Putting UBF to Work

(and Getting the Outside World to Talk to Erlang)

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

Gemini Mobile Technologies, Inc.

November 12, 2009

A Quick Survey

Universal Binary Format

Who has heard of UBF ...

- inside the Erlang community?
- outside the Erlang community?

Who has tinkered with UBF ...

- at home?
- at work?

Who has deployed UBF ...

- as part of a commercial service?
- as part of a commercial product?

Introduction

Protocols and Specifications

Many protocols have formal specifications

- ASN.1, ONC-RPC, Corba, AMQP, Thrift, Protocol Buffers, zillion more ...

Why does industry use such specifications?

- Specify bits "on the wire" in a way all parties agree
- API documentation: how the protocol's API works

Introduction

continued...

We have found that there are very important other reasons ...

- System design and architecture: input for humans
- Protocol meta-data: input for tools

UBF has proved to be very handy in helping us with the above items ...

What is UBF?

in a nutshell

- UBF(A) is a protocol above a stream transport (e.g. TCP/IP), for encoding structured data roughly equivalent to well-formed XML.
- UBF(B) is a programming langauge for describing types in UBF(A) and protocols between clients and servers. UBF(B) is roughly equivalent to to Verified XML, XML-schemas, SOAP and WDSL.
- UBF(C) is a meta-level protocol used between UBF client and servers.

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

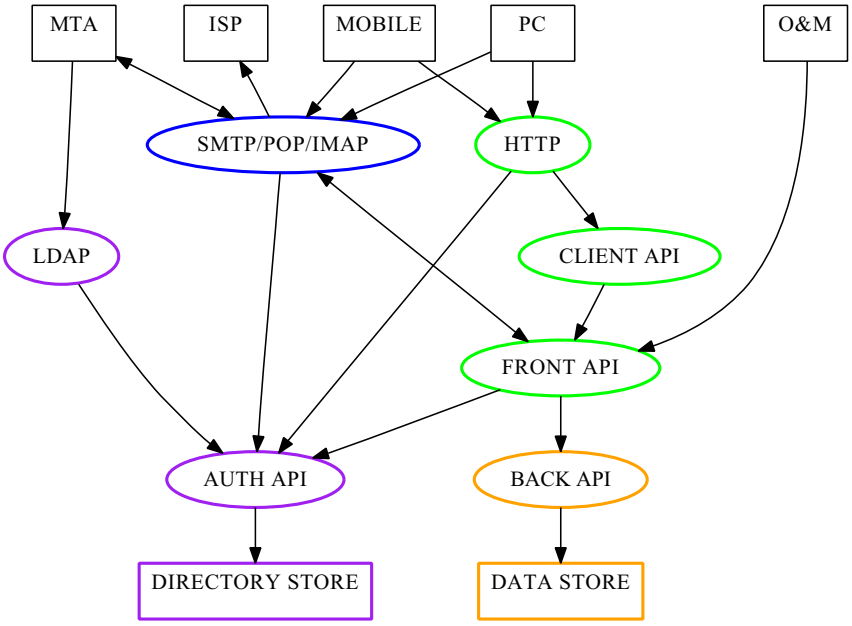
Why UBF?

in a nutshell

- RPC with a formal, precise specification
- Erlang server implementation
- Erlang and non-Erlang client implementations
- Simple yet elegant, concise yet expressive
- And most importantly ... easy to extend and to customize to our needs

UBF Case Study

A Custom Webmail System



UBF Case Study

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*

These code snippets show how to obtain the methods, types, leaf types, and records of an UBF contract.

```
Methods = [ {Req,Res} || {Req,Res} <- Mod:contract_anystate() ].
Types = [ {T,Mod:contract_type(T)} || T <- Mod:contract_types() ].
LeafTypes = [ {T,Mod:contract_type(T)} || T <- Mod:contract_leaftypes() ].
Records = [ {R,Mod:contract_record(R)} || R <- Mod:contract_records() ].
```

UBF Contracts

Changes & New Features

Predefined primitives

- Renamed constant() to atom()
- Renamed int() to integer()
- New float(), tuple(), list(), and proplist() primitives
- Optional type()? - 'undefined' or type()
- Optional attributes (e.g. binary(ascii,nonempty))
  - ascii: only ASCII values
  - asciiprintable: only printable ASCII values
  - nonempty: not equal to '', [], <<>>, or {}
  - nonundefined: not equal to 'undefined'

UBF Contracts

continued ...

User-defined primitives

- New binary and float constants
- Support Erlang syntax for integer constants
- New integer ranges (..integer(), integer()..)
- #record{...} syntax with automatic generation of Erlang record defines
- [type()]? for optional lists
- [type()]+ for mandatory lists
- [type()]{N}, [type()]{N,}, and [type()]{M,N} for length-constrained lists

UBF Contracts

continued ...

New feature "type importing" ...

- Permit type only contracts (i.e. STATE and ANYSTATE contract blocks are now optional)
- Import UBF types from other UBF contracts
- Check and permit duplicate UBF types only having the same definition

UBF Contracts

continued ...

New feature "type importing" ...

- Import ABNF-based types from ABNF specifications
  - ABNF-based types are formal specifications for binary() types.
  - Ander Nygren's abnfc module is used to parse ABNF specifications into an internal abstract syntax tree (AST).
  - Implemented new ABNF parser for UBF's contract checker to verify binaries against ABNF-based types.
- Import EEP8-based types from any Erlang module
  - Using a parse transformation, UBF contract types are automatically added to an existing Erlang module having type defines.
  - Not all EEP8-based types are supported (pid(), fun(), ...)
  - Support for EEP8-based records is in progress

UBF Plugin Callbacks

Changes & New Features

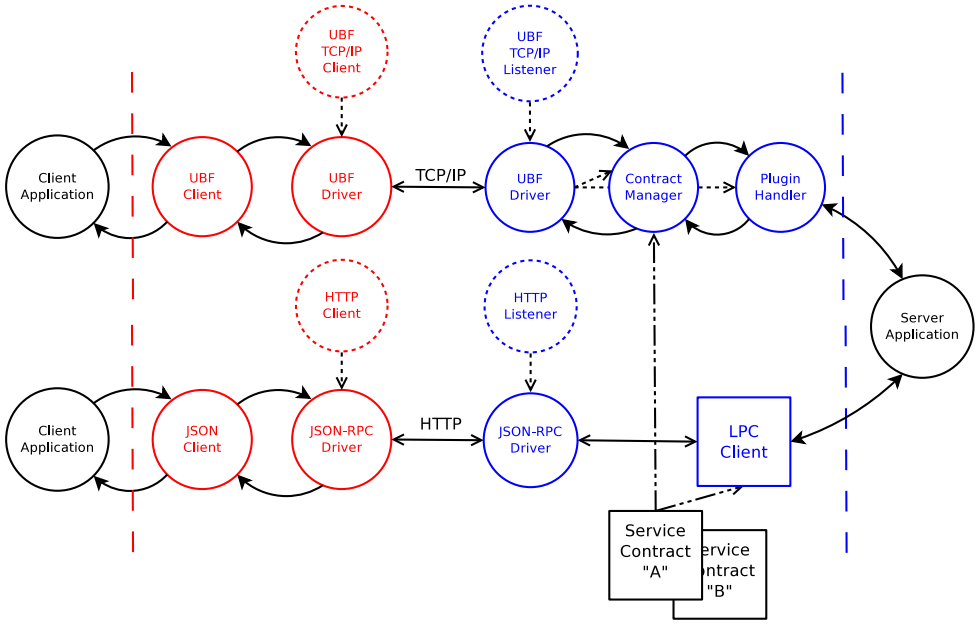
Stateful is the original callback API implementing a shared plugin manager process, a per-session contract manager process, and a per-session plugin process.

Stateless is a new callback API implementing only a per-session contract manager process and per-session plugin process. The implementation callback function API is a bit less complex.

LPC is a new callback API implementation that has no side-effects. LPC stands for Local Procedure Call.

The implementor of a UBF plugin can choose to implement one, two, or all three of the above callbacks.

UBF Transports



Key Points

- Same contract with multiple transports
- Same application with multiple contracts

UBF Transports

Changes & New Features

EBF is "Erlang Binary Format", a simple TCP/IP protocol that uses Erlang-style conventions.

- Uses Erlang BIFs term_to_binary() and binary_to_term() to serlialize terms.
- Terms are framed using the 'gen_tcp' {packet, 4} format: a 32-bit unsigned integer specifies packet length.

ETF is "Erlang Term Format", a simple protocol that relies on Erlang's native distribution. This approach can be useful for Erlang-only deployments.

UBF Transports

continued ...

JSF is "JavaScript Format", a simple TCP/IP protocol that uses JSON (RFC 4627).

- Uses LShift's Erlang-rfc4627 to serialize terms.
- A few extra conventions are layered on top of LShift's implementation to help distinguish between atoms, tuples, records, and ubf strings.

UBF-JSONRPC is a framework for integrating UBF and JSON-RPC over HTTP.

- Relies on JSF and provides new helper utilities to encode and to decode JSON-RPC requests and responses.
- Includes a simple inets-based HTTP client and HTTP server module that demonstrates how to use the LPC callback API.

UBF Meta-Data

Documentation

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()] } | res_err();
```

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

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


UBF Meta-Data

Testing

Functional Testing

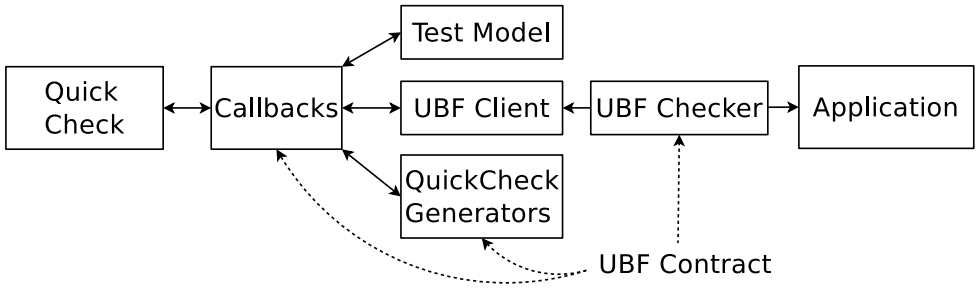
- Integration - external clients and external servers
- Dialyzer - re-use type definitions
- EUnit - automatic test input generation
- QuickCheck - automatic generation of QuickCheck generators and abstract state machine property tests

Performance Testing

- Load client generators
- Load server stubs
- Transaction logs and statistics

UBF and QuickCheck

Basic Strategy



- Boilerplate generators and properties *(with only one push of a button)*
- Custom generators
- Custom properties
- Non-UBF APIs
- Top-down and bottom-up testing
- Low-level transport and high-level application layers

What's Next?

Documentation

- Complete edocs and examples for the UBF code repositories
- ABNF specification for UBF(A), UBF(B), and UBF(C)
- ABNF specification for EBF, JSF, and JSON-RPC

Interoperability

- EUnit: open-source the input generators for UBF
- QuickCheck: open-source the generators and the framework for UBF
- EEP8: better integration with Erlang specs, types, and records
- FFI: Erlang ports and drivers based on UBF for other languages (e.g. C/C++, Haskell)
- Other tools and approaches (e.g. Protocol Buffers, Thrift, BERT)

Thank You

http://github.com/norton/ubf
http://github.com/norton/ubf-abnf
http://github.com/norton/ubf-eep8
http://github.com/norton/ubf-jsonrpc