Webmail for Millions

Powered by Erlang

Scott Lystig Fritchie / Joseph Wayne Norton
fritchie@geminimobile.com / norton@geminimobile.com

Gemini Mobile Technologies, Inc.

March 25, 2010

Road Map

- Introduction: who, what, why. . .
- Architecture at high altitude
- UBF's many, many uses
- One-slide introduction to the CAP Theorem
- A new distributed key-value store: Hibari
- Testing!
- What worked, what didn't

Introduction

Who is Gemini Mobile Technologies?

- Founded: July, 2001
- Offices: San Mateo, CA; Shibuya, Tokyo; Star City, Beijing
- Milestones:
  - 2003: Multimedia messaging service (MMS), Vodafone Japan
  - 2005: MMSC, Nextel International
  - 2006: MMSC, eMobile Japan
  - 2008: eXplo(tm) service, China Unicom
  - 2009: International MMS gateway, NTT docomo
- Investors: Goldman Sachs, Ignite, Mizuho Capital, Tokyo MUFJ, Nomura, Access, Aplix
- Erlang: Apps in Japan & China telecoms use for 3 years

Introduction

Does the World Need Another Webmail System?

- Stand out from the crowd
- Replace something else. . .

Introduction

The Something Else

- Based upon Oracle DBMS
- Could not easily and cheaply scale to GByte mailboxes
- Not Gmail-like enough, not customizable enough
- Could not easily support mailbox convergence

GEMINI Mobile Technologies™

Introduction

Must-Have Features

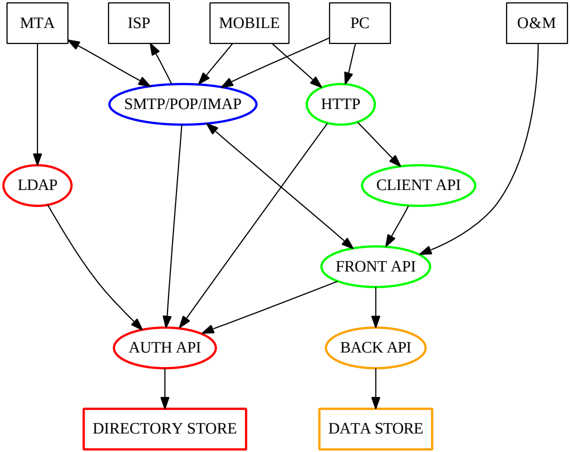
- UI at least as rich as Gmail, preferably more
- Very high quality, reliability, and customizable
- Focused customization for Japanese customers, Japanese language
- PC clients, “smart” handsets, not-so-dumb legacy handsets
- Cheap! Supported only by advertising or very low monthly fee
- Must integrate with legacy authentication, monitoring, billing, and full-text search services

Road Map

- Introduction: who, what, why. . .
- Architecture at high altitude
- UBF's many, many uses
- One-slide introduction to the CAP Theorem
- A new distributed key-value store: Hibari
- Testing!
- What worked, what didn't

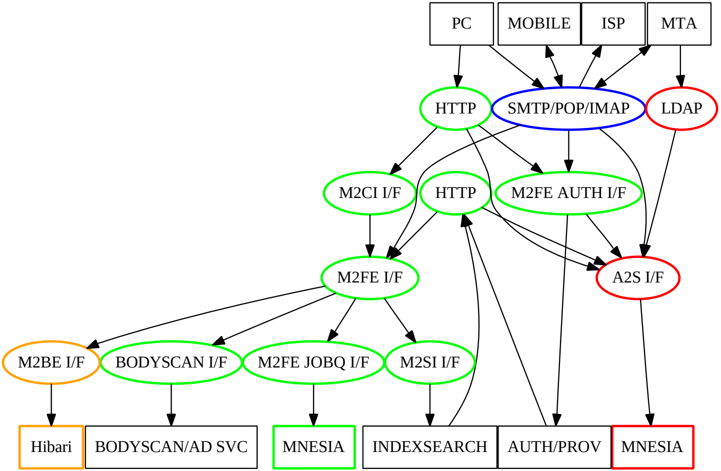
Multi-Tier Architecture

20K Meter View



Multi-Tier Architecture

10K Meter View



Erlang

What's It Doing?

- JSON-RPC with the customer's Web browser-based UI (based on UBF)
- SMTP, POP, and IMAP with the external email world
- HTTP and LDAP with authentication and full-text indexing services
- UBF for most inter-application communication
- Interface with C++ components for speed, legacy protocol support, and code re-use
- Application logging and tracing
- Transaction logging for message tracing
- Custom distributed, scalable key-value store for all persistent data (English: Skylark, Japanese: Hibari)
- Mnesia for job queuing and multi-indexed profile data

Screen Shot

Every presentation has to have at least one screen shot. . .

Screen Shot

Yes, Really

?

Road Map

- Introduction: who, what, why. . .
- Architecture at high altitude
- UBF's many, many uses
- One-slide introduction to the CAP Theorem
- A new distributed key-value store: Hibari
- Testing!
- What worked, what didn't

UBF: The Communication Workhorse

What Is It?

- UBF is RPC with a formal, precise specification
- Abstract syntax, concrete ("on the wire") syntax, and meta-level protocol
- Strict enforcement of protocol specification: the "contract"
- Erlang server implementation, clients in various languages
- Simple yet elegant, concise yet expressive
- Easy to extend and to customize to our needs

Many, many thanks to Joe Armstrong, UBF's designer and original implementor.

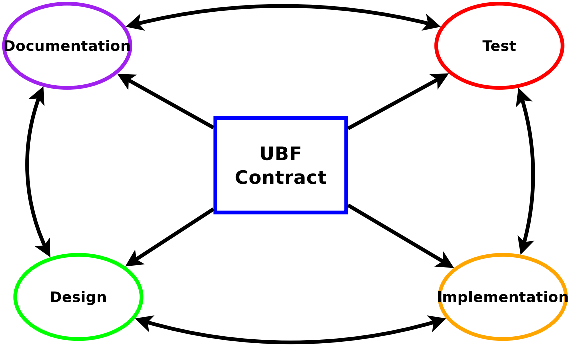
UBF: The Communication Workhorse

Tier/Application Layers

- Customer Interface API: UBF over JSON-RPC to/from the Web browser
- Authentication API: interface to custom and legacy authentication services
- Front End API: includes more protocols, O&M interfaces
- Back End API: low-level API for managing mailboxes, profiles, address book/vCard data, filtering rules, etc.
- M2G API: Separate C++ app for legacy services and protocol support

UBF: The Communication Workhorse

One Contract, Many Uses



UBF: The Communication Workhorse

Contract Statistics

API	Contracts	Methods	Types	Leaf Types	Records
Auth	2	26	96	53	4
Client	5	28	288	231	13
Front	11	61	469	358	32
Back	10	29	186	136	5
Total	*28*	*85*	*628*	*443*	*35*

“Front End” Add a Draft Message

Same Contract. . .

```
+TYPES
mail_add_draft_req() = {mail_add_draft
                       , authinfo()
                       , maildraft_olduid()?
                       , mailheaders()
                       , draftbody_parsed()
                       , [rfc2396_url()]
                       , maildraft_options()?
                       , timeout_or_expires()};
mail_add_draft_res() = {ok, uid(), [mimepart_url()]} | folder_res_err();
+ANYSTATE
mail_add_draft_req()    => mail_add_draft_res();
```

"Front End" Add a Draft Message

. . . Different Protocols

Client API - add a draft mail (UBF, EBF, and ETF style)

```
{ mail_add_draft, authinfo(), maildraft_olduid()?, mailheaders(), draftbody_parsed()
   , [rfc2396_url()], maildraft_options()?, timeout_or_expires() }

=> { ok, uid(), [mimepart_url()] } | folder_res_err();
```

Client API - add a draft mail (JSON-RPC style)

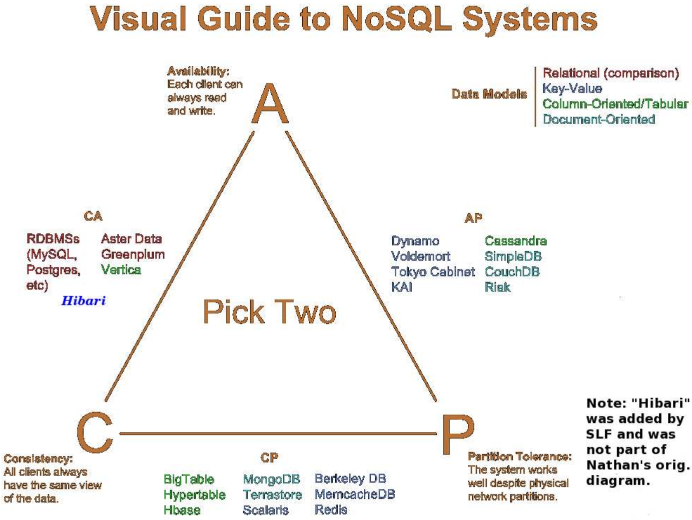
```
request {
        "version" : "1.1",
        "id"      : "Y101",
        "method"  : "mail_add_draft",
        "params"  : [ maildraft_olduid()?, mailheaders(), draftbody_parsed()
                    , [rfc2396_url()], maildraft_options()?, timeout_or_expires() ]
 }
response {
        "version" : "1.1",
        "id"      : "Y101",
        "result"  : {"$T" : [ {"$A" : "ok"}, uid(), [mimepart_url()] ]}
                    | folder_res_err() | null,
        "error"   : error()?
 }
```

Road Map

- Introduction: who, what, why. . .
- Architecture at high altitude
- UBF's many, many uses
- One-slide introduction to the CAP Theorem
- A new distributed key-value store: Hibari
- Testing!
- What worked, what didn't

One-Slide Intro to CAP

Nathan Hurst, http://blog.nahurst.com/visual-guide-to-nosql-systems



Visual Guide to NoSQL Systems

Availability: Each client can always read and write.

A

Data Models
- Relational (comparison)
- Key-Value
- Column-Oriented/Tabular
- Document-Oriented

CA

RDBMSs (MySQL, Postgres, etc)

Aster Data
Greenplum
Vertica

Hibari

AP

Dynamo
Voldemort
Tokyo Cabinet
KAI

Cassandra
SimpleDB
CouchDB
Riak

Pick Two

C

Consistency: All clients always have the same view of the data.

CP

BigTable
Hypertable
Hbase

MongoDB
Terrastore
Scalaris

Berkeley DB
MemcacheDB
Redis

P

Partition Tolerance: The system works well despite physical network partitions.

Note: "Hibari" was added by SLF and was not part of Nathan's orig. diagram.

Hibari

Key-Value Storage for (Almost) Everything

The only (?) key-value DB that offers on strong consistency...

- "Chain replication" for strong consistency
  - See paper by van Renesse and Schneider, OSDI 2004 conference proceedings
  - Previous slide: "CA" (Consistency, Availability)
- Consistent hashing for distributed key placement
- Automatic repair of crashed/rebooted bricks
- MD5 checksums on all data on disk
- Replication factor (chain length) changeable online
- Cluster size (number of chains) changeable online

Hibari

Open Source Release, Soon

Distributed Key Value Database
Hibari
HyperScale® Cloud Database System
Coming Soon

- Working on license details, code prep, and documentation
- Planning to release via GitHub by mid-May 2010

Mnesia

Storage for Everything Else

- User profile storage: indexing & retrieval by various attributes
- Job queuing: notifications to handset, notifications to external text indexer, . . .
- Doable with Hibari-based storage, but Mnesia was easier

Road Map

- Introduction: who, what, why. . .
- Architecture at high altitude
- UBF's many, many uses
- One-slide introduction to the CAP Theorem
- A new distributed key-value store: Hibari
- Testing!
- What worked, what didn't

Testing, Testing, Testing, Testing

. . . and more testing . . .

- Unit tests via EUnit (Erlang)
- Typical hand-coded test cases (Python)
- QuickCheck models
  - Automatically derived from UBF contracts, easy to create custom generators whenever necessary
  - Hand-made generators & models to test other code
- Load/stress testing (Erlang, Python)
- Remember: Implementation & testing is 90% of your effort. Carrier testing is the other 90%.

Post (almost) Mortem

Stuff We'll Repeat

- Erlang, the secret sauce
  - Ericsson's support of Erlang/OTP is wonderful
- UBF
- QuickCheck
- Auto-compilation of UBF contract → QuickCheck generators
- Documentation tools: Git, AsciiDoc, Graphviz, "mscgen"
- Automate everything possible: regression tests, performance tests, cluster setups, post-mortem log file gathering, . . .
- Test in various environments:
  - Exactly the same hardware as customer, on really old & slow hardware, and on a single box/laptop

GEMINI Mobile Technologies™

Post (almost) Mortem

Stuff We Would Probably Do Differently

- Negotiate “less aggressive” schedule
  - Keep dreaming, Scott. . . .
- Buy more hardware
- Always test X & Y before customer tries doing X & Y
  - Get their test plan, then do it before they do.
- Better and more peer code review
- Always revisit and cleanup “initial” prototypes
- 100% automated unit test and code coverage analysis

Summary

- Technically, Erlang was a great fit for this large system.
  - Used another language (C++) whenever convenient.
- UBF is a very good tool for design, implementation, and testing phases of a large project.
- Combining UBF and QuickCheck was invaluable in finding bugs that otherwise would've been discovered by the customer.
- It's feasible to develop real-time apps on top of a distributed key-value database.
  - Hibari's "strong consistency" support is a large advantage.

GEMINI Mobile Technologies™

Thank You Very Much!

Distributed Key Value Database
Hibari
HyperScale® Cloud Database System
Coming Soon

- Look for Hibari announcements in April-May
    - Email me or Joe if you cannot wait. . . .
- UBF code is already available at GitHub
    - http://github.com/norton/ubf
    - http://github.com/norton/ubf-abnf
    - http://github.com/norton/ubf-eep8
    - http://github.com/norton/ubf-jsonrpc

GEMINI Mobile Technologies™