N(=3)-Tiered Systems

3-Tiered Systems

Two tiers in a client/server architecture

Client

Presentation logic

Middle-Tier Server

Business rules

Database Server

Data

System Architecture Choices

- **Monolithic**
  - 1 large program, imports/exports data

- **Client/Server**
  - collection of clients, updates database
  - “fat client”

- **3-tiered**
  - collection of clients, 1 mid-tier process for “business rules”
  - “thin client”

Example Business Rule

- pay = hours_worked * pay_rate

- In a client/server architecture:
  - Prompt the user for employee_number & hours_worked
  - Fetch pay_rate from db
    - select pay_rate from pay_table where employee_id = <id>
  - Calculate the pay for the employee
  - Generate and execute an SQL statement to update the db
    - update payroll
      - set pay = <calculated_pay>
      - where employee_id = <id>
Basic Problems with this Approach

- Want to change the db as little as possible.
  - the most fragile component

- DB is not a great execution engine
  - inefficient
  - limited choice of language
  - hard to interact with outside services
  - poor development environment
  - poor error recovery

- Vendor lock-in

Alterately

- A database stored procedure could be used to compute the pay.
  - e.g., Oracle PL/SQL
  - Java extension to db

- Clients could then concentrate exclusively on presentation.

- Single database would have to be changed, re-tested & migrated.

Architectural Problems

- Client-resident business rules
  - client bloat + lack of scalability on client machines
  - need to address lowest common denominator machine
    - 386 with 16M
  - transactions involving more than just db (e.g., queues)
    - must configure all client machines!

- DB-resident business rules
  - db bloat (too much for the db to do – runs out of steam)

- Common Issues
  - large # db connections
  - lack of support for caching
  - wide-area data distribution (data partitioning strategy)
  - fault tolerance
Some Industry Statistics

- 2/3 of respondents had a formal system architecture
  - Monolithic
    - 14%
  - client/server
    - 26%
  - n-tier client/server
    - 54%
  - web centric
    - 3%

- Source
  - Cutter Consortium
    - Jan, 1999
    - survey of Fortune 1000 internal IT projects
  - “Client-server in general, and n-tier client-server in particular, gives IT the flexibility to deploy available computing resources most effectively.”

Legacy Issues

- In large corporations, different departments develop their own client/server systems
- Inevitable in the case of mergers and acquisitions

Solution

- Add a middle tier to isolate clients from databases.
- Re-engineer the databases going forward.

Case Study

- Source:
  - AMIA (American Medical Informatics Association) 1998 Conference
  - “A Software Architecture to Support a Large-Scale, Multi-Tier Clinical Information System”
    - J.A. Yungton, D.F. Sidig, J. Pappas, S. Flammini, H.C. Chueh, and J.M. Teich
  - Partners HealthCare System
    - Merger of two Boston-area hospitals
      - Brigham and Women's Hospital
      - Massachusetts General Hospital
  - Clinical Information System
    - patient health records
    - tests and results
    - ...
  - Each hospital had its own HOMEGROWN system
    - decision was made to merge the systems
    - neither was superior to the other
    - each system had its strengths
Case Study

• Major requirements
  – Ease of software distribution/installation
    • 20,000+ workstations in the network
  – A solid data access tier
    • software services
    • data access routines
    • reusable modules to
      – minimize duplication of effort
      – maximize application interoperability
  – Intuitive, consistent, clinical computing environment
    • diverse end-user population
    • distributed client development
      – “In the absence of a unifying force, applications would take on their own
        look and feel leaving end-users to sort out a myriad of different styles
        and functionalities”

Case Study – Software Distribution

• Hybrid approach
  – Client maintains local program cache
    • executables, support files, shared libraries
  – On each execution, cache checked against server to ensure most
    recent updates are installed.
  – “Launcher” installed on each client
    • “Version Console” resides on a network server
      – front-end to version control database
    • Uses “pull” (”client pull“)
  – 2 key features
    • defines projects = collection of files
      – project dependencies
      – project + dependents bundled on-the-fly as a “release”
    • workstation types
      – architecture
      – class: alpha test, beta test, production

Case Study – Software Distribution

• Options
  – network architecture
    • applications resident on servers
      – pro: applications always up-to-date
      – con: excessive load on servers for menial tasks
  – client-server architecture
    • local executables
      – pro: frees server from download and execution
      – con: program and patch distribution
        » initial distribution: Micorosoft Systems Management Server
        » update distributions: ?
        » uses “push” on reboot, therefore stale client potential

Case Study – Data Access Tier

• Faced with challenge of enterprise-wide data consistency and data
  access
  – no existing common denominator
  – inevitable that additional systems would need to be integrated
  – corporate strategy:
    • add an abstract “data access” tier
      – provides common data objects & services to client applications while
        hiding the details of disparate back-end systems
  • Technology
    • Microsoft COM
      • robust, easy to use, relatively fast
      • allows application development to proceed in parallel with middle-tier
        development
  • Location
    – could reside anywhere
    – chose to distribute data access servers to client workstations
      • better performance
Case Study – Data Access Tier

- Analyzed to identify key objects and services
  - PatientObject
  - UserObject, UserSecurity
  - OrderEntry-based objects:
    - Order, Test, Medication, …
  - Service-based objects:
    - PatientLookup, Observation, Procedure, Therapy, …
  - Results-based objects
  - PCISClientManager
    - MGH data stored on Tandem Nonstop SQL

Case Study – Data Access Tier

- Client-to-data access tier communications
  - callable well-defined interface
    - names of callable routines
    - parameters
      - set in stone
      - modifications require justifications and approvals
    - returning well-known objects
      - heavily documented online
    - objects can be plugged into applications
  - proven system agility
    - built web-based clinical info viewer
    - built web-based phone directory
    - longitudinal medical record application
    - back-end redirected to first look into a data cache before attempting a retrieval

Case Study – Data Access Tier

- client-to-client communications
  - e.g.,
    - PatientObject can be passed from one application to another.
    - UserSecurity object can be passed

- Security
  - with servers resident on clients,
    - e.g., can use Excel/VB to interface to COM objects such as PatientLookup.
  - sol’n:
    - db of authorized applications
    - launched applications receive an ALK (application launch key)
    - using ALK, will get an SLK that must match the local server’s SLK, or server will not respond.

Case Study - Application Framework

- Clinical Application Suite
  - a framework used to house applications
    - merges multiple clinical applications into a single visual a functional context
    - maintains a single CurrentPatient and CurrentUser object across all applications
    - consolidates common system services
      - e.g., only one connection to PatientLookup objects
      - one GUI for displaying patient fields
    - button bars along top and down sides
      - launch apps and switch between them
    - because of its persistence on the screen, CAS provides a constant point of reference for the user
    - app builders code to the CAS API