ECE 651 – Software Engineering
Week Thirteen – SOA, Mobile and Web
April 13th, 2016

Ric Telford
Adjunct Associate Professor

Founder, Telford Ventures

Duke | PRATT SCHOOL OF ENGINEERING

04/13/16 – Week 13 Overview

- Recap / Announcements
- Lecture – Service Oriented Architecture
- Lecture – Mobile App Design
- Break
- Visit from Digital Turbine
- Lecture – Web App Design
- About Week 14

Recap

- Software Evolution
  - For custom systems, the costs of software maintenance usually exceed the software development costs.
  - The process of software evolution is driven by requests for changes and includes *change impact analysis, release planning and change implementation*.
  - *Legacy systems* are older software systems, developed using obsolete software and hardware technologies, that remain useful for a business.
  - The business value of a legacy system and the quality of the application should be assessed to help decide if a system should be *replaced, transformed or maintained.*
  - There are *3 types of software maintenance*, namely bug fixing, modifying software to work in a new environment, and implementing new or changed requirements.
  - *Software re-engineering* is concerned with re-structuring and re-documenting software to make it easier to understand and change.
  - *Refactoring*, making program changes that preserve functionality, is a form of preventative maintenance.
- Distributed Systems
  - The *benefits of distributed systems* are that they can be scaled to cope with increasing demand, can continue to provide user services if parts of the system fail, and they enable resources to be shared.
  - Issues to be considered in the design of distributed systems include *transparency, openness, scalability, security, quality of service and failure management.*
  - Client-server systems may have several tiers, with different layers of the system distributed to different computers.
  - *Architectural patterns for distributed systems* include master-slave architectures, two-tier and multi-tier client-server architectures, distributed component architectures and peer-to-peer architectures.
  - Distributed component systems require *middleware* to handle component communications and to allow components to be added to and removed from the system.
  - *Peer-to-peer architectures* are decentralized with no distinguished clients and servers. Computations can be distributed over many systems in different organizations.
  - *Software as a service* is a way of deploying applications as thin client- server systems, where the client is a web browser.

Announcements

- You are invited to attend the DUhatch Showcase at 7pm, April 14th, Schiciano
- For anyone that has not earned your full 5 points of **extra credit**, you can do that by attending, BUT you MUST stay for all presentations (first hour) and **sign-in and sign-out** with me
- Free food!

DUhatch COMPANY SHOWCASE Spring 2016

DUhatch Business Incubator @ the Pratt School of Engineering,
Welcomes you to the DUhatch Company Showcase- Spring 2016!

The DUhatch Company Showcase is a biannual event organized by DUhatch, the student business incubator at Duke University, to showcase the work being done by the current batch of student companies, who have start-ups in areas as diverse as embedded sensor technology, nutrition, image processing and so on. Guests will include executives, entrepreneurs from RTP and the Duke Entrepreneurship network. There'll be great people to meet and some amazing free food!

WHEN: 14 APRIL, THURSDAY

WHERE: SCHICIANO AUDITORIUM from 7 PM-9 PM

SEE YOU THERE!

Project Presentations

- Project Presentations are next week
- Presentation is worth 50 points.
  - You have up to 15 minutes, but that includes a few minutes for Q&A
  - Slides are optional, but do need to take us through your documentation and give a demonstration.
  - Not everyone needs to speak but team needs to be at front of room
- Things I will be looking for in the presentations:
    - How well organized and prepared were the presenters?
    - How well did they structure for a 15 minute presentation?
    - Did they cover the key elements of the documentation?
    - How interesting was the demonstration of their project?
    - Did the code run?
    - Did they cover the key functions?
    - Did they show “real” data like you would expect to see in a product demo or was it “garbage” data?
    - Did we come away with a good understanding of what the problem is they are trying to solve, and how the project addressed that problem?

Project Documentation

- Project Documentation is due next week
- Documentation is worth 100 points.
  - At a minimum needs to contain all the sections in “Example Project Documentation.pdf”
  - The more (relevant) content the better, especially when it comes to depth of Requirements, Architecture and Design content
- Things I will be looking for in the documentation:
    - How well is the document organized and does it flow logically?
    - Are there a lot of typos, misspellings, or other errors?
    - Does everything match the actual code (design, architecture, plan, etc)?
    - Does the document meet or exceed what was requested?
    - Are the Requirements of sufficient scope and granularity?
    - Does the Architecture give a good overview of the project?
    - How expansive is the level of Design documentation and would it be sufficient for someone to code to?
    - Was the overall Project / Test plan complete?
    - Is the project documentation complete enough for someone to install and use the system?

Chap 18 – Service-oriented Software Engineering - Topics covered

- 18.0
- 18.1 - Service-oriented architectures
- 18.2 - RESTful services
- Service engineering
- Service composition

Web services

- A web service is an instance of a more general notion of a service:

  "an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production".

- The essence of a service, therefore, is that the provision of the service is independent of the application using the service.
- Service providers can develop specialized services and offer these to a range of service users from different organizations.

Reusable services

- Services are reusable components that are independent (no requires interface) and are loosely coupled.
- A web service is:
  - *A loosely coupled, reusable software component that encapsulates discrete functionality, which may be distributed and programmatically accessed. A web service is a service that is accessed using standard Internet and XML-based protocols.*
- Services are platform and implementation-language independent

Benefits of service-oriented approach

- Services can be offered by any service provider inside or outside of an organization so organizations can create applications by integrating services from a range of providers.
- The service provider makes information about the service public so that any authorized user can use the service.
- Applications can delay the binding of services until they are deployed or until execution. This means that applications can be reactive and adapt their operation to cope with changes to their execution environment.

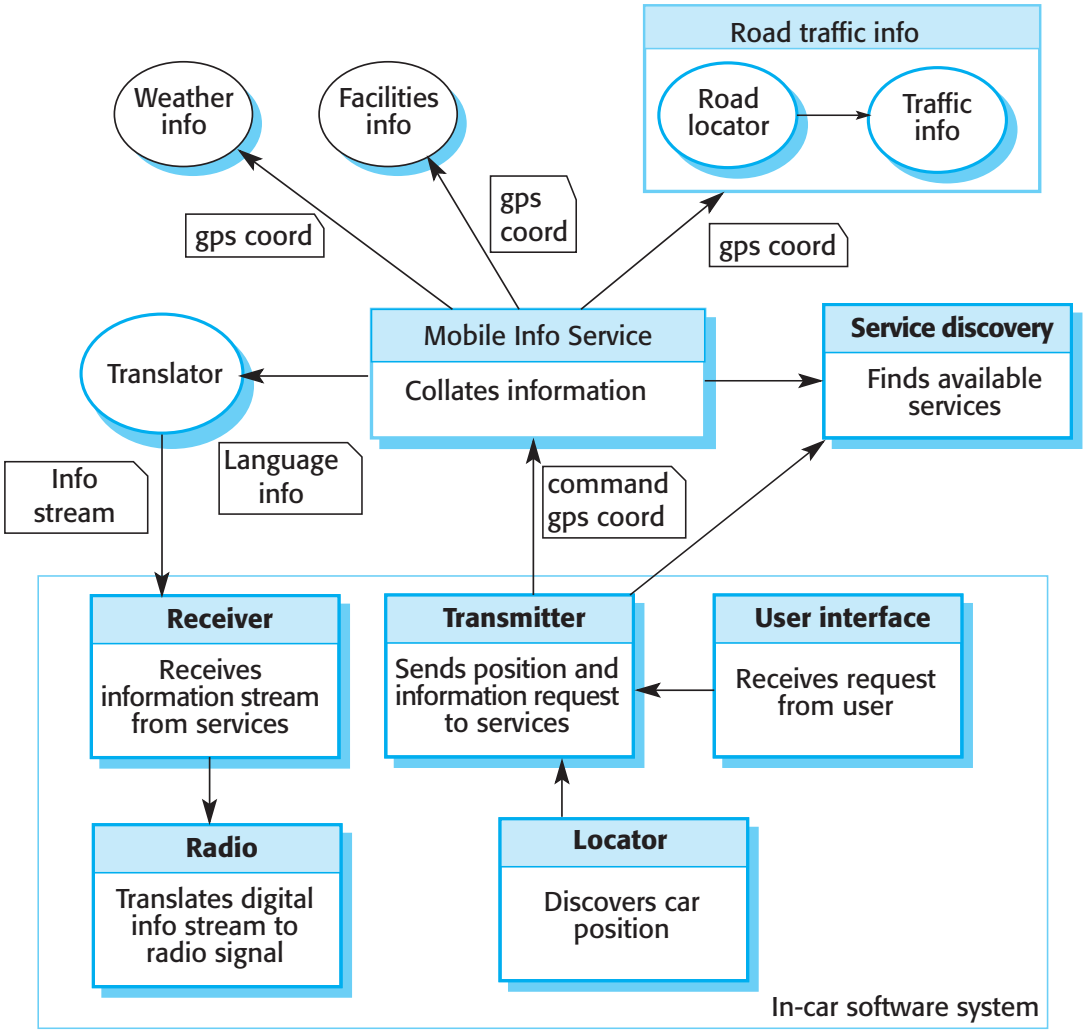
Benefits of a service-oriented approach

- Opportunistic construction of new services is possible. A service provider may recognize new services that can be created by linking existing services in innovative ways.
- Service users can pay for services according to their use rather than their provision. Instead of buying a rarely-used component, the application developers can use an external service that will be paid for only when required.
- Applications can be made smaller, which is particularly important for mobile devices with limited processing and memory capabilities. Computationally-intensive processing can be offloaded to external services.

Services scenario

- An in-car information system provides drivers with information on weather, road traffic conditions, local information etc. This is linked to car audio system so that information is delivered as a signal on a specific channel.
- The car is equipped with GPS receiver to discover its position and, based on that position, the system accesses a range of information services. Information may be delivered in the driver's specified language.

A service-based, in-car information system



Advantage of SOA for this application

- It is not necessary to decide when the system is programmed or deployed what service provider should be used or what specific services should be accessed.
  - As the car moves around, the in-car software uses the service discovery service to find the most appropriate information service and binds to that.
  - Because of the use of a translation service, it can move across borders and therefore make local information available to people who don't speak the local language.

Service-oriented software engineering

- As significant a development as object-oriented development.
- Building applications based on services allows companies and other organizations to cooperate and make use of each other's business functions.
- Service-based applications may be constructed by linking services from various providers using either a standard programming language or a specialized workflow language.

Service-oriented architecture

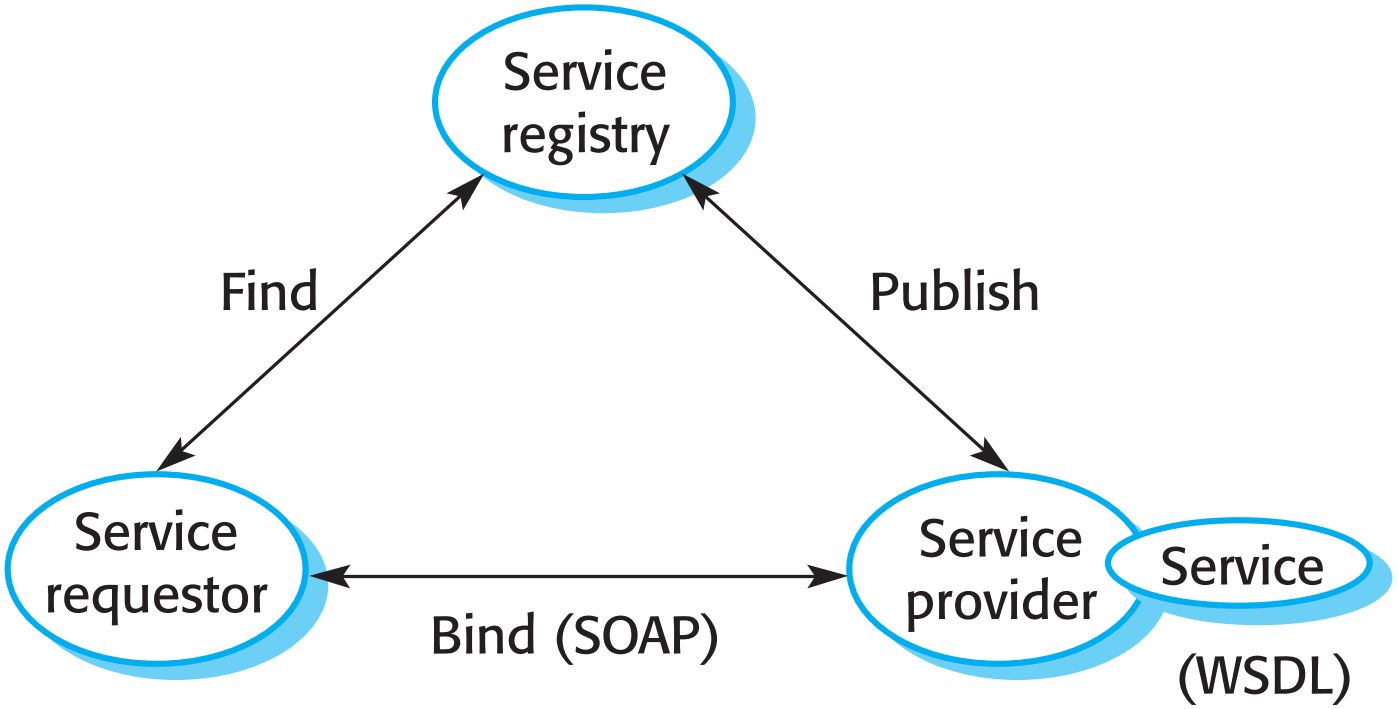
Service-oriented architectures

- A means of developing distributed systems where the components are stand-alone services
- Services may execute on different computers from different service providers
- Standard protocols have been developed to support service communication and information exchange

Key standards

- SOAP
  - A message exchange standard that supports service communication
- WSDL (Web Service Definition Language)
  - This standard allows a service interface and its bindings to be defined

Service-oriented architecture



Benefits of SOA

- Services can be provided locally or outsourced to external providers
- Services are language-independent
- Investment in legacy systems can be preserved
- Inter-organizational computing is facilitated through simplified information exchange

RESTful services

RESTful web services

- Current web services standards have been criticized as ‘heavyweight’ standards that are over-general and inefficient.
- REST (REpresentational State Transfer) is an architectural style based on transferring representations of resources from a server to a client.
- This style underlies the web as a whole and is simpler than SOAP/WSDL for implementing web services.
- RESTful services involve a lower overhead than so-called ‘big web services’ and are used by many organizations implementing service-based systems.

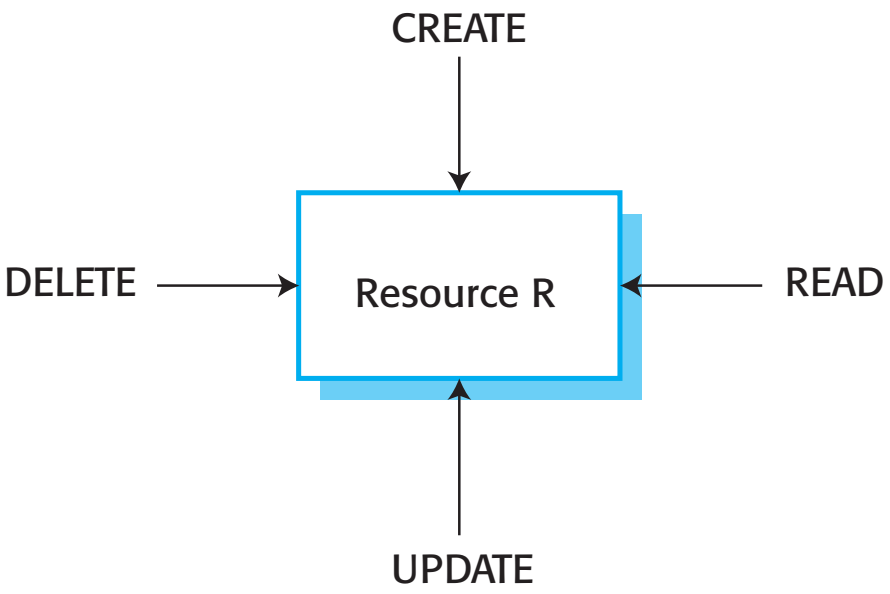
Resources

- The fundamental element in a RESTful architecture is a resource.
- Essentially, a resource is simply a data element such as a catalog, a medical record, or a document, such as this book chapter.
- In general, resources may have multiple representations i.e. they can exist in different formats.
  - MS WORD
  - PDF
  - Quark XPress

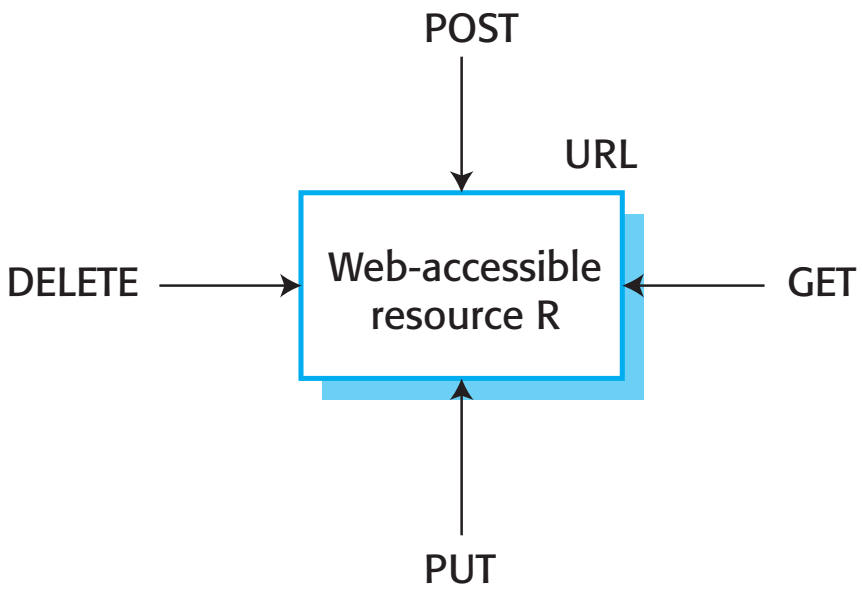
Resource operations

- Create – bring the resource into existence.
- Read – return a representation of the resource.
- Update – change the value of the resource.
- Delete – make the resource inaccessible.

Resources and actions



(a) General resource actions



(b) Web resources

Operation functionality

- POST is used to create a resource. It has associated data that defines the resource.
- GET is used to read the value of a resource and return that to the requestor in the specified representation, such as XHTML, that can be rendered in a web browser.
- PUT is used to update the value of a resource.
- DELETE is used to delete the resource.

Resource access

- When a RESTful approach is used, the data is exposed and is accessed using its URL.
- Therefore, the weather data for each place in the database, might be accessed using URLs such as:
  - http://weather-info-example.net/temperatures/boston
    http://weather-info-example.net/temperatures/edinburgh
- Invokes the GET operation and returns a list of maximum and minimum temperatures.
- To request the temperatures for a specific date, a URL query is used:
  - http://weather-info-example.net/temperatures/edinburgh?date=20140226

Query results

- The response to a GET request in a RESTful service may include URLs.
- If the response to a request is a set of resources, then the URL of each of these may be included.
  - http://weather-info-example.net/temperatures/edinburgh-scotland
    http://weather-info-example.net/temperatures/edinburgh-australia
    http://weather-info-example.net/temperatures/edinburgh-maryland

Disadvantages of RESTful approach

- When a service has a complex interface and is not a simple resource, it can be difficult to design a set of RESTful services to represent this.
- There are no standards for RESTful interface description so service users must rely on informal documentation to understand the interface.
- When you use RESTful services, you have to implement your own infrastructure for monitoring and managing the quality of service and the service reliability.

Key points

- *Service-oriented architecture* is an approach to software engineering where reusable, standardized services are the basic building blocks for application systems.
- Services may be implemented within a service-oriented architecture using a set of XML-based *web service standards*. These include standards for service communication, interface definition and service enactment in workflows.
- Alternatively, a *RESTful architecture* may be used which is based on resources and standard operations on these resources.
- A RESTful approach uses the http and https protocols for service communication and maps operations on the standard http verbs POST, GET, PUT and DELETE.

Mobile App Design

- What are some of the unique design considerations when building a mobile app vs. traditional?
  - Device limitations (computation and storage)
  - User interface limitations
  - Short development cycles
  - Power management
  - App stores with differing acceptance rules and tool requirements
  - Changing connectivity
- What are some other key design considerations?
  - Multiple hardware and software platforms
  - Security and privacy models/policies
  - Integration of external services

User Interface Design Considerations

- Overall page design – is it easy to read and navigate?
- Does the app make optimal use of space (table views, etc)?
- Does the app take screen size differences into account?
- Does the user interface conform to the display and interaction standards adopted for the targeted mobile device(s)?
- Is the UI consistent across different views in the app?
- Does the test plan cover all targeted user environments and for all targeted devices?
- Does the maintenance plan in place to ensure the app remains current as UI guidelines change?
- Is the user interface as SIMPLE as it can be?

Some lessons from doing development for Apple devices

- Apple provides style guides and expects you to follow them. If there is an applicable "style" for the app you are creating, you should use it.
- Apple templates for some of the key styles are useful, but make sure you know what the "template" code is doing.
- A lot of what Apple does is there to make maximum use of a small screen. Unclear how relevant this is when developing for an iPad Pro
- Xcode (the Apple IDE) assumes an MVC approach to app design.
- Apple has a rigid acceptance process for the App Store and can reject your app for any number of design or development reasons

Mobile and Services

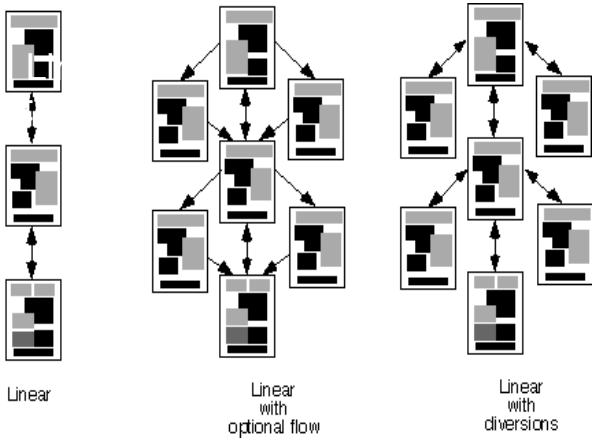
- Mobile apps are a perfect use case for service discovery and composition
- Allows developers to write less code on the mobile device and instead leverage functionality in the network
- “Loosely coupled” model means no need to “bind” to the service before runtime
- APIs allow services to be treated like an abstract black box

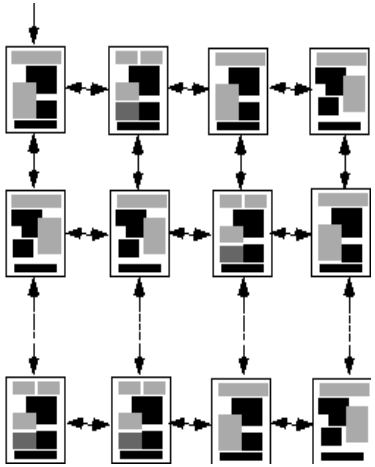
Mobile and Cloud Computing

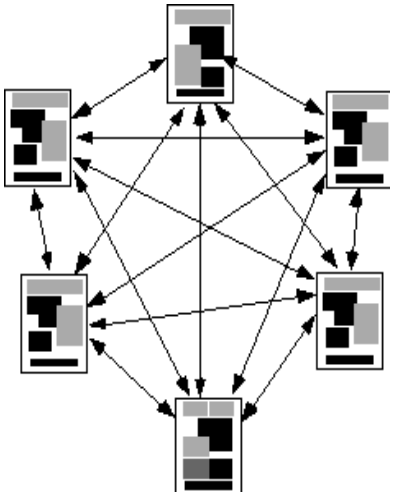
- Cloud is also key to the emergent mobile app model.
- Allows for dynamic growth of an app's services as the user base grows
- Focuses on the effective delivery of services to users through flexible and scalable resource virtualization and loading balancing
- Allows the app to request computing capabilities as needed, across network boundaries anywhere or any time

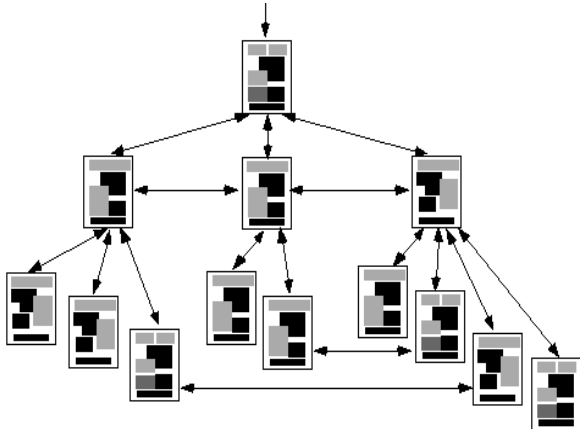
BREAK, followed by presentation from Digital Turbine

The uniqueness of a Web Application derives from the notion of hyperlinks and how you organize these links

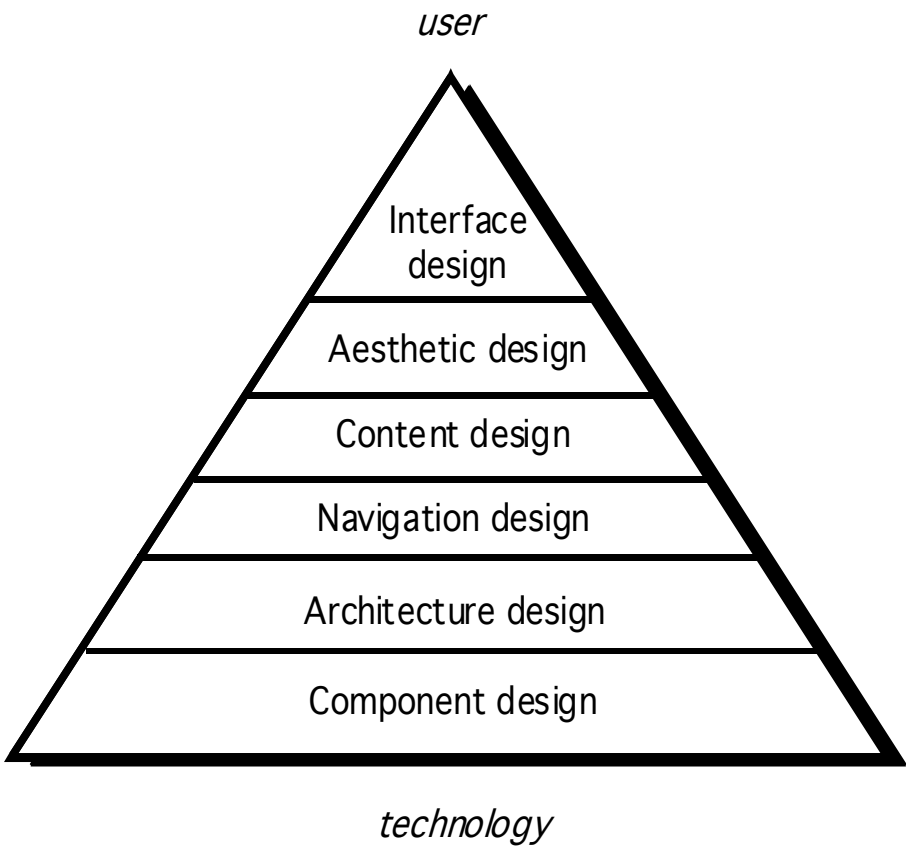








Web Design Pyramid



Web Application Design Concepts

- *Interface design* is similar to any other UI guidelines, except in the Web you have no idea where the user entered
- *Aesthetic design* means that the application presents a consistent and appealing look across all parts of the application
- *Content design* focuses on the data objects and their relationships
- *Architecture design* establishes templates that lead to a consistent hypermedia structure
- *Navigation design* provides for consistency in the way links work across the application
- *Component design* should define consistent modes of interaction, navigation and content display

Interface Design

- Where am I? The interface should
  - provide an indication of the WebApp that has been accessed
  - inform the user of her location in the content hierarchy.
- What can I do now? The interface should always help the user understand his current options
  - what functions are available?
  - what links are live?
  - what content is relevant?
- Where have I been, where am I going? The interface must facilitate navigation.
  - Provide a "map" (implemented in a way that is easy to understand) of where the user has been and what paths may be taken to move elsewhere within the WebApp.

Aesthetic Design (or Graphic Design) – laying out the visual elements of the web application

- Don't be afraid of white space.
- Emphasize content.
- Organize layout elements from top-left to bottom right.
- Group navigation, content, and function geographically within the page.
- Don't extend your real estate with the scrolling bar.
- Consider resolution and browser window size when designing layout.

Content Design (Class Models, Object Models)

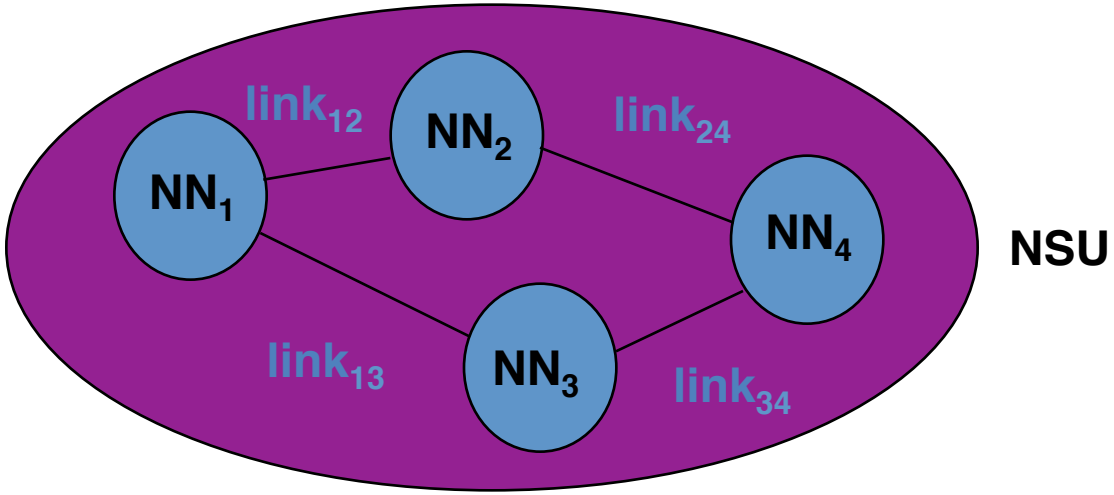
- Develops a design representation for content objects
  - For Web applications, a content object is more closely aligned with a data object for conventional software
- Represents the mechanisms required to instantiate their relationships to one another.
- A content object has attributes that include content-specific information and implementation-specific attributes that are specified as part of design

Navigation Design

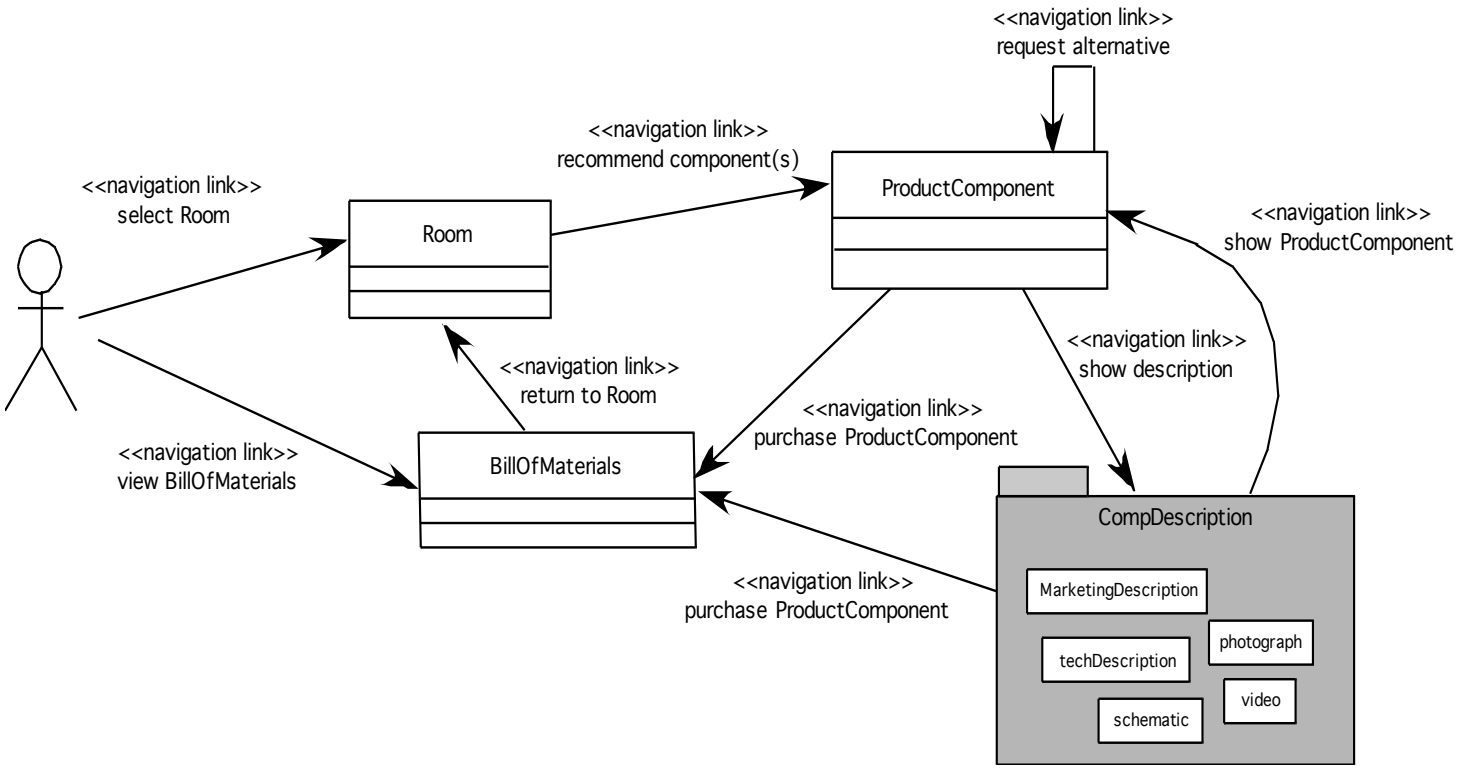
- Begins with a consideration of the user hierarchy and related use-cases
  - Each actor may use the application somewhat differently and therefore have different navigation requirements
- As each user interacts with the application, they encounter a series of navigation semantic units (NSUs)
  - NSU—"a set of information and related navigation structures that collaborate in the fulfillment of a subset of related user requirements"

Navigation Semantic Units

- Navigation semantic unit
  - Ways of navigation (WoN)—represents the best navigation way or path for users with certain profiles to achieve their desired goal or sub-goal. Composed of …
    - Navigation nodes (NN) connected by Navigation links



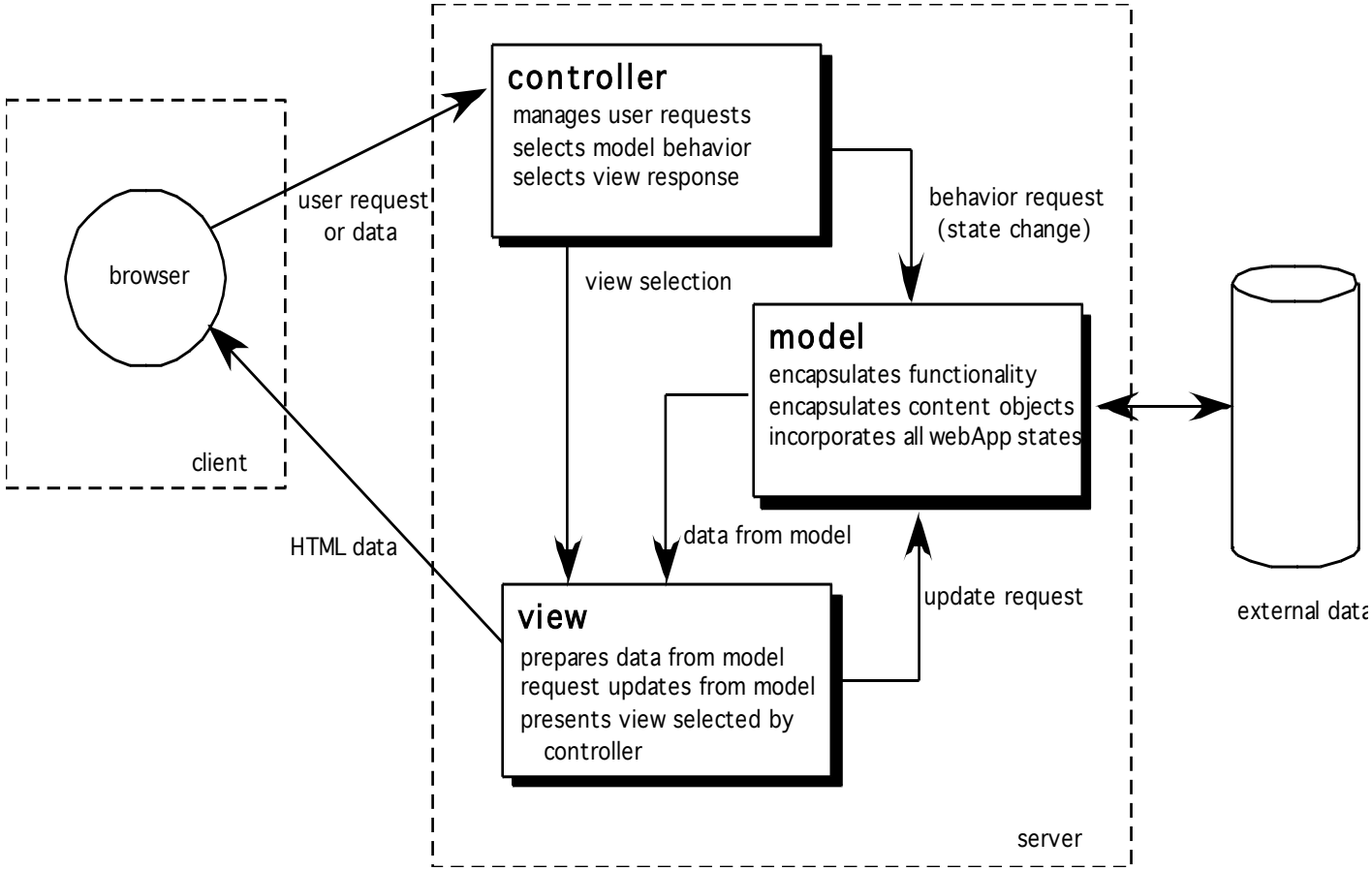
Creating an NSU



Architecture Design

- Content architecture focuses on the manner in which content objects (or composite objects such as Web pages) are structured for presentation and navigation.
  - The term information architecture is also used to connote structures that lead to better organization, labeling, navigation, and searching of content objects.
- Web application architecture addresses the manner in which the application is structured to manage user interaction, handle internal processing tasks, effect navigation, and present content.
- Architecture design is conducted in parallel with interface design, aesthetic design and content design.

MVC Architecture on the Web



Component-Level Design

- Web application components implement the following functionality
  - perform localized processing to generate content and navigation capability in a dynamic fashion
  - provide computation or data processing capability that are appropriate for the application's business domain
  - provide sophisticated database query and access
  - establish data interfaces with external corporate systems.

About Week 14 – April 20th

- Make sure your final review with your TA is complete this week!
- Make sure your project Documentation is uploaded in Sakai by 11pm on April 19th (TUESDAY)
- I will move all Documentation to Resources on WEDNESDAY the 20th so we can follow along.
- We will start right at 4:40 with a random draw and then the first presenter
- We won't take a break but feel free to take a short break if needed

Reference Material

Some of the content in this slide deck is taken from

Software Engineering: A Practitioner's Approach, 8/e

by Roger S. Pressman and Bruce R. Maxim