SOA from the trenches

Jordi Pradel
The project

13th Century Social Network of Deeds in France
(http://www.howweknowus.com/2008/07/23/great-work-lousy-title/)
The project

• Development of a new implementation of an existing social network:
  – Mobile and geolocation capabilities
  – Over 500K users
  – Mobile clients: iPhone, Java ME, Android
  – Backend

• Existing system not properly scaling
  – PHP backend
Main features

• Friendships
  – Friendship requests, accept, deny, block users

• Interests (moderated)
  – Subscribe to interests, create new interests, send mass messages to everyone that has an interest
  – Interests do have a profile picture
  – Aprox. 90k interests
Main features

- Profile picture (moderated)
- Online / Offline status

- Status: One line message of the current status of the user (moderated)
  – Connected to Twitter
Some complex features

• The messaging system as a chat wannabe
  – Mail like
  – Messages to interests (up to 30k users)
  – Notification of new messages to devices
  – Message sent by email as well

• List of everything every user did
  – Friendships, status changes, interests, messages, etc.
Some complex features

- (Almost) every list of users has some interesting tags:
  - Current status (one line message)
  - Distance in friendship graph to the logged in user
  - Number of friends in common with the logged in user
  - Friendship status to the logged in user (friends, pending friendship request, etc.)
  - Number of interests in common with the logged in user
  - Online / Offline status
  - Profile picture
Some complex features

- Friendship connection
Some complex features

• Interesting people

  – Interests in common
  – Contacts in common
  – Location
Encounters

• Locate the user device on each request
• Look for:
  – Users at bluetooth range
  – Users in a 2 km radius
  – Users in a 7 km radius
  – Users in a 50 km radius
• Create encounters
  – Directly and transitively
Architecture overview

Clients
- Mobile client
- Current web application client

Compatibility layer
- APIs
- Business Layer
- Message Bus

Services layer
- Interests
- Users
- (...)
- Etc.
Projects

- apis
- business logic
- helper libraries
- device notification
- image storage

- activity
- session
- status
- settings
- interests
- users
- encounters
- friends
- location
- marketing
- messaging
Practice and theory

In theory, there is no difference between theory and practice (attributed to Jan L.A. van de Snepscheut, Yogi Berra, Chuck Reid...)
Fundamentals: How services relate

- Services can be used by other services or programs as far as they are aware of the service they want to use
  - Service description:
    - Name
    - Location
    - Data exchange requirements
  - Messaging
    - Messages as independent units of communication
Fundamentals: How services relate

Mobile client

Current web application client

Compatibility layer

APIs

Business Layer

Message Bus

Services layer

Interests

Users

(...) Etc.

Google Location API

Twitter

Apple Push Notifications
Fundamentals: How services relate

```java
public class UsersController {
    private UserService userServiceM
    ...
}

@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public interface UserService {
    @POST @Path("/")
    public Long createUser();

    @GET @Path("/name/{userName}")
    public UserDTO getUserByName(@PathParam("userName") String userName);
}
```
Fundamentals: How services are used

• How should we design...
  – Services
    • Web Services
    • REST like
  – Service descriptions
    • Based on a Java Interface
  – Messages
    • Translated to REST + JSON parameters & return types
  – Relationships between services
    • Only from Business Layer to other services
Fundamentals: Principles of SO

- ✓ Low Coupling
- ✓ Service Contract
- ✓ Autonomy
- ✓ Abstraction
- ~ Reusability
- ✓ Composability
- ✓ Statelessness
- ✗ Discoverability
Fundamentals: Contemporary SOA

• Generally
  – Based on open standards
    • Pragmatism over heavy standards: REST
  – Architecturally composable
  – Capable of improving QoS
    • INDEED!
      • One of the main reasons in adopting SOA
Fundamentals: Contemporary SOA

- Support, foster and promote
  - ✓ Vendor diversity
  - ✓ Intrinsic interoperability
  - ✗ Discoverability
  - ✗ Federation
  - ~ Inherent reusability
  - ✓ Extensibility
  - ✗ Service-Oriented business modelling
  - ✓ Layers of abstraction
  - ✓ Enterprise-wide loose coupling
  - ✓ Organizational agility
Message Exchange Patterns

• Request-response
  – Single destination, synchronous
  – Main MEP

• Fire-and-forget
  – Single destination, no response
  – Could have been used, but wasn’t (see pub&subs)

• Publish-and-subscribe
  – Asynchronous, JMS based
  – Event system to avoid waiting for a response
Activity Management & Composition

- Atomic transactions: Compensation instead
- Business activities: There aren’t
- Coordination
- Orchestration
- Addressing
- Reliable messaging
- Correlation
- Policies
- Metadata exchange
- Security: Ad-hoc
- Notification and eventing: Internally
Activity Management & Composition

• Services
  – Internal:
    • Choreography written in Java
    • No need for discoverability neither mgmnt activities
  – External
    • Choreography written in Java
      – Java is easier to use than WS choreography tools
    • Twitter, APN, Google location API
    • Fixed providers: No need for disoverability
    • Ad-hoc security on each one
Reality strikes!
Listing users

• (Almost) every list of users has some interesting tags:
  – Current status (one line message)
  – Distance in friendship graph to the logged in user
  – Number of friends in common with the logged in user
  – Friendship status to the logged in user (friends, pending friendship request, etc.)
  – Number of interests in common with the logged in user
  – Online / Offline status
  – Profile picture
Listing users

• by last activity (online only / all)
• by signup date
• by interest
• visitors of a user profile
• friends of a user
• by country
• interesting people
• ...
Listing users: Interesting people

- Interesting people

- Interests in common
- Contacts in common
- Location
Listing users

1. Query users service to get users info
   – Name, sex, age, city, country, signup date, etc.
2. Query interests service to get users interests
3. Query friendship service to get users friends
4. Query activity service to get users activities
Listing users

• Lists are paginated
• Over 500K users in the DB
• Therefore:
  – Order&filter criteria is really important!
  – e.g. List friends of mine (from friends service) whose interests include interest I (from interests service) ordered by signup date (from users service)
  – None of the services can paginate and filter by itself
    • We need to paginate and filter across services
Caching

• Some service requests are easy to cache at the service level
  – GET /users/profile/23

• Others are really difficult
  – GET /users/profile?id=23&id=34&… (hundreds of ids)

• So, service requests were cached at the client level by hand
  – Ugly, I know, but efficient
Low coupling

• How could we maintain low coupling between services?
  – Services never use other services
  – They are like DBs on steroids
  – Only the Business layer is coupled to every other service (which is quite a high coupling)
Eventing

• Some business processes are better modeled with eventing in mind
  – When a user sends a request from a mobile device...
    • The request must be served
    • The device must be located
      – Then, encounters with other users must be triggered
    • In case the request is interesting to marketing, the event must be recorded in a marketing DB
    • In case an Apple device is interested (e.g. someone who has been encountered), the device must be notified
    • Similar for Bluetooth devices found
  – Therefore: An event is generated and multiple components subscribe to it
public void onEvent(UserLocated event) {
    Map<Long, Float> users = locationController.getUsersNearLocation(...);
    for (Long encounteredId : users.keySet()) {
        triggerEncounter(event.getUserId(), encounteredId);
    }
}
Performance & Scalability

• Greatly improved!

• Attending a request is a matter of coordinating N services
  – e.g. Many services to list users
  – e.g. Many services when the device provides a location

• But we can serve partial results
Performance & Scalability

• Bottlenecks are located in a single service, which may not be critical
  – e.g. Marketing was executing huge queries against the production DB
    • e.g. Total number of users per month and country
    • Marketing is only stressing the marketing DB now
    • Which contains information originated in events marketing is listening to
  – Every service can be fine-tuned to its usage scenarios
    • DB
    • Caching
    • etc.
  – Some services could use different DB technologies
Performance & scalability

- We needed bulk requests
  - e.g. Listing users requires the online/offline status of hundreds of users
  - We can’t query it one by one
    - e.g. 1 request → 1 ms
    - 500 requests → 0.5s

- Transactions?
  - 1 tx -> 10ms
  - Are they really needed?
Conclusions & Lessons learned

Jean-Honore Fragonard – The Music Lesson
Conclusions & Lessons learned

- Partitioning into services is **critical**
- Test & measure to find bottlenecks before optimizing blindly
- Need for an application layer which uses services and composes a rich functionality
  - **Very high cohesion** of services
  - Services tend to be quite simple
  - This layer has an important level of coupling
- Good architecture in technical terms
  - Solves performance & scalability issues
- Diagnosis gets worse
  - We need to identify a request across all the services
Thanks!

jpradel@essi.upc.edu