CITS4401 Software Requirements and Design
Design decisions and constraints

Lecturer: Arran Stewart

Major design decisions

Re-cap

We mentioned earlier (in the first design lecture) several major aspects of design which most systems must address:

- Concurrency
  - Does the system have to operate concurrently? If so, how should this be organised?
- Hardware/software mapping
  - Which subsystems are made/bought/re-used
- Persistent data management
  - How is persistent data to be stored?
- Access control
  - Who can access what?
- Software control
  - Do events have to happen in a particular order? If so, what is it?
- Boundary conditions
  - How are system startup, shutdown and failure handled?

Concurrency

Concurrency

- Often, we will want multiple system components to execute at the same time
- This gives rise to potential problems:
  - How can data data shared among concurrently executing components be kepts consistent?
  - How can we ensure that one action does not interfere with another?

Terminology

- *Processes* are operating system tasks, each of which has their own set of open files, global data, etc
- *Threads* are threads of execution *within* a process – they usually share global data, but have their own run-time stack

Possible solutions

Design for concurrency is a major area all on its own, but a few possible solutions to the problems of keeping consistency/avoiding interference:

Locks (also “semaphores”, “mutexes”) - can be used to limit access to shared resources

- Critical section: piece of code which accesses a shared state that must not be concurrently accessed by another thread
    - must be kept as small as possible, should not contain loops
- Problems: easy to get wrong; can take too many locks, too few, wrong locks, or in wrong order; difficult to recover from errors
- Deadlock: two threads are bl

Possible solutions

Software transactional memory - Optimistically *try* transactions, on failure, re-try when things may have changed.

- Easier to handle errors than locks
- No locks, so can't deadlock
- For each task in a transaction, must keep track of how to *undo* it

Possible solutions

No shared resources ("actor-based concurrency") - Some languages, like Erlang, are designed around the idea that threads have *no* shared resources – all interaction between threads is by sending *messages*.

- Not as familiar to programmers as locks
- Usually easier than lock-based

Solutions

For a gentle-ish explanation:

Locks, Actors, And Stm In Pictures

UML

- Threads:
  - A thread of control is a path through a set of state diagrams on which a single object is active at a time.
  - A thread remains within a state diagram until an object sends an event to another object and waits for another event
  - Thread splitting: Object does a non-blocking send of an event.
- The task here is to identify concurrent threads and address concurrency issues.
  - If an object is an aggregation of other objects, it is possible to have concurrent state machines.
- Design goal: response time, performance.

Concurrency questions

- Which objects of the object model are independent?
- Does the system provide access to multiple users?
- What kind of concurrency control is relevant?
  - Pessimistic concurrency control (with locking)
  - Optimistic concurrency control (without locking)
- Can a single request to the system be decomposed into multiple requests? Can these requests be handled in parallel?

Implementing concurrency

- Concurrent systems can be implemented on any system that provides
    - physical concurrency (hardware), or
    - logical concurrency (software)

Hardware software mapping

Hardware software mapping

- This activity addresses two questions:
  - How shall we realize the subsystems: Hardware or Software?
  - How is the object model mapped on the chosen hardware & software?
    - Mapping Objects onto Reality: Processor, Memory, Input/Output
    - Mapping Associations onto Reality: Connectivity
- Much of the difficulty of designing a system comes from meeting externally-imposed hardware and software constraints.
  - Certain tasks have to be at specific locations

Mapping the objects

- Processor issues:
  - Is the computation rate too demanding for a single processor?
  - Can we get a speedup by distributing tasks across several processors?
  - How many processors are required to maintain steady state load?
- Memory issues:
  - Is there enough memory to buffer bursts of requests?
- I/O issues:
  - Do you need an extra piece of hardware to handle the data generation rate?
  - Does the response time exceed the available communication bandwidth between subsystems or a task and a piece of hardware?

Mapping the subsystems associations: connectivity

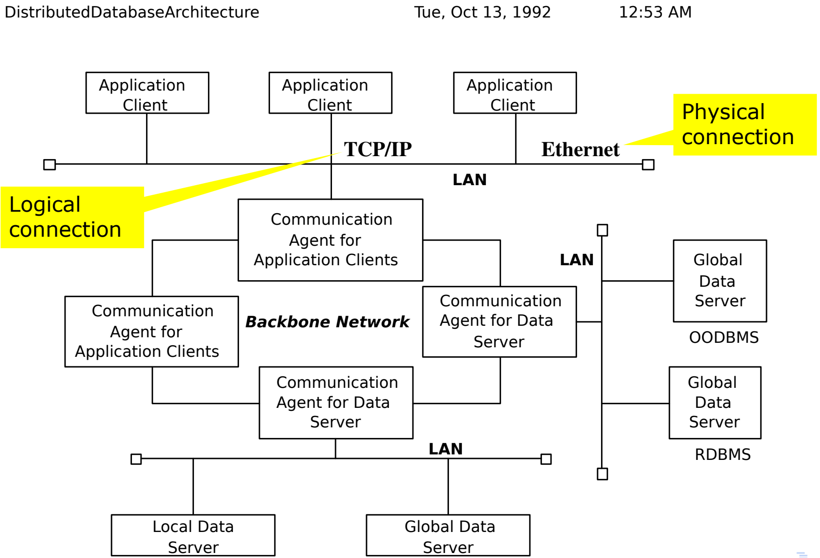
- Describe the physical connectivity of the hardware
  - Often the physical layer in ISO's OSI Reference Model
    - Which associations in the object model are mapped to physical connections?
    - Which of the client-supplier relationships in the analysis/design model correspond to physical connections?
- Describe the logical connectivity (subsystem associations)
  - Identify associations that do not directly map into physical connections:
  - How should these associations be implemented?

ISO – International Standards Organization
OSI – Open Systems Interconnection

Connectivity in distributed systems

- If the architecture is distributed, we need to describe the network architecture (communication subsystem) as well.
- Questions to ask
  - What are the transmission media? (Ethernet, Wireless)
  - What is the Quality of Service (QoS)? What kind of communication protocols can be used?
  - Should the interaction be asynchronous, synchronous or blocking?
  - What are the available bandwidth requirements between the subsystems?

A physical connectivity drawing



Hardware/software mapping questions

- What is the connectivity among physical units?
  - Tree, star, matrix, ring
- What is the appropriate communication protocol between the subsystems?
  - Function of required bandwidth, latency and desired reliability
- Is certain functionality already available in hardware?
- Do certain tasks require specific locations to control the hardware or to permit concurrent operation?
  - Often true for embedded systems
- General system performance question:
  - What is the desired response time?

Data management

Data management

- Some objects in the models need to be persistent
- A persistent object can be realized with one of the following mechanisms
  - Flat files
    - Cheap, simple, permanent storage
    - Low level (Read, Write)
    - Applications must add code to provide suitable level of abstraction
  - Relational database
    - Powerful, easy to port
    - Supports multiple writers and readers
    - Mapping complex object models to relational database is challenging
  - Object-oriented database
    - Provides services similar to relational database
    - Stores data as objects and associations;
    - OODBs are slower than relational DBs for typical queries

Flat files or database?

- When should you choose flat files for data storage?
  - Are the data voluminous (bit maps)?
  - Do you have lots of raw data (core dump, event trace)?
  - Do you need to keep the data only for a short time?
  - Is the information density low (archival files, history logs)?
- When should you choose a (relational or OO) database?
  - Do the data require access at fine levels of details by multiple users?
  - Must the data be ported across multiple platforms (heterogeneous systems)?
  - Do multiple application programs access the data?
  - Does the data management require a lot of infrastructure?

Object-oriented databases

- Support all fundamental object modeling concepts
  - Classes, Attributes, Methods, Associations, Inheritance
- Mapping an object model to an OO-database
  - Determine which objects are persistent.
  - Perform normal requirement analysis and object design
  - Create single attribute indices to reduce performance bottlenecks
  - Do the mapping (specific to commercially available product). Example:
    - In ObjectStore, implement classes and associations by preparing C++ declarations for each class and each association in the object model

Relational databases

- Based on relational algebra
- Data is presented as 2-dimensional tables. Tables have a specific number of columns and arbitrary numbers of rows
    - Primary key: Combination of attributes that uniquely identify a row in a table. Each table should have only one primary key
    - Foreign key: Reference to a primary key in another table
- SQL is the standard language for defining and manipulating tables.

Access control

Access control

- Describes access rights for different classes of actors
- Describes how object guard against unauthorized access

Access control questions

- Does the system need authentication?
- If yes, what is the authentication scheme?
    - User name and password? Access control list
    - Tickets? Capability-based
- What is the user interface for authentication?
- Does the system need a network-wide name server?
- How is a service known to the rest of the system?
    - At runtime? At compile time?
    - By communication port number?
    - By host name?

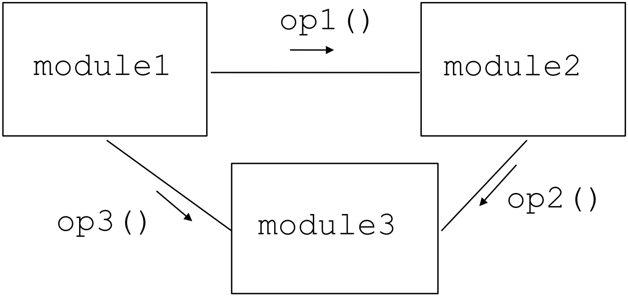
5. Decide on software control

- Control flow is the sequence of actions in a system. It gives the order in which things can happen in the system.
- Deciding this depends on whether the things can happen
  - fairly independently and in parallel (threads/tasks) or
  - only in sequence in a given order (procedural) or
  - activities one at a time with their order determined by external events (event driven)

Guidelines for choosing control flow

- activities must occur in a fixed order with little time overlaps between activities
  $\rightarrow$ choose procedural control
- activities may occur in different orders, as determined by external requests, but usually one activity at a time
  $\rightarrow$ choose event driven control (+ central controller)
- activities are largely independent and can be time overlapped
  $\rightarrow$ choose threads

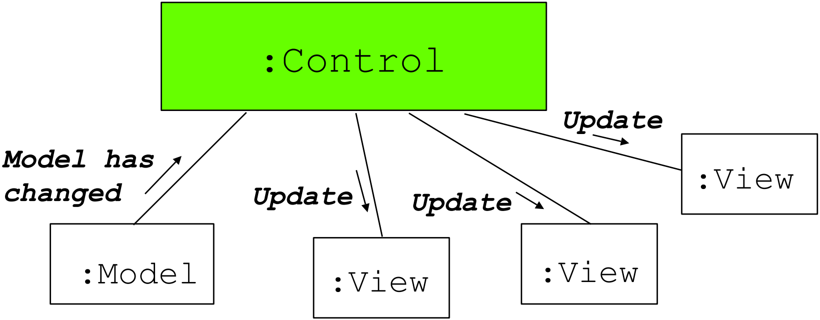
Procedure-driven control example



op1(), op2(), and op3() are procedure calls

Event-based system example: MVC

- Smalltalk-80 Model-View-Controller
- Client/Server Architecture



Centralized vs. decentralized designs

- Should you use a centralized or decentralized design?
- Centralized Design
  - One control object or subsystem (“spider”) controls everything
  - Change in the control structure is very easy
  - Possible performance bottleneck
- Decentralized Design
  - Control is distributed
  - Spreads out responsibility
  - Fits nicely into object-oriented development

Boundary conditions

Boundary conditions

- Most of the system design effort is concerned with steady-state behavior.
- However, the system design phase must also address the initiation and finalization of the system.
    - initialisation
    - termination
    - failure

Boundary questions

- Initialization
  - Describes how the system is brought from an non initialized state to steady-state (“startup use cases”).
- Termination
  - Describes what resources are cleaned up and which systems are notified upon termination (“termination use cases”).
- Failure
  - Many possible causes: Bugs, errors, external problems (power supply).
  - Good system design foresees fatal failures (“failure use cases”).

Boundary questions (cont.)

- Initialization
  - How does the system start up?
    - What data need to be accessed at startup time?
    - What services have to be registered?
  - What does the user interface do at start up time?
    - How does it present itself to the user?
- Termination
  - Are single subsystems allowed to terminate?
  - Are other subsystems notified if a single subsystem terminates?
  - How are local updates communicated to the database?
- Failure
  - How does the system behave when a node or communication link fails? Are there backup communication links?
  - How does the system recover from failure? Is this different from initialization?

Summary

We reviewed major activities in system design:

- Concurrency identification
- Hardware/Software mapping
- Persistent data management
- Access Control
- Software control selection
- Boundary conditions

Each of these activities revises the subsystem decomposition to address a specific issue. Once these activities are completed, the interface of the subsystems can be defined.

Recommended reading

- Bruegge & Dutoit, 2010:
  - §7.4 System Design Alternatives
    - §7.4.1 Mapping Subsystems to Processors and Components
    - §7.4.2 Identifying and Storing Persistent Data
    - §7.4.3 Providing Access Control
    - §7.4.4 Designing the Global Control Flow
    - §7.4.5 Identifying Boundary Conditions

Design constraints

Design Goals

- When we move from Requirements Analysis into System Design, we should ensure that we have identified the design goals for our system
- Many design goals can be inferred from the non-functional requirements or the application domain. Others should be checked with the client.
- Design Goals need to be stated explicitly so that future design criteria can be made consistently, following the same set of criteria

Types of Design Goal

- There are many desirable qualities which may be design goals for your system:
  - performance
  - dependability
  - cost
  - maintenance
  - end user criteria
- Meeting some of these goals may conflict with meeting others - can you think of an example of conflicting goals ?

Design Goal Example

- Classify each design goal below according to performance, dependability, cost, maintenance, end user criteria
  - Users must be given feedback within 1 sec of issuing a command
  - The TicketDistributor must be able to issue train tickets even in the event of a network failure
  - The housing of the TicketDistributor must allow for new buttons to be installed if the number of different fares increases
  - The AutomatedTellerMachine must withstand dictionary attacks (i.e. ID numbers discovered by systematic trial)
  - The user interfaces of the system should prevent users from issuing commands in the wrong order

Design Goals come from requirements

- A functional requirement describes a system service or function.
- A non-functional requirement is a constraint placed on the system or on the development process
- Check lists are useful for identifying non-functional requirements

Type of Non-functional Requirements

- User interface and human factors
- Documentation
- Hardware considerations
- Performance characteristics
- Error handling and extreme conditions
- System interfacing
- Quality issues
- System modifications
- Physical environment
- Security issues
- Resources and management issues

Non-Functional Requirements Trigger Questions (1)

- User interface and human factors
  - What type of user will be using the system?
  - Will more than one type of user be using the system?
  - What sort of training will be required for each type of user?
  - Is it particularly important that the system be easy to learn?
  - Is it particularly important that users be protected from making errors?
  - What sort of input/output devices for the human interface are available, and what are their characteristics?

Non-Functional Requirements Trigger Questions (2)

- Documentation
  - What kind of documentation is required?
  - What audience is to be addressed by each document?
- Hardware considerations
  - What hardware is the proposed system to be used on?
  - What are the characteristics of the target hardware, including memory size and auxiliary storage space?

Non-Functional Requirements Trigger Questions (3)

- Performance characteristics
  - Are there any speed, throughput, or response time constraints on the system?
  - Are there size or capacity constraints on the data to be processed by the system?
- Error handling and extreme conditions
  - How should the system respond to input errors?
  - How should the system respond to extreme conditions?

Non-Functional Requirements Trigger Questions (4)

- System interfacing
  - Is input coming from systems outside the proposed system?
  - Is output going to systems outside the proposed system?
  - Are there restrictions on the format or medium that must be used for input or output?
- Quality issues
  - What are the requirements for reliability?
  - Must the system trap faults?
  - Is there a maximum acceptable time for restarting the system after a failure?
  - What is the acceptable system downtime per 24-hour period?
  - Is it important that the system be portable (able to move to different hardware or operating system environments)?

Non-Functional Requirements Trigger Questions (5)

- System Modifications
  - What parts of the system are likely candidates for later modification?
  - What sorts of modifications are expected?
- Physical Environment
  - Where will the target equipment operate?
  - Will the target equipment be in one or several locations?
  - Will the environmental conditions in any way be out of the ordinary (for example, unusual temperatures, vibrations, magnetic fields, . . .)?

Non-Functional Requirements Trigger Questions (7)

- Security Issues
  - Must access to any data or the system itself be controlled?
  - Is physical security an issue?
- Resources and Management Issues
  - How often will the system be backed up?
  - Who will be responsible for the back up?
  - Who is responsible for system installation?
  - Who will be responsible for system maintenance?

Non-Functional (Pseudo) Requirements

- Non-functional (Pseudo) requirement:
  - Any client restriction on the solution domain
- Examples:
  - The target platform must be an Android phone
  - The implementation language must be Java
  - The documentation standard X must be used
  - A data-glove must be used
  - Direct3D must be used
  - The system must interface to a barcode reader

Evaluating Designs

- When is a design correct?
  - If it can be shown to capture all the functions of the requirements document
  - If it captures all the users' requirements
- What makes a design a good design?
  - It is correct, complete, consistent, realistic and readable

Some Evaluation Criteria

- product vs process
- differing views: client, developer, user
- design goals (from non-functional requirements)
- cohesion and coupling in subsystems
- comparing designs: evaluation matrix
- rationale

Modular design

- A design is modular when
  - each activity of the system is performed by exactly one component
  - inputs and outputs of each component are well-defined, in that every input and output is necessary for the function of that component
  - the idea is to minimise the impact of later changes by abstracting from implementation details

Correct Designs

- Does the design correctly capture the requirements?
- Are the requirements the right ones?
- These questions can be addressed by:
  - testing the design against both the requirements document and against user expectations.
  - analysing the requirements for completeness, consistency, realism
  - design review meetings
  - formal proof that design model D satisfies requirements model R

Correct OO Designs

- Can every subsystem be traced back to a use case or nonfunctional requirement?
- Can every use case be mapped to a set of subsystems?
- Can every design goal be traced back to a nonfunctional requirement?
- Is every nonfunctional requirement addressed in the system design model?
- Does each actor have an access policy: what data and functionality is available to each actor?
- Is the access policy consistent with the nonfunctional security requirement?

Complete OO Designs

- Has every requirement and every system design issue been addressed?
- Have the boundary conditions been handled?
- Was there a walkthrough of the use cases to identify missing functionality in the system design?
- Have all use cases been examined and assigned a control object?
- Have all aspects of system design been addressed?
- Are all subsystems well-defined?

Consistent OO Designs

- Does the design contain any contradictions?
- Are conflicting design goals prioritized?
- Are there design goals that violate a nonfunctional requirement?
- Are there multiple subsystems or classes with the same name?
- Are collections of objects exchanged among subsystems in a consistent manner?

Realistic OO Designs

- Can the design be implemented?
- Are there any new technologies or components in the system?
- Have the appropriateness and robustness of these technologies been investigated?
- Have performance and reliability requirements been reviewed in the context of the subsystem decomposition?

Concurrency Issues

- Contention: 2 processes competing for access to the same resource
  - e.g. writing to a network bus such as the CANbus
- Deadlock: 2 processes are waiting for each other and therefore can make no progress
  - e.g. the dining philosophers each holding one fork
- Mutual exclusion: a resource must only be accessed by one processes at a time
  - e.g. crediting and debiting a bank account

Readable OO Designs

- Can developers not involved in the system design understand the model?
- Are subsystem names understandable?
- Do entities with similar names denote similar phenomena?
- Are all entities described at the same level of detail?

Design Evaluation Matrix: a tool for comparing different designs

- Characteristics for comparison include:
  - easy to change algorithm
  - easy to change data
  - easy to change function
  - good performance
  - ease of reuse
  - modularity, testability, maintainability, efficiency,
  - ease of understanding, ease of modification, consistency

Comparing Designs - Measures

- We can compare two different designs by
  - identifying a list of relevant design characteristics c0 to cn and (optionally) a weight w0 to wn for each
  - checking for each design characteristic whether the given design exhibits it or not: ei = 0 or ei = 1
  - Quality = e0 * w0 + e1 * w1 + ... + en * wn
- Suitable characteristics include: modularity, testability, maintainability, efficiency, ease of understanding/modification, consistency ...

Design Evaluation Matrix Example

Design Characteristic	*Weight*	*Design 1*	*Design 2*
Portability	5	1	0
Easy to use & robust	2	1	1
Response time	1	0	1
TOTAL	8 max	7	3

Now you try one

- List up to 4 characteristics you would use in a design evaluation matrix for an automatic bank teller system
- Identify weights for each characteristic giving reasons for your choices
- What information do you need to evaluate each characteristic?

Recommended reading

- Bruegge & Dutoit, 2010:
  - §4.4.7 identifying non-functional requirements
  - §6.4.2 identifying design goals