Architectural and Detailed Design

John D. McGregor
Clemson University
Dept.of Computer Science
johnmc@cs.clemson.edu

Where are we?

- We have gone through the activities that must occur before a project can be started.
  - Feasibility
  - Planning
  - Scheduling
- And we have also considered how to capture and document the requirements.
- IMPORTANT: We understand what the problem is.

Quiz

- Consider a software system that provides on-line banking services.
  - List and describe two actors.
  - Write the main scenario, one alternative path and one exception for one use case for this system.

Analysis into Design

- A use case model has been created and validated
- We have an initial idea of what the system must do
- Can now begin to solve the problem that we assume we understand

Types of design

- Conceptual design
  - shows the customer how the system will work
  - the use cases are elaborated to provide this
- Technical design
  - architecture
    - macro-level
  - detailed component design
    - micro-level

Decomposition

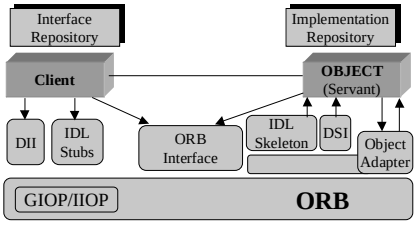
- Identify the “pieces” of the solution from a broad monolithic blob
  - modular/functional - each piece is a function
  - data-oriented - each piece is a data structure
  - event-oriented - what states the system must have and how does it move between them
  - outside-in - based on inputs from users
  - object-oriented - classes of objects that encapsulate data and behavior in one piece

Architectural design

- Did you grow up in a ranch or colonial house?
- How will the pieces be arranged?
- The architecture of a system is the structure into which the pieces are placed.
- Logical structure
  - constrained by quality attributes
- Physical structure
  - constrained by operational environment

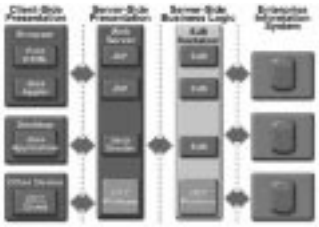
Architectural design and standards

- Common Object Request Broker Architecture - CORBA



J2EE Architecture

- **A complete set of components is available.**



Architectural styles

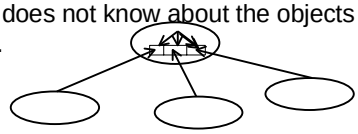
- Architects use a number of techniques
  - personal experience
  - standards
  - best current practice
- Experience is exchanged by recognizing patterns and defining styles
  - a style includes the suggested structure and describes how that style affects system qualities

Pipe and Filter Style

- Think about how a compiler works
  - lexical analysis, parsing, semantic analysis,code generation
- System is conceived as output from one phase is input to next phase
- Useful for organizing a set of reusable filters to perform a well-understood task
- Not good for GUI/event-driven situations
- Does not address concurrency

Implicit Invocation

- Objects register for events such as mouse presses and releases and key presses
- The relationships are uni-directional. An object that registers knows about the events that will be created but the object creating the events does not know about the objects registering.



Client/server style

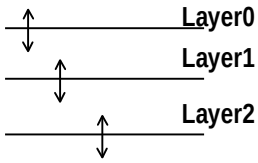
- A component provides a set of services that can be used by multiple clients
- The clients can be located anywhere relative to the server
- Became a popular style for applications that present a form on the screen and store the result in a database & query later
- CORBA is an example of this basic style

Client/server style

- Good for distributing processes on multiple machines and giving multiple processes access to a single resource such as a database
- May increase through-put but may not be good for high-performance apps
- Flexible, may have multiple servers serving multiple clients depending upon load

Layering mechanism

- Separates concerns and responsibilities
- Layer 0 knows about layer 1 but not layer 2



4 tier architecture

- 4 Layers
  - user interface/thin client
  - presentation layer/rest of the client - web server
  - business logic - app server
  - database
- Uses four processes; may replicate on any number of machines; load balance dynamically

4 tier architecture

- Good through-put
- Very modular
  - ease of replacement
  - ease of maintenance
- more complexity
- many technologies must work and work together

Architectural qualities

- Performance
- Security
- Availability
- Functionality
- Usability
- Modifiability
- Portability
- Scalability
- Reusability
- Integrability
- Testability
- Conceptual integrity
- Buildability
- Correctness
- Completeness

Qualities

- Performance - the speed with which software performs its function
- Security - the ability of the software to control access based on personal permissions
- Availability - how likely is the software to be ready to perform when requested
- Functionality - the completeness of the functionality provided by the system

Qualities

- Usability - the anticipated actors against the system will be able to use the system
- Modifiability - the ease with which needed changes can be made to the system
- Portability - the ease with which the system can be used on a platform for which it was not originally intended
- Scalability - the ability of the system to be made to handle more load than planned

Qualities

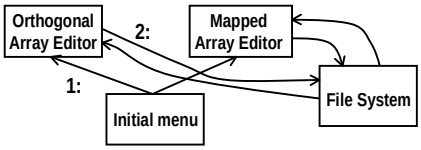
- Reusability - the ease with which portions of the system can be used in other systems
- Integrability - the ease with which the system can be made to work with other systems
- Testability - the ease with which defects in the system can be identified
- Conceptual integrity - the system is a faithful representation of the domain

Qualities

- Buildability - the ease with which the system can be built
- Correctness - an accurate solution to the problem
- Completeness- solution covers all requirements

Representation

- Architectures are usually represented as components and connectors
- There are special tools such as Acme, or
- UML collaboration diagrams can be used



Architecture Evaluation

- Guided Inspection is the basic technique
- Test cases come from the use case scenarios
- The test cases must be written at the same high level as the architecture
- Completeness, correctness and consistency are also judged at the high level

Detailed Design

- We now have a basic structure
- Need modules to fill in the component places in the architecture
- We will use objects as the modules
- Objects come from classes

Detailed Design Principles

- Modularity - pieces are better than a single monolith
- Abstraction - layers of module definitions add/remove detail
- Information hiding - the external world knows what a module does but not how it does it
- Encapsulation - a set of related pieces are grouped within a module

Different pieces require different approaches

- GUI design is very different from functionality design
- GUI focuses on usability while functionality design focuses on performance and correctness
- Internationalization
  - language/fonts/direction
  - numeric formats
  - symbols for currency
  - spelling

Concurrency

- Different models
  - sequential - one action, one process, one processor - still seen at the application level
  - parallel - multiple actions, multiple processes, multiple processors - special hardware
  - concurrent - multiple actions, one or more processes but multiple threads, one processor - GUI and many other areas
  - UML activity diagram can represent

Shared resources

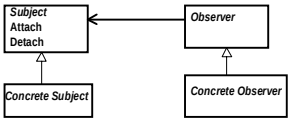
- Any resource that is visible to multiple threads must be protected from interleaved access
- A program must be able to place all its data to be printed into the print queue without some data from another program placing some of its data in the queue - synchronization
- Use monitor in the program or guardian outside the program

Design Pattern

- This is a way to capture experience and share it with less experienced designers
- A pattern is expected to be “best current practice within a context”
- Patterns may even be grouped into pattern languages

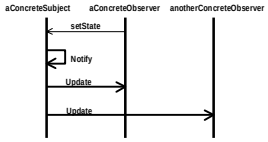
Design Pattern - Observer

- Motivation - want to maintain consistency among a set of objects without tight coupling
- Applicability - use this pattern when a change in one object implies changes in other objects
- Structure



Design Pattern - Observer

- Participants
  - Subject
  - Observer
- Collaborations



Design Pattern - Observer

- Consequences
- Implementation
- Known Uses
- Related Patterns

Good Design - Components are independent

- Coupling - a bad thing
  - A component depends upon another component for state or behavior
  - Component A aggregates instances of components B, C, D
  - Component A is passed an instance of component E as a parameter
  - Where there is an A, there must be B, C, D, E
  - Harder to replace/modify A

Good Design - Components are independent

- Cohesion - a good thing
  - Everything in A is needed to represent the concept
  - Each method in A is a behavior of A
  - Each attribute in A is part of the state of A
  - Component A should:
    - return a string representation of itself
    - take a stream as a parameter and print itself to the stream

Good Design - Exception Handling

- Error return codes vs an exceptional route
- public float average(){…} where returning minFloat signals an error
- code segment

```
grade = average();
if(grade != minFloat){
   …
}else ...
```

Good Design - Exception Handling

- public float average() throws DivideByZero
- code segment

```
try{
   grade = average();
   …
}catch(DivideByZero dbz){
   “handle dbz”
}
```


Good design - fault tolerance

- Takes a licking and keeps on ticking
- Failure - visible problem
- Fault - the human error that causes a failure
- Fault detection - voting
- Fault correction - transactions
- Fault tolerance - redundancy

Improving design - reduce complexity

- Reduce the number of interactions
- Reduce the number of paths among components

Improving design - Design by Contract

- For each behavior
  - pre-condition - assumptions about what must be true before the method can be used
  - post-condition - guarantee of service if pre-condition was true
- For each component
  - invariant - a statement about the component’s state that is always true

Improving design - Design by Contract

NumericSet::average()

pre: NumericSet::numberOfElements > 0

post: sum(forall e:NumericSet::isMemberOf(e))/
NumericSet::numberOfElements

Class NumericSet

inv: NumericSet::numberOfElements >= 0

Improving design - Defensive Design

NumericSet::average()

pre: none

post: if (NumericSet::numberOfElements > 0)

sum(forall e:NumericSet::isMemberOf(e))/
NumericSet::numberOfElements

else

throw DivideByZero

Object Constraint Language (OCL)

- UML constraint language
  - http://www.fast.de/Projekte/forsoft/ocl/2_OCL_Examples.html
  - http://www.csci.csusb.edu/dick/samples/ocl.html
  - http://dec.bmth.ac.uk/staff/swebster/UML_OCL/sld001.htm

Improving design - Prototyping

- Throw away prototype versus evolutionary development
- No error checking
- Aids in developing a better design by exploring an algorithm before doing the final implementation

Making design decisions

- Identify criteria
- Determine priorities
- Identify alternatives

	Priority	HTML	Postscript	Roll your own
cost	3	1	2	3
schedule	2			
modifiability	1			

Evaluating Designs

- Metrics
  - Cyclomatic complexity
- Review/Inspection
  - See chapter 4 of McGregor/Sykes via web site

Conclusion

- Architectural design provides the overall structure for an application/system.
- Detailed design fills in the components that populate the architectural structure.
- Designs follow patterns that can be used to train new designers.
- Designs can be specified using the full range of UML diagrams.