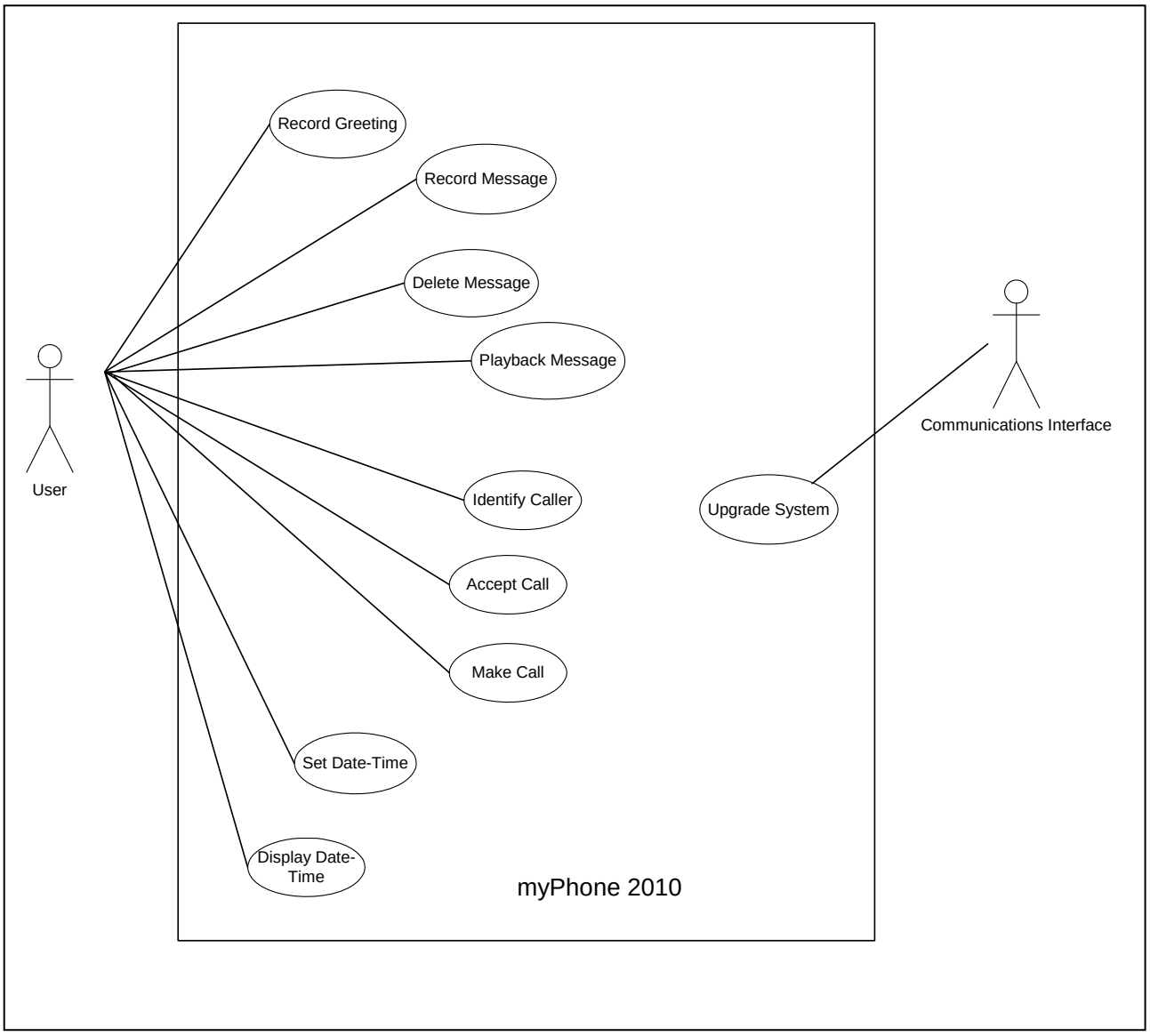
Use Case Diagram

The Use Case diagram gives us several pieces of information. First the major pieces of functionality…it answers the question: What does the user want to do….from the outside. Second, it gives us a brief description of what each such feature is or does. Third, it identifies, at an early stage, possible problems that we must address during the design phase.

Graphical View



Textual Description

Record Greeting

The user can record a greeting that will be played when a call is received.

Exceptions – out of memory, power off.

Record Message

An incoming call has been received, the outgoing greeting has been played, and the caller has begun speaking. The message will be tagged with the current time, the caller's identification, and stored in the message memory.

Exceptions – out of memory, power off.

Playback Message

The user selects a recorded message, presses a play button. The message is played until the user selects stop or deletes the message.

Exceptions – out of memory

Delete Message

The user selects a recorded message, presses a delete button. The message is deleted.

Exceptions – message not found

Identify Caller

The caller's name and telephone number are identified and made available for storage.

Exceptions – caller id information not found

Accept Call

An incoming call has been received, the user accepts the call by answering and then speaking.

Exceptions – none

Make Call

The user initiates an outgoing call by dialing the desired number and pressing the call button.

Exceptions – incorrect number, line busy, call refused

Set DateTime

The user may update the current date and time through a series of button presses.

Exceptions – incorrect format, invalid selection

Display DateTime

The display of date and time is the default case for the display, however, the user may choose to pre-empt another displayed message to display current date and time.

Exceptions – data lost

Upgrade System

The system can be upgraded by first checking for available upgrades, selectively downloading then installing the chosen upgrades

Exceptions – connection loss during access, partial download

Functional Decomposition

The use cases have given us an outside view of the system that we are designing. We work with those and our customer to put together a Requirements Specification which formally captures the system requirements. We then quantify those requirements in a Design Specification.

The use cases, Requirements, and Design Specifications are independent of the implementation of the system, including any choice of language or hardware. At this stage in the development, we move inside the system and begin to think about the implementation that will satisfy the specified requirements.

We begin by trying to identify the major functional blocks that comprise or give rise to the functionality or behaviour of the system. We repeat the hierarchical decomposition until we are satisfied that the level of detail allows us to move further into the specific implementations details of the design such as the microprocessor, the implementation language, etc.

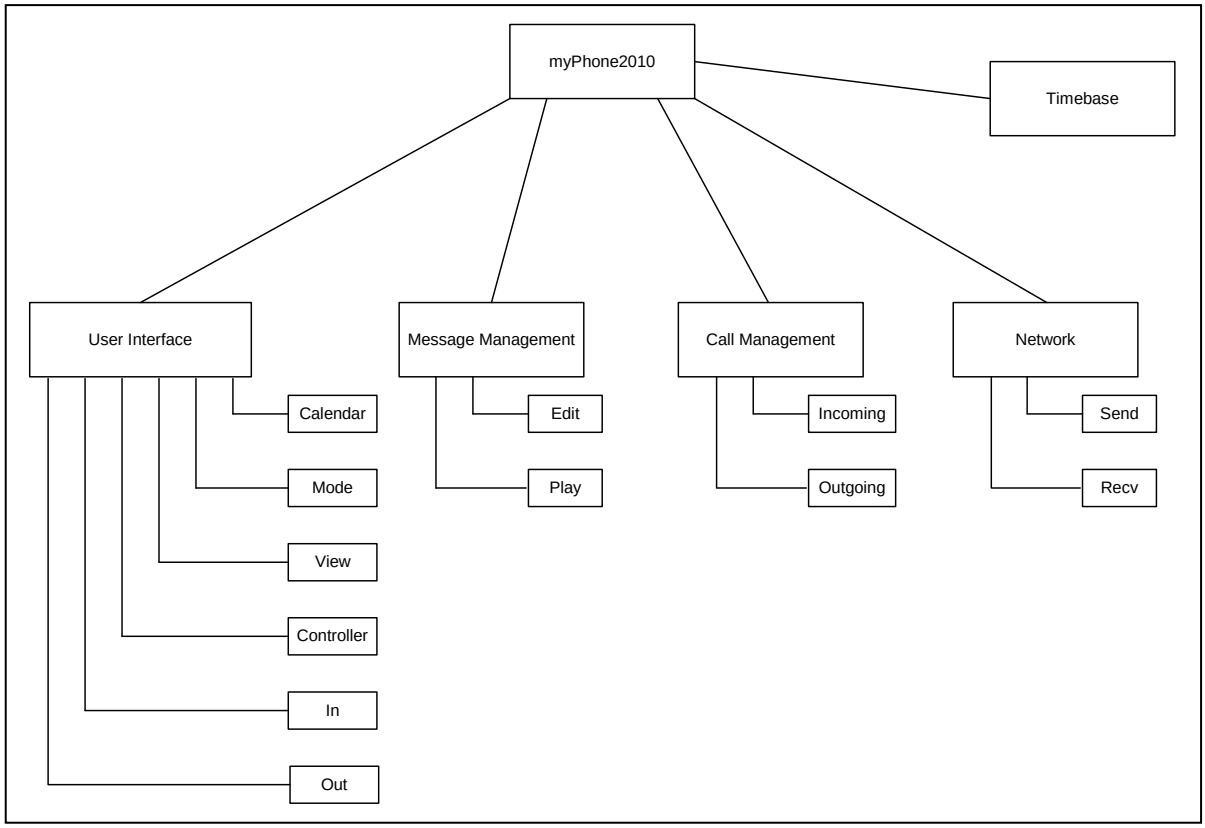
The current design is utilizing an approach from Smalltalk. This is called the Model, View, Controller (MVC) paradigm. The entity that is implementing a piece of functionality is called the Model. A model can have many associated view-controller pairs. The View provides an interface to the Model and the Controller provides the input-output mechanics.

When the user (which can be another piece of software) of the module sends an input into the module, that input is brought in by the controller and sent to the model which handles it. If that input necessitates a change in the output of the module, that information is sent to the view for updating.

Consider a simple case of a database filled with numbers. This is the Model. Now add the Views and Controllers. One view of the numbers may be as a pie chart, another as a histogram, a third as a line graph, and a fourth as an Excel spreadsheet. Whenever any of the data changes, for any reason, all of the views are updated to reflect hose changes.

Now for the cell phone. The functional decomposition follows in the next diagram.

We identify four top-level functions: User Interface, Memory Management, Call Management, and the Network. These are further decomposed into the secondary level functions shown.



CRC Cards

The CRC cards are the next tool that we'll use. We use these to begin to map the functional blocks onto sets of implementing modules or objects. These will ultimately lead to our classes for an object-centered design. Each card will provide two major pieces of information: what are the responsibilities of the module and what other modules will this one have to work with to exchange information to get its job done.

We don't include the hardware, but will include drivers for or interfaces to that hardware if necessary.

The diagram that follows gives one CRC card for the system. That for the User Interface.

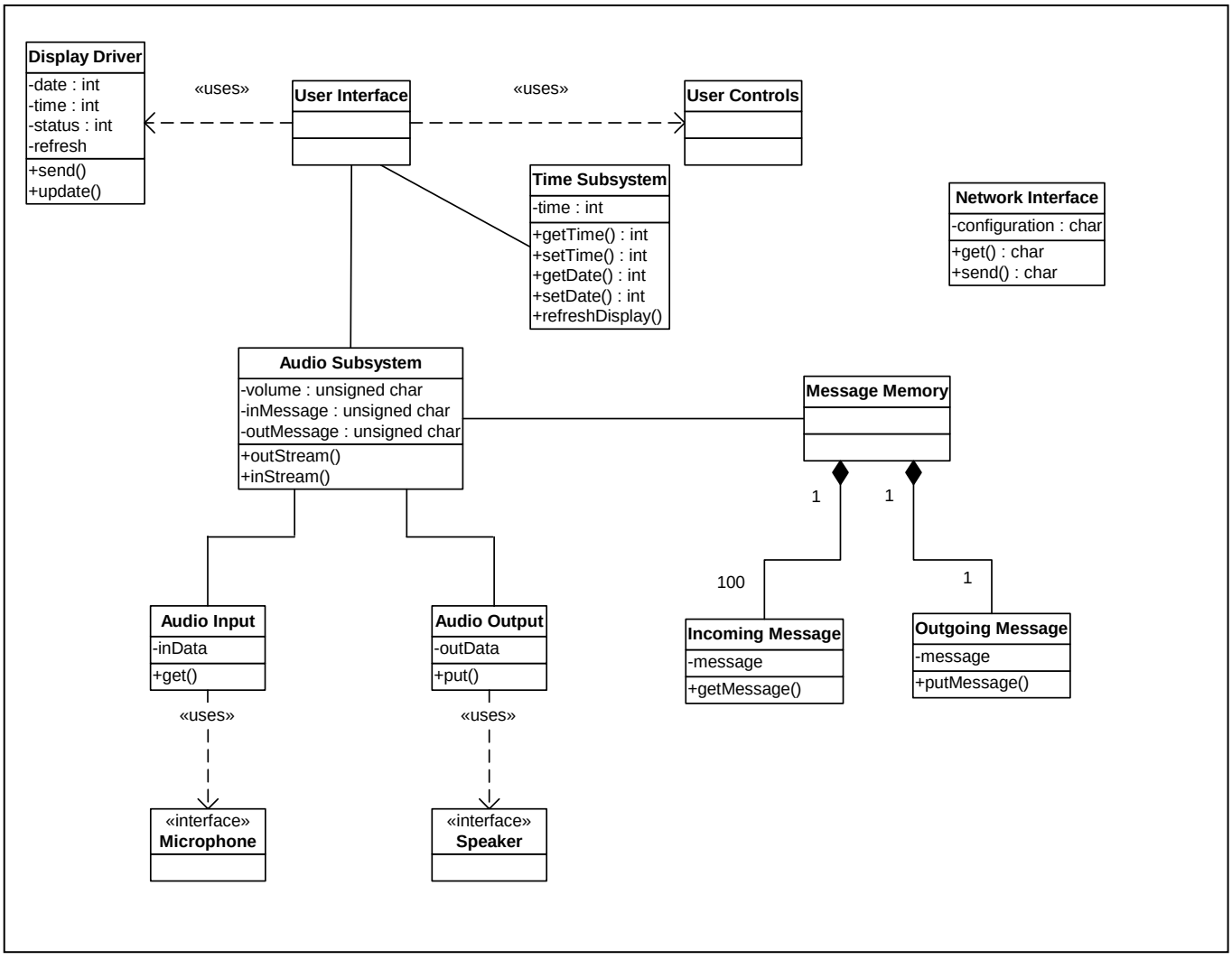
class: UserInterface	
superclass	
subclasses	
responsibilities	collaborators
1. get user input 2. manage display 3. manage status info 4. manage time functions 5. coordinate audio I/O	1. time subsystem 2. display interface 3. battery subsystem 4. keypad interface 5. audio subsystem

Class Diagram

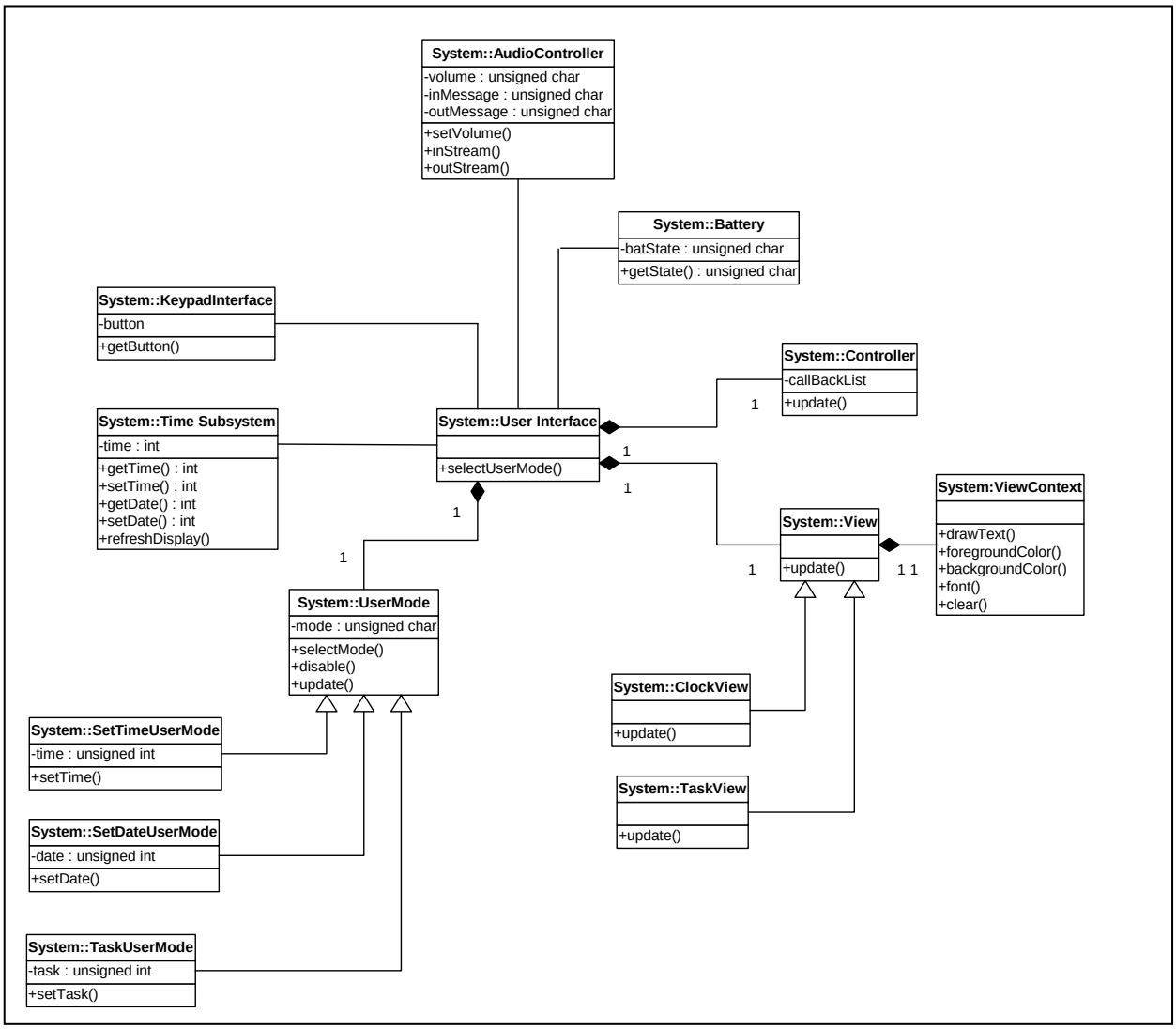
From the CRC cards, we evolve the classes and their public interfaces. The first diagram shows the diagram for the full system. The next diagram expands the User Interface class into its implementing classes.

Observe that the diagram is decomposed along the lines of major pieces of functionality…audio, memory, display, network, time…and each of these is further decomposed. A similar decomposition occurs for the UserInterface with classes that implement the major responsibilities expressed in the CRC cards.

The hardware pieces, the keyboard and the speaker, have associated software drivers.



The UserInterface (UI) class is implemented using the MVC model discussed earlier. Observe how the UI class comprises a collection other classes along the lines of different pieces of functionality. The controller has a callback list of views that must be updated whenever new information is available. The top view subclasses the time and clock views.


System::AudioController
-volume : unsigned char
-inMessage : unsigned char
-outMessage : unsigned char
+setVolume()
+inStream()
+outStream()
System::Battery
-batState : unsigned char
+getState() : unsigned char
System::KeypadInterface
-button
+getButton()
System::Controller
-callBackList
+update()
1
System::Time Subsystem
-time : int
+getTime() : int
+setTime() : int
+getDate() : int
+setDate() : int
+refreshDisplay()
System::User Interface
+selectUserMode()
1
1
1
1
System::View
+update()
1
1 1
System:ViewContext
+drawText()
+foregroundColor()
+backgroundColor()
+font()
+clear()
System::UserMode
-mode : unsigned char
+selectMode()
+disable()
+update()
System::ClockView
+update()
System::SetTimeUserMode
-time : unsigned int
+setTime()
System::TaskView
+update()
System::SetDateUserMode
-date : unsigned int
+setDate()
System::TaskUserMode
-task : unsigned int
+setTask()