SFWR ENG 3A04 Summary

Author: Kemal Ahmed
Instructor: Dr. Ridha Khedri
Date: Fall 2014

Math objects made using MathType; graphs made using Winplot.

Please join GitHub and contribute to this document. There is a guide on how to do this on my GitHub.

Table of Contents

Lecture 2

Hierarchy of Requirement Specifications

Pre Requirements:

- Requirements:
  - Requirements Document
    - System Specifications
    - Other Documents
      - Legal
      - Security
      - Privacy
  - Architectural Design
    - Types:

        - Dynamic
        - Stable
        - Determined by:
            - Elements
            - Connectors
    - Detailed Design

Traceability Matrix

Traceability Matrix: a method of showing how each of the elements satisfies a requirement. You can use this to determine if a feature is necessary or if you are missing a feature.

Elements (E_i) \ Requirements (R_i)	R_1	R_2	R_n
E_1		P	P
E_2	T		
E_n			

Early Assignment Details

- The assignment can be submitted to a contest
- 2014-15 connect
- dx.org/connect
- Deadline: April 1st, 2015
- Prize: $2000

Requirements Cont.

Business Event (BE): the first, initiating input to a system that, but worded in the form of an event
Note: time can be an event, e.g. time to update your clocks

Environment / system interactions:

- *I/O between system and user*
- look at the system as a black box
- the last output occurs when the "business has been carried

Viewpoints (VP):

- *A target set of requirements*
- Think of it as different perspectives of how someone would want the system to be designed
- Includes things like who is using your product, but also who will be affected, such as economic perspective, i.e. cost

The more viewpoints you have, the better the representation of the system because you get a better overall perspective.

e.g. 1)

For a BE_1, you have a list of VPs from VP_1 to VP_n, and for BE_2 you have a list of VPs from VP_1 to VP_m.

If you have 2 viewpoints that have little relevance, you don't get rid of it. Instead, you mark them as void. This is because you may need it for the next BE(s)

Functional Requirements: something the system must do
Non-functional Requirements: properties the system must have, e.g. precision, availability, security, usability, look, etc.
Constraint: global issue that shapes the requirements

Determine functional, *then* non-functional requirements.

Scenario: interactions between the system and the user / environment (could be time)

Mode: what you think it means, but formally, a non-empty set of equivalent states

- reflexive
- transitive
- symmetric
- $x'Ry$ and $y'Rx$

Complete graph with n nodes is K_n.

Design Space

- Hardware-hiding modules:
  - Language to communicate with the hard drive
  - Virtual Machine hiding module
- Behaviour hiding modules:
  - Controller classes: sequence of events
  - Change due to requirements
- Software decision-hiding modules:
  - Algorithms
  - Physics constants
  - Theorems (i.e. math)
  - Data types
    - n-Tuple; a record
      - n gets
      - n sets
    - Set
      - IsMember
      - IsEmpty

        - Insert
        - Remove
    - List
        - IsEmpty
        - GetHead
        - GetNext (last element)

Asynchronous operation: process operates independently of other processes
Synchronous operation: other processes finish before some other process has finished
Blocking: process causes other processes to stop
Non-blocking: process runs without stopping other processes
More

Semaphore:

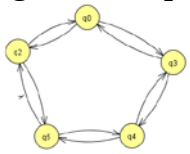
Protocol: a method of communication

MVC: the way every software program is analyzed
Model: (a.k.a. Data level) constants and stored data the system interacts with
View: (a.k.a. Interface) what the users see and how they interact with the system
Controller: (a.k.a. Business Logic) what processes the data from the model

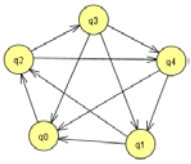
Connector:
Signature-based connector: works as long as you communicate using the correct inputs (like Radio)
Protocol-based connector: when communicating, both communicate with each other and confirm a connection (like WiFi)

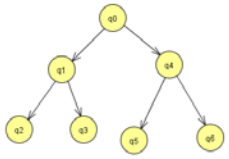
Formal model: a representation of what you are going to buildbased on math
Informal model: not formal

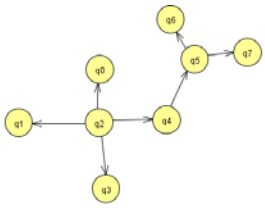
Configuration topology: different shapes of networks, including bus ignore arrows

ring 

star 

mesh 

hierarchy 


and extended star

Unified Modelling Language (UML):

Class Name
Attributes: name: String address: String
Operations / Functions

It is usually organized in **structural diagrams**, which show relationships between classes through connectors.

Architecture Description Language (ADL):

Inheritance: [*identified by arrows*] the lower object gets some of its data / functions from the higher objects, although local functions have higher presidence
Aggregation: [*identified by black diamonds*] something is made of parts which aren't useless on their own
Composition: [*identified by hollow diamonds*] any combination of the higher object can make up the lower object

It's especially important to have low coupling when you can't change the higher level object

Diagram Types

Dashed arrows: dependencies

Structural

Composite Structure Diagram

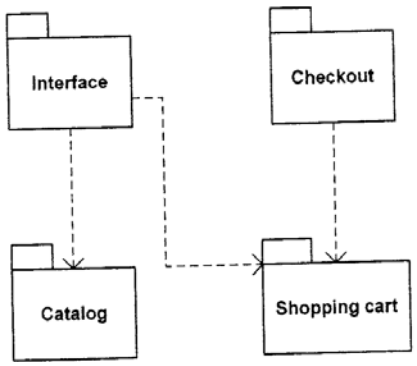
- Rectangle: structural classes
- Ellipse: abstract construct of relationship between classes

Component Diagram

- Balls: class that outputs
- Sockets: class that takes input from balls

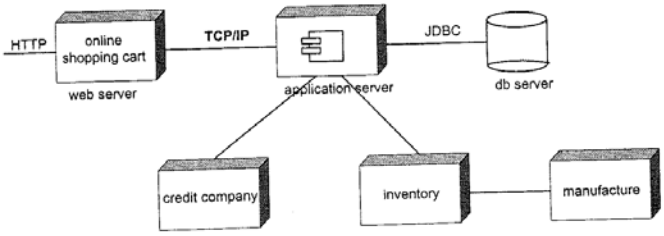
Package Diagram: package structure

- Folders: packages



Deployment Diagram: physical hardware, software, network connections

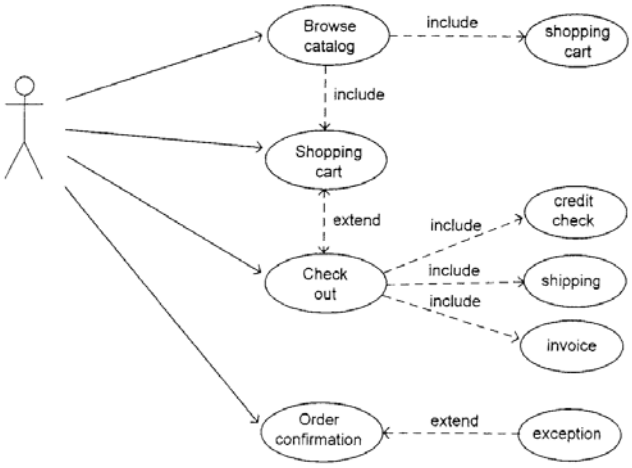
- Cubes: computing resources
- Cylinders: database [sometimes]



Behavioural

Use Case: how system reacts to BEs

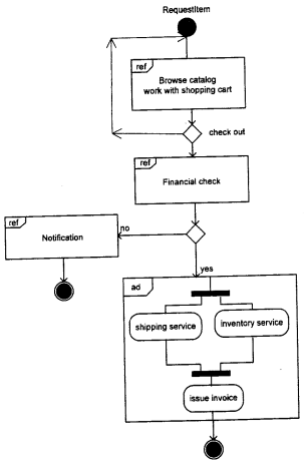
- Communication between actors
- **Actors**: [represented by a stick figure] does not have to be a human
- Include: mandatory behaviour
- Extend: optional behaviour
- "Use Case" ⇔ "Scenario"
- Each ellipse is a use case



Activity Diagram: data and control flow of system

- Rounded rectangles: actions in system
- Solid hub: fork and joint points

- Surrounded disk: terminate
- Diamond: decision



Sequence Diagram: how flow thru classes to fulfill requirements

- Rectangles on top identify classes
- Arrows show flow of data and how they fulfill requirements
- Smaller boxes inside the bigger boxes are other implementations of the same object