CS 349: Algorithms

Michael Noukhovitch

Winter 2015, University of Waterloo

Notes written from Michael Terry's lectures.

Contents

1 Introduction

1.1 Definitions

Interface: external presentation to user

- **controls**: manipulated to communicate intent
- **presentation**: what communicates response

Interaction: the actions a user must do to elicit corresponding response

- action and dialog
- unfolds over time

2 Events

2.1 Event Loop

```
while(true) {
    if there is an event on queue:
        dequeue it
        dispatch it
}
```

2.2 Timer

Some events are triggered by a timer, if that event's execution time is longer than the timer interval then by the end of the event execution, you should add another of your event to the queue!

2.3 Interactor Tree

We need a way to send information about what object is clicked
interactor tree: hierarchical tree-based organization of widgets

- each component's location is specified relative to parent
- we use **containers** whose sole purpose is to contain components
- events go **down** the tree to **capture** the target clicked
- event bubble **up** the tree to **handle** an event (e.g. EventListener)

2.4 Event Propogation

when an event happens:

1. calculate the parent node path
2. loop through it and execute capture phase handlers

3. execute DOM level 1 phase handler
4. execute bubble phase handlers
5. execute default browser behaviour

3 Model View Controller

3.1 Idea

We decouple presentation from data using the **observer** design pattern. This separation allots benefits:

- **change the UI**: easy to change how we interact with data
- **multiple view**: have different views of same data
- **code reuse**: different logic for same view etc..
- **testing**: data separation allows better logic testing

3.2 Description

Model: manages the data

- represent the data
- methods to manipulate data
- create and notify listeners

View: manages the presentation

- renders the data in a model
- references to the model
- is a listener to the model

Controller: manages user interaction

- between the model and view
- helps interpret input and model events

4 Layout

4.1 Layout Manager

Layout Manager: keeps the layout for components given their constraints and preferences

- uses composite and strategy design pattern

4.1.1 Dynamic Layout

Dynamic Layout: maintain consistency with spatial layout

- reallocate space for widget
- adjust location and size
- change visibility, look, feel

4.1.2 Layout Strategies

- **fixed layout**
- **intrinsic size**: find each item's preferred size and the container will grow to perfectly contain each item
- **variable intrinsic size**: layout determined in bottom-up and top-down phases
- **struts and spring**: items can either be fixed (strut) or variable (spring)

4.2 Responsive Design

Responsive Design: change layout to adapt to screen sizes of different devices

4.2.1 CSS

CSS: specifying formatting

- consistency
- reduce size (cache CSS)
- code reuse
- separation of concerns

CSS reset: normalize appearance across browsers

4.2.2 Cascade

Layout resolves CSS rules and renders following these rules:

1. find all declarations that match the element
2. sort declarations by `!important`
3. sort by origin (author > web browser)
4. sort by specificity of selector
5. sort by order (later rule wins)

5 Visual Design

Impose as little thinking as possible on the user

5.1 Rules

Simplicity:

- facilitate recognition instead of recall
- use only the essentials

Consistency:

- exploit perceptual patterns
- avoid ambiguous presentation
- present information consistent with user goals

Organization and Structure:

- grouping
- hierarchy
- relationship

5.2 Gestalt Principles

Theories of visual perception that describe how people organize groups

- **proximity**: elements associated with nearby elements
- **similarity**: visual similarity
- **common fate**: moving together
- **continuity**: continuous forms are easy to percieve
- **closure**: see a complete figure even when info is missing
- **symmetry**
- **area**: visual field split into background and foreground
- **uniform connection**: connecting lines/regions
- **alignment**

6 Transformation

6.1 Basics

translate add scalar

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

scale multiply by scalar

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

rotate $x' = x\cos\Theta - y\sin\Theta$
$y' = x\sin\Theta + y\cos\Theta$

$$\begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

7 Widgets

7.1 Definition

Logical Input Device : graphical component defined by its function/behaviour

Widget : part of an interface that has its own behaviour

- the **model** manipulated by the widget
- the **events** generated by the widget
- the widget **properties** which change behaviour and appearance

7.2 Types

7.2.1 Simple Widgets

- labels and images
- button
- boolean *e.g. radio button*
- number *e.g. slider*
- text field

7.2.2 Container Widgets

- pane
- tab
- menu
- list choice *e.g. dropdown menu*

7.2.3 Abstract Model Widgets

...

7.3 Widget Toolkit

Widget Toolkit: software that defines a set of (event-driven) GUI components via API

- **complete**: GUI designers have all they need
- **consistent**: look, feel and usage paradigms are consistent
- **customizable**: devs can reasonably extend functionality

There are two common implementations of widgets:

- **Heavyweight widgets**: OS provides widgets and hierarchical windowing system *e.g. X, HTML*
- **Lightweight widgets**: OS provides top-level window, use toolkit for drawing and events *e.g. Java Swing*

8 Responsiveness

Responsiveness: the fulfillment of a user's real-time needs

8.1 Human Deadline

Make your program seem faster by using tricks to respond to a user's actions:

- busy indicator
- progress indicator
- rendering important information first
- fake heavyweight computations *e.g. scrolling*
- work ahead during periods of low load

8.2 AJAX

Classic web architechture relied on 'thin client, fat architechture', side step it using JS:

- AJAX issues call to web server (API)
- server handles request and returns data feed
- client updates UI with feed using JS

Advantages:

- minimize bandwidth
- increase speed
- avoid HTML headers

Example 8.1. AJAX backend using Bottle (Python)

```python
from bottle import route, run, template, request

@route('/ajax', method='POST')
def ajax():
    # Retrieve the JSON
    json_data = request.json

    # upper-case and return the text
    upper = json_data['text'].upper()

    # Bottle will automatically send back
    # Python dictionaries as JSON objects
    return {'reply': upper}
```

9 Design Process

We need to have descriptions of UI to clarify intent and **define interaction**.

9.1 Formal Language

The design process can be strongly described with formal languages such as **Finite State Machines** allowing us to label states and transitions. But this rigid formality has troubles describing the rich informal world of the user:

- complexity of interfaces is too great
- can't clearly represent some ideas *e.g. timed events*
- transition time (to create/update model) is not negligable

9.2 User Centered Design

Design (and test) with real people in mind, using semi-formal languages

- understand user needs
- design UI first (then architecture)
- iterate
- use it yourself
- observe others using it

This can be put into a design process:

9.2.1 Understand the User

- observe existing solutions
- list scenarios
- list functions required
- prioritize (freq and commonality)

9.2.2 Design the UI

- identify and design components
- design component distributions
  - storyboards
  - interaction sequeunces: macro structure
  - interface schematics: micro structure
- test the design with users
  - prototyping (high vs low fi, paper, Wizard of Oz)
  - user/usability studies
- iterate again!
- document the design
  - visual vocabulary

9.2.3 Refine the Design

- refine requirements
- add new scenarios
- walk through new scenarios
- adjust UI design