Reflections on the Platform Approach

Mendel Rosenblum
mendel@cs.stanford.edu
Talk outline

- What do I meant by the platform approach?
  - Got a problem, the solution must involve a platform

- Ideal platforms and the challenges of platform design
  - Platform should provide functionality to clients but not constrain them

- Platform approach in action
  - The problem: enterprise software sucks
  - Build a platform for enterprise SaaS applications

- A platform for building enterprise applications and some observations
  - Disclaimer: Names have been changed to protect the innocent
Ideal Platform

- Solutions to hard/complex problems common to many clients
- Useful functionality - Ease clients building/running
- Delight client developers: more productive/correct/robust/...

Example: Platform for Software as a Service (SaaS)
- Handle scale, fault tolerance, geographic distribution, monitoring, debugging, response time, security practices, etc.
Platform design

- Easier when client needs are well understood:

- Risky when clients are speculative and not well characterized:
Platform design tradeoffs

- The more a platform can do for clients, the higher value it offers the clients
  - Encourages the platform design to include more

- Web app framework platform pejorative adjective: opinionated
  - Obstinate or conceited with regard to the merit of one’s own opinions; conceitedly dogmatic
    - Platform functionality that doesn't work or isn't preferred by the client

- Platform design should offer high client value but not be opinionated
The Problem of Interest

- Enterprise applications suck!
  - Inflicting pain during the large fraction of our life spent at work

- Painfully bad user interfaces on enterprise apps
  - Harmful to user productivity - Month long training courses to use

- Applications become gatekeepers of the enterprise’s information
  - Information available only through the bad application

- The Solution (conceived in late 2012):
  - Figure out user-centered enterprise app design
  - Platform approach: A killer SaaS platform for these new enterprise apps
Idea for a SaaS platform for enterprise application

- Capture information about the enterprise using objects
  - Capture all enterprise information in a common object repository
  - Employ ideas from object modelling and simulation research
- Free the information from the grip of applications
- Applications with delightful user interfaces
  - Apps are portals for creating and accessing the enterprise’s objects
  - All apps have the same, consistent view of the enterprise’s info
    - Supports tight cooperation
Designing an object repository and SaaS platform

- Enforce separation of applications and objects

  Focus on the object repository:
  - Objects capture complete history of enterprise’s information
  - Objects annotated with who and why the changes were made
  - Access control for objects done by the objects themselves (not applications)
  - Fast: Low-latency (in memory) object repository

- Platform for building applications on the object repository
  - Leverage and extend industry SaaS platform: AngularJS and NodeJS
    - Agile software development influence
  - Extend NodeJS to access and curate the objects
    - Persistent object JavaScript runtime accessing the object repository
    - Nice programming environment (familiar language, transactions, transparent access)
Persistent object JavaScript runtime

- **Object type system with objects having types (class) with inheritance**
  - Capture a model of the enterprise in objects (e.g. Employee, Job Position, Job Application,...)
  - Stores a complete history of changes including annotating metadata
    - Capture the who and the why

- **Access the objects as big JavaScript data structure**
  - Transparent, snapshot consistent reads at any point of time (i.e. time travel)
  - Object mutation using transactions (ACID properties)
  - Per-object identity-based access control enforcement
    - Access Control List (ACL) per-object enforced by runtime

- **Query system supporting lookups based on object content and ACLs**
Object Modeling Infrastructure Classes

- **People - Persona**
  - Identifies a person wearing a particular hat
  - Used in access control, ownership, relations, etc.

- **Action - Activity**
  - Notion of why something is being done
  - Used both to track changes and define an API of available operations

- **Coordination - Activity Plan**
  - Coordinate people working together to accomplish a common goal
  - Define roles and actions
Activity: Platform's interface to front-end

- **Idea**: collect annotations by making all actions specify the higher level intent
  - Activity - Capture the who (Persona) and why (Activity class)

- **Allows user to transactionally view and mutant the persistent objects**
  - Start an Activity (e.g. `Add_JobApplication_To_JobPosition`)
  - Activity delivers UI view information useful for action (MVC model data)
    - Activity's InterfaceModel
      - Allow applications to define external abstraction build from modeling objects
      - Snapshot consistent reads of the objects
      - Live updating to keep application’s view fresh
  - Activity saves draft of work done by the user
  - User triggers `activity.perform()` to atomically commit changes to persistent objects
Platform design and development

- **Platform approach lesson #0**: Need clients to drive design
  - Design of process for delightful user-centered enterprise applications
  - Platform features

- Choose an app in the Hiring Area: Applicant tracking system
  - No dominant player or industry standard practice, could dogfood
  - Similar to other enterprise application areas (Customer relationship management, etc.)
  - Rich in roles, objects, actions, and protection need:
    - Roles: Employees, Hiring Managers, Interviewers, Recruiters, Sourcers, Candidates
    - Objects: Job Positions, Interviews, Job Applications, etc.
    - Actions: Scheduling interviews, collecting feedback, searching through applications, etc.
    - Protection: Confidential applications, Candidate can't see own interview, etc.
Big company acquires the platform

- Platform was so cool it attracted a big company
  - Big company expressed love for the platform and acquired company

- Big company loved platforms, full of platform approach zealots
  - No shortages of platforms: available and/or required
  - Amazing globally consistent database required to use for persisted data
    - Has everything but:
      - Low latency (Recall "transparent access")
      - Protection (Need additional layer to enforce ACLs)
Overall Platform Takeaway

● App demonstrated enterprise apps don’t need to suck
  ○ Favorite customer feedback: Don't need to read a manual to use
  ○ Strikingly better than what we use here at Stanford

● Can definitively conclude platform didn’t prevent building a great application

● Hard to tell the value offer by the platform:
  ○ Comparison with other approaches difficult
  ○ Focus on single application meant value of object repository idea not fully explored
    ■ Support for sharing data between applications not evaluated
  ○ Annotations were not used a lot
    ■ One app meant relatively small set of who and why.
  ○ Protection could be done in app so object level protection support was not strictly required
Experience with complete annotated history

- Some positives of keeping complete history:
  - Unparalleled debugging environment
  - Useful for machine learning training data
- Big company had a love/fear relationship with data (new oil/asbestos)
  - Sued many times over data collection (lots of effort to prevent it happening again)
  - Wipeout - Robust data deletion governance
  - Right to be forgotten - Confusion between consumers and work for hire
- Many companies believe that old information is more valuable to enemies
  - Corporate data retention policies focus on destroying everything not currently needed
  - On-boarding training course warned against collecting information without definitive purpose
- Deleting is problematic in interconnected objects
Experience with complete annotated history

- Disappointment: Complete annotated history implementation didn’t deliver
  - Too performance expensive to compute values from history
    - Example: Candidate stage times
  - Reports and analytics performance problems
    - Needed indexes, was not SAP Hana
    - Replicate into analytics database

- Had to import data into/export from different systems
  - Lost history
Experience with embedding protection in the objects

- Did significantly simplify backend code to not have access enforcement
  - Nice to not have to think about protection when reading/querying
  - Thankful when we hit protection violations
  - Particularly for search

- Some app protection couldn’t be captured in ACLs
  - Can’t look at interview rating until user submits own
  - Need understandable protection system - query to see who had access.

- Personas - “Person wearing a Hat”
  - Attractive vision - hard to realize with identities coming into the system with different ways
  - Pairing Personas to Persons was challenging. (Candidate/Employee)
  - Users want clear separations - Personal/Company - Single login not wanted (at BigCompany)
Experience with Transactions & Snapshot reads

- **Huge win for eliminating races and related concurrency errors**
  - Need to use a system that lack these features to realize this
  - Almost every place I saw multi transaction operations had bugs

- **Bridging transaction between different external systems was a problem**
  - Generic systems problem in federated systems
  - Issue for inter-tenant single-system image

- **Task system was an interesting example**
  - Lame by task systems standards
  - Advantage of integration with transaction system was huge
  - Used to interface to external systems
    - Required due to optimistic concurrency control
Experience with Transactions & Snapshot reads

- Discovered of the importance of transaction limits
  - Systems folklore: Only valid numbers are 0, 1 and ∞
  - Platforms: Transactions above a certain size take too long and use too much resources
  - Developers: Need to detect and special case in places

- Optimistic concurrency control for transactions is great
  - Until they quit working on legitimate workloads
Activity API

- Lack of direct needs for annotations
- Read-only activity too frequent to create objects or even versions of objects
  - If you want to record reads, use a log
- Developers had brains stuck on CRUD
  - Frequently made intent to be described by the operations on the persistent objects
- Automatic draft feature didn’t work for front-end developers
- Couldn’t implement a general undo/redo mechanism
  - Should have been possible in theory
Interface Models - Live Updating

- Idea: Decouple the app view of objects from persistent objects
  - Good idea except for some developer complaints. GraphQL is interesting

- Idea: Front-end should show the most current value of object data
  - Object mutations are detected and pushed to clients viewing the information
  - Full stack connection: Changes detected in objectstore result in views updating

- App views tend to be less out of date than other apps I’ve seen
  - Pretty painful to waste work on collaborative apps because of stale views

- Always-on live updating was not the right decision
  - Front-end developers had to consume the update pushes
  - Could cause unnecessary load on servers if fresh data isn’t needed
  - Lesson: Would make an frontend developer specified option
Conclusion thoughts

- **Centralized information store concept - Attractive in concept**
  - Current data is splayed across many systems - Not going to switch overnight
  - Trend to microservices going the other direction - Hard to get buyin for centralized object store
  - Takeaway: Consider real world on your ideal world

- **Transactions, snapshot consistent reads, fine-grain protection are good**
  - Going to be challenging with federated systems approach
  - Protection lacking databases doesn't help things

- **Opinionated SaaS platform design**
  - Takeaway: Options to do something is better than always