Responsibility Driven Design

Responsibility Driven Design, Rebecca Wirfs Brock, 1990

The Coffee Machine Design Problem, Alistair Cockburn, C/C++ User's Journal, May and June 1998.

Introducing Object-Oriented Design with Active Learning, Rick Mercer , Consortium for Computing in Small Colleges, 2000

In Rebecca Wirfs Brocks' Words

Responsibility-Driven Design is a way to design that emphasizes behavioral modeling using objects, responsibilities and collaborations. In a responsibility-based model, objects play specific roles and occupy well-known positions in the application architecture. Each object is accountable for a specific portion of the work. They collaborate in clearly defined ways, contracting with each other to fulfill the larger goals of the application. By creating a "community of objects", assigning specific responsibilities to each, you build a collaborative model of our application.

Responsible: able to answer for one's conduct and obligations—trustworthy, Merriam Webster

Responsibility Driven Design

in Rick's words

1. Identify candidate objects that model a system as a sensible set of abstractions
2. Determine the responsibility of each object
   - what an instance of the class must be able to do,
   - and what each instance must know about itself
3. Understand the system through role play
   - To help complete its responsibility, an object often needs help from other objects

OO Design Principle

- *The Single Responsibility Principle*

  Classes should have a single responsibility

http://en.wikipedia.org/wiki/Single_responsibility_principle

- Why?
  - Cohesion, when high, reduces complexity, makes the system more understandable

http://en.wikipedia.org/wiki/Cohesion_%28computer_science%29

  - Maintenance: Fixing or changing a module should not break other parts of the system

First Design a Model

Note: design is iterative

- Find a set of objects (candidate classes) that model a solution
- Each will be a part of the bigger system
- Each should have a single responsibility
- What are these objects?

Find the Objects

- Candidate objects may come from
  - An understanding of the problem domain
    - knowledge of the system that the problem specification may have missed or took for granted
  - The words floating around the room *Alistair Cockburn*
  - The nouns in the problem statement
    - **Underline the noun phrases** to look for the objects that could model the system

The Problem Specification repeated

The student affairs office want to put some newfound activity fee funds toward a Jukebox in the student center. The Jukebox must allow students to play a song. No money will be required. Instead, a student will swipe a magnetic ID card through a card reader, view the song collection and choose a song. Students will each be allowed to play up to 1500 minutes worth of "free" Jukebox music in their academic careers, but never more than two songs on any given date. No song can be played more than five times a day*.

**What a drag it would be to hear "Dancing Queen" 14 times while eating lunch (apologies to ABBA)*

A First Cut at the Candidate Objects (may become classes)

What objects effectively model the system? What is the responsibility, Example

`Song:` Know song title, artist, playtime, how often it's been played today

Others?

Yesses

`Jukebox`: coordinates activities

one instance to start things and keep them going

`JukeboxAccount` *changed from Student*: maintain one account: model user who play songs

`Song`: one song that can be played

`CardReader`: reads the magnetic ID card

A No

`StudentIdCard`: store user data

- *Object-Oriented Design Guideline*

Eliminate classes that are outside the system

- The hallmark of such a class is one whose only importance to the system is the data contained in it.
- Student identification number is of great importance
- The system should not care whether the ID number was read from a swiped magnetic ID card, typed in at the keyboard, or "if a squirrel arrived carrying it in his mouth" *Arthur Reil*

More Candidate Objects?

- **`SongCollection`**: songs to choose from
- What about storing a collection of accounts?

  `JukeBoxAccountCollection`
- Use a compact disk player or real Jukebox?

- Could have a software equivalent like **`SongPlayer`** to play audio files?

Date

`Date`: Can determine when a song is played and the current date.

- Maybe
- Can we use use java.util.GregorianCalendar?

Another No?

`StereoSystem`: Amplifies the music

- No, it's on the other side what we have to build

- The next slide summarizes some needed candidate objects
  - It also sets the boundaries of the system
    - There are model of the real world objects

Candidate Objects *and the system boundary*

Rick drew this before UML existed

CardReader
Gets student ID

JukeboxAccountCollection
Stores all JukeboxAccount objects

JukeboxAccount

JukeBox
Coordinates activities

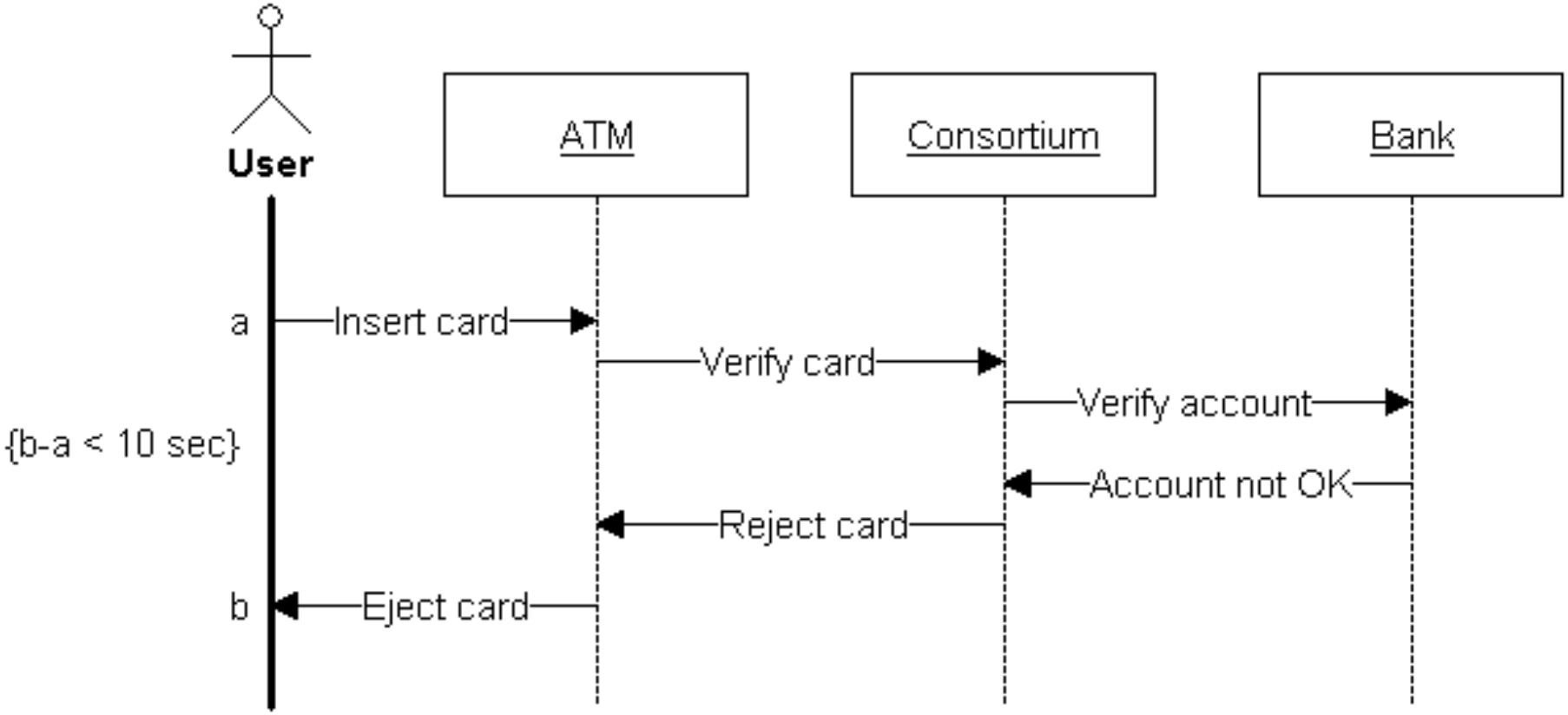
SongCollection
Stores all Songs that can be played

Song

SongPlayer
Plays a song

Another Example

http://www.ifi.uio.no/in219/verktoy/doc/html/doc/user/mg/dgmsuml6.html



Scenario: The user tries to use an ATM, but the account is not known

Role Play

- Need 7 students to play the role play the scenario Rick wants to play “Feelin’ Alright”

1. CarderReader
2. JukeboxAccountCollection
3. JukeBoxAccount
4. Jukebox
5. Songplayer
6. SongCollection
7. Song

The rest of you will have to write a sequence Diagram by hand, it will be like taking notes, a start