN_MODE);

  procedure Close (File_ID : in out OBJECT);

  function Exists (Name : in STRING) return BOOLEAN;
    -- True if named file exists

  procedure Delete (Name : in STRING);

  procedure Get (File_ID : in out OBJECT;
                 Item    :    out CHARACTER);
  procedure Get (File_ID : in out OBJECT;
                 Item    :    out STRING;
                 Index   :    out NATURAL);  -- index of last char in Item
  procedure Get_Line (File_ID : in out OBJECT;
                      Item    :    out STRING;
                      Index   :    out NATURAL);

  procedure Put (File_ID : in out OBJECT;
                 Item    : in     CHARACTER);
  procedure Put (File_ID : in out OBJECT;
                 Item    : in     STRING);
  procedure Put_Line (File_ID : in out OBJECT;
                      Item    : in     STRING);

  procedure New_Line (File_ID : in out OBJECT);

  procedure New_Page (File_ID : in out OBJECT);

private -- hidden from the user of this package

  type MONTH is (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC);
  subtype DAY    is NATURAL range 1..31;
  subtype HOUR   is NATURAL range 0..23;
  subtype MINUTE is NATURAL range 0..59;

  type OBJECT is record
      ID          : TEXT_IO.FILE_TYPE;     -- used by system to identify a file
      Name        : STRING(1..100);        -- storage for name of file
      Name_Length : NATURAL range 1..100;  -- number of chars in file name
      Size        : NATURAL;               -- 0 on up, so hope this is enough
      Mod_Year    : NATURAL;               -- date and time of file's last
      Mod_Month   : MONTH;                 --   modification
      Mod_Day     : DAY;
      Mod_Hour    : HOUR;
      Mod_Minute  : MINUTE;
  end record;

end File;
```

6-2. C++ Class Specification for Text File Objects

Note: This solution is not complete, but it gives a fair idea of the concept.

```
#include <stdio.h>

class File {
  FILE *ID;
  char *Name;
  int Size;
  int Mod_Year;     // positive
  int Mod_Month;    // 1-12
  int Mod_Day;      // 1-31
  int Mod_Hour;     // 0-23
  int Mod_Minute;   // 0-59
public:
  int Open (char *FName, char *Mode);  // mode is "r", "w", "a"
  void Close(void);
  static int Exists(char *FName);
  static void Delete(char *FName);
  char Get(void);
  void Get(char *Item);
  void Get_Line(char *Item);
  void Put(char Item);
  void Put(char *Item);
  void Put_Line(char *Item);
  void New_Line(void);
  void New_Page(void);
};
```