CS 446: Software Design and Architecture

Michael Noukhovitch

Spring 2016, University of Waterloo

Notes written from Victoria Sakhini's lectures.

Contents

1 Mobile Application

1.1 Overview

A mobile application is structured of mulitple layers: **presentation**, **business**, and **data**.

1.2 Design Considerations

1.2.1 Client Type

Rich local processing required, must work in ocassionally connected scenario

Thin can depend on server processing and will always be fully connected

Rich Internet Application requires a rich UI and only limited access to local resources (+ maybe portably to other platforms)

1.2.2 Devices to Support

Consider

- screen size and resolution
- cpu power
- memory and storage space
- dev tool availability
- user requirements, org constraints
- specific hardware requirements

1.2.3 Connectivity

If internet access is required, plan for intermittent or unavailable network connection

- caching
- state management
- batch communications

1.2.4 Device Constraints

Think of platform constraints, mainly:

- memory
- battery life
  - processing requirements
  - backlighting
  - memory I/O

    - wireless connections
- responsiveness of design
- security
- network bandwidth

1.2.5 Architecture

- layered architecture (multiple layers can be on device)
- reuse and maintainability
- smallest footprint possible

1.3 Design Issues

1.3.1 Authentication/Authorization

- security and reliability
- think about more than single user

1.3.2 Caching

- improve performance
- support offline work
- decide on what to cache based on limited resources

lazy acquisition defer acquiring resources as long as possible

1.3.3 Communicaion

- wifi, wired, bluetooth
- secure communication
- wireless is unreliable

active object support async processing by encapsulating service request and completion response

communicator encapsulate internal details of communication

entity translator transforms message data types into business types for requests and reverses for responses

reliable sessions end to end reliable message transfer

1.3.4 Configuration Management

- how to handle device resets
- how to allow configuration (OTA, from some host?)

1.3.5 Data Access

- low bandwidth
- high latency
- intermittent connectivity

active record include data access object within domain entry

data transfer object object storing data transported between processes, reducing method calls

domain model business objects that represent entities in a domain and relationships between them

transaction script organize logic for each transaction in a single procedure, making calls directory to DB (or through wrapper)

1.3.6 Device Specifics

- screen size
- orientation
- memory, storage space
- network bandwidth
- connectiviy
- OS
- hardware constraints

1.3.7 Exception Management

- prevent sensitive exception details from being revealed to the user
- improve application robustness
- keep application in consistent state after an error

1.3.8 Logging

- log only essentials because of size constraints
- may need to synchronize logs with server

1.3.9 Power Management

- power is limiting design factor
- research communication protocols and their effect on battery life

1.3.10 Synchronization

- secure communications OTA
- handle connection interruptions

sync design pattern component installed on device tracks changes to data and tells server when connected

1.3.11 Testing

Mobile debugging is costly, so make sure to invest heavily in testing beforehandas emulators may not be adequate to simulate a device in debugging.

1.3.12 UI

- build mobile first
- design for simplicity
- design around blocking operations (since user can only see once screen at a time)

application controller object that contains all the flow logic

MVC separate the data, presentation, and actions into three separate classes

  - model manages behaviour and data (logic)
  - view manages information display
  - controller manages user input

MVP same as MVC but presenter manages presentation logic and interaction between view and model

pagination separate content into individual pages

1.3.13 Validation

- protect device and application
- improve usability
- validate client-side and server-side

2 Software Architecture

2.1 Definition

No perfect definition but AINSI/IEEE defines it as

> recommended practice as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

major concepts

- processing/functionality/behaviour
- data/information/store
- interaction/communication/coordination

2.2 Components

component encapsulates processing and data

- encapsulates a subset of the system's functionality and/or data
- restricts access to that subset via an explicitly defined interface
- has explicitly defined dependencies on its required execution context
- typically provides application-specific interface

2.3 Connectors

connector effecting and regulating interactions among components

- can be simple procedure calls or shared data access
- provide application independent interaction facilities

functions:

- modelling arbitrary complex interactions
- aiding system evolution (w/ flexibility)
- support for connector interchange

2.3.1 Roles

- communication
  - supports different communication mechanisms (procedure call, message passing)
  - constraints on commincation structure (pipes)
  - constriaints of quality of service (persistence)
  - separates communication from computation

- coordination
  - determine computation control
  - control delivery of data
  - separate control from computation
  - elements of control are in communication, conversion, facilitation
- conversion
  - interaction of mismatched components (adaptors, wrappers)
  - mismatches based on interaction
    - type
    - number
    - frequency
    - order
- facilitation
  - mediate and streamline interaction of components intended to interoperate
  - govern access to shared data
  - ensure performance profiles (load balancing)
  - provide sync mechanisms (critical sections, monitors)

2.3.2 Types

- procedure call
- data access
- event
- stream
- linkage
- distributor
- arbitrator
- adaptor

2.3.3 In Practise

To select a connector, determine:

- interaction and interconnection needs
- roles to be fulfilled (e.g. communication)
- each connector's:

  - type (e.g. stream)
  - dimensions (e.g. delivery)
  - values for each dimension (e.g. best effort, once, at most once ...)
- if multi-type, atomic connector compatibilities

2.4 Configuration

configuration (topology) set of specific associations between components and connectors

non-functional constraints:

- technical: technologies to use, usually non-negotiable
- business: design constraints for business reasons, usually non-negoitiable
- quality: quality attributes, usually for users

2.5 Architecture Views

logical view

- decompose the system structure into software components and connectors
- Map functionality/requirements/use cases onto the components
- concern: functionality
- audience: devs and users

process view

- model dynamic aspects of architecture
- describe how processes/threads communicate
- concern: functionality, performance
- audience: devs

development view

- static organization of the software code artifacts
- mapping between the logical view and the code is also required
- concern: reuse, portability, build
- audience: devs

physical view

- define the hardware environment (hosts, networks, storage, etc.) where the software will be deployed
- mapping b/w logical and physical also necessary
- concern: performance, availability, scalability, reliability
- audience: dev ops

2.6 Quality Attributes

performance how much work the application needs to do in a given time

- throughput: amount of work to do in unit time (transactions per second)
- response time: latency in processing a transaction (guaranteed vs average)
- deadlines: limited window to complete a transaction

scalability how well a solution works when a problem size increases, what can increase?

- request load (e.g. more users)
- simulataneous connections
- data size
- deployment

modifiability how easy it is to change the application for new funcitonality

security understanding security requirements and devising mechanisms to support them, most commonly

- authentication
- authorization
- encryption
- integrity
- non-repudiation

availability proportion of required time a system is usable

- downtime is caused by failures in applications
- recoverability is close to availability

integration ease with which application can be incorporated into broader context (data integration, providing an API)

portability how easily an application can be executed on different hardware/software than what it was developed for

- good portability comes from modularity
- will depend on libraries and platform choices

testability how easy is it to test an application

- more complex = more difficult to test

supportability how easy it is to support once deployed

- support involves diagnosing and fixing problems
- good supportability involves built-in facilities (e.g. in-depth logs)

implementability how easy it is to implement

3 Middleware Architectures

3.1 Introduction

middleware connect software components so they can use exchange info with easy-to-use mechanisms, layer of software between application and OS

- location, service discovery, replication
- protocol handling, quality of service
- sync, concurrrency, storage
- access control, authentication

examples of offerings

- app updates
- messaging and notification services
- integration brokering
- device detection
- location API
- asset transcoding
- mobile analytics
- capacity offload
- app-level security

3.2 Layers

Comes in four layers (from bottom):

- transport
- application servers
- message brokers
- business process orchestrators

3.2.1 Transport

basic pipes

- sending requests and moving data
- making communication straightforward in distributed architectures

examples:

- message-oriented middleware
- distributed OS
- SOAP

3.2.2 Application Server

on top of transport, provides:

- transaction
- security
- directory services

examples:

- .NET
- JEE
- CCM

3.2.3 Message Brokers

on top of either application server or transport, provides message processing engine:

- fast message transformation
- features for defining how to exchange and manipulates route messages between components

examples:

- Mute
- WebSphere Message Broker
- SonicMQ

3.2.4 Business Process Orchestrators

on top of message brokers, support workflow-style applications

- provide tools to describe business processes
- execute and manage intermediate states during execution

4 Architectural Analysis

4.1 Introduction

architectural analysis the activity of discovering important system properties using system's architectural models

- analyzing the architecture we have designed and modeled

4.2 Goals

completeness (internal and external)

- external: fulfill system's requirements using correct notation
- internal: fully model all elements and properly capture all design decisions

consistency ensure that different model elements do not contradict each other (internal)

- name
- interface: consistent return values, paramters ...
- behaviour: consistent behaviour of elements (e.g. 0-indexed or 1-indexed)
- interaction: consistent function calls on object (e.g. should still be able to call `remove` on empty queue)
- refinement: relationships must be maintained between high and lower level models (e.g. can't override lower level design decisions)

compatibility adheres to guidelines and constraints (external)

- the adopted style/styleguide
- reference architecture
- architectural standard

correctness architectural model fulfills system spec, and implementation fulfills model (external)

- *fulfillment* is key to correctness
- account for non-functional elements, properties

4.3 Cohesion/Coupling

Two extra goals

cohesion whether components fit cleanly with minimal overlap and extras

coupling whether components and connectors have excessive interaction

- component/connector-level
  - does each component and connector provide specific service correctly
  - does the composition of components and connectors do this
- subsystem/system-level: analyze compositions of components and connecttors to form subsystem, then complete system
  - pairwise conformance of two interacting components in terms of interface
  - over-all properties as sub-systems and system is built

- data exchanged
  - structure: data typing, organization
  - flow through system: point-point, client-server ...
  - properties: performance, security, statefulness ...
- consistency at different abstraction levels
  - refined models stay consistent with higher levels
- comparison of architectures
  - composition (of components and connectors)
  - interactions (of components and connectors)
  - characteristics of data exchange

4.4 Characteristics

We are interested in several "key concerns"

- structural (static): connectivity of components
  - lowel level components contained in higher level composite
  - points of network distribution and concurrency paths
- behavioral (static): individual component/connector functionalities
  - composite and collaborative functionalities (especially w/ off the shelf components, connectors)
- interaction (dynamic): number and type of connectors and protocols
  - timing
  - synchronicity
  - buffering
- non-functional (static/dynamic): properties across whole system
  - security
  - performance
  - quality

4.5 Levels of Formality

level of formality in analysis requires levels of formality in models used

- informal (box-line) models
  - high level analysis
  - performed manually with little automation

- semi-formal models (e.g. UML)
  - deeper level of analysis
  - requires a little training
  - partial automation
- formal models (e.g. Wright, Acme)
  - very deep analysis
  - requires good understanding of syntax + semantics used
  - better automation

4.6 Types

static analysis without executing models

- structural concerns

dynamic analysis executing/simulating models

- behaviour
- interaction
- some non-functional properties

scenario-driven analysis asserting a property for entire system

- can be static of dynamic
- very food for specific non-functional across whole system

4.7 Technique Categories

4.7.1 Inspection and Review

requires

- preparation for inspection
- preparation of participants
- review/analysis of architectural material
- anaysis of review results and recommended actions
- follow-up and closeout

- Goals: completeness, consistency, correctness, and compatibility
- Scope: spans components, connectors and the complete system; also includes data-exchange and compatibility to reference architecture and compliance to standards
- Concerns: structural, behavioral, interaction and non-functional

- Types of models: mostly semi-formal
- Types of analysis: best for static analysis and scenario-based
- Automation: manual
- Stakeholders: all stakeholders may participate

4.7.2 Model

uses system's architectural descriptions and manipulation of the model

- Goals: internal completeness, consistency, correctness
- Scope: spans components, connectors and the complete system; also includes data-exchange and compatibility to standards
- Concerns: mostly structural
- Types of models: mostly semi-formal to formal
- Types of analysis: best for static analysis of connectivity, interface ...
- Automation: partially automated
- Stakeholders: technical stakeholders

4.7.3 Simulation

software simultion of the architecture model

- Goals: completeness, consistency, correctness, and compatibility
- Scope: entire system, specific subsystem, data exchange
- Concerns: behavioral, interaction, non-functional
- Types of models: formal
- Types of analysis: dynamic, scenario-based
- Automation: mostly automated
- Stakeholders: all stakeholders may participate

5 Architecture Design

5.1 Frameworks

- n-tier client server
    - web clients → web server → application server → databases
- messaging

    - clients → queue → server

- publish-subscribe
    - publisher → topic → subscriber
- broker
    - senders → (inport) broker (outport) → reciever
- process coordinator
    - process request → process coordinator → result
        * step 1 → server 1
        * step 2 → server 2
        * . . .

5.2 Complex Frameworks

5.2.1 C2

C2 indirect invocation where independednt components communicate only through message routing connectors with rules

- no component-component links
- connector-connector links allowed
- requests go *up* in architecture
- notifications flow *down* in architecture
- no circularity

characteristics:

- state components in upper layers
- control logic in middle layers
- interface components in lower layers

5.2.2 Distributed Objects

COBRA

- objects are all separate
- procedure calls only interface to objects
- adaptor interface between object and state

5.3 Validation

increase the confidence that architecture is fit for purpose

- scenarios: come up with scenarios that test quality requirements
  - choose the quality attribute (scalability)
  - come up with a stimulus (user load doubles in 3 weeks)
  - find response (server scaled to two clusters)
- prototypes
  - proof of concept: can the architecture satisfy requirements
  - proof of technology: does the tech selected behave as expected

5.4 Finding a Design

- analogy searching: examine other fields for ideas that are analagous to the problem
- brainstorming: rapidly generate many different ideas and thoughts
- literature searching: examine current state-of-art and read about subject
- morphological charts: identify all functions, means to do function, choose means that work
- removing mental blocks: change problem to one you can solve
- control design strategy: identify and review critical decisions
- insights from reqs: find improvements to exisiting systems from reqs
- insights from implementation: use constraints on implementation to narrow down

6 Security and Trust

refer to CS458 notes

6.1 Security Models

6.1.1 CCAC

connector-centric architectural access control safeguards resources based on the connections a requesting program has

- if no connections to resource, deny
- if direct connections to resource, allow
- else accumulate the safeguards needed to get from the requestor to object and check if the requestor has necessary priveleges

6.2 Trust Management

6.2.1 Overview

trust subjective probability that an agent will do something to affect your actions

reputation expectation about entities behaviour

decentralized no central authority to coordinate and control

trust management how entities establish and maintain trust relationships

- credential and policy-based
- reputation-based

trust model describes trust information and how it

- is used to establish trust
- is combined to determine trustworthiness
- is modified in response to experiences

6.3 Threats

Ways decentralized systems can be attacked:

- impersonation
- fraudulent actions
- misrepresentation
- collusion
- additions of unknowns

Ways to combat threats:

- authentication: combat impersonation and repudiation
- separation of internal and external data: resolve conflicts
- making trust visible: shouldn't be localized to one component
- comparable trust: should be able to measure it somehow

6.4 Architectural Style

communication unit

- handles interactions with other entities
- no dependencies

information unit

- stores trust and app-specific info
- depends on interactions with entities to get info

trust unit

- computes trusworthiness and guides trust-related decisions
- depends on communication and information

application unit

- app-specific functionality
- enables local decision making
- builds on other three units

6.5 PACE

Practical Architectural approach for Composing Egocentric trust a trust-centered architecture based off of C2

6.5.1 Communication Layer

- multiple protocol handlers translate internal events into external messages and vice-versa
- communication manager creates and manages protocol handlers
- signature manager signs requests and verifies notifications

6.5.2 Information Layer

separates internal beliefs of a peer from reported info of other peers

- internal info stores internal beliefs
- external info stores messages from others

6.5.3 Trust Layer

incorporates different trust models and algorithms

- key manager generates unique public-private keys
- credential manager keeps track of peers identities
- trust manager requests public keys from peers and responds to revocations

6.5.4 Application Layer

encapsulates user-interface and app-specific components

- application trust rules encapsulates chosen rules for assigning trust and supports trust relationships

7 Design Patterns and Styles

7.1 Architecture Styles

- event-driven architecture
- pipes and filters
- multi-layered architecture
- p2p architecture

https://cs.uwaterloo.ca/~rtholmes/teaching/2015winter/cs446/slides/SE2_09-10-11_arch-styles.pdf

7.2 Design Patterns

- observer
- composite
- facade
- command
- state
- strategy
- visitor
- decorator
- proxy
- mobile code
- interpreter

https://cs.uwaterloo.ca/~rtholmes/teaching/2015winter/cs446/slides/SE2_17_design-patterns.pdf