Elixir in an Enterprise Telecommunications Product

Stephen Pallen

July 2014

ELIXIR|> CONF 2014

E-MetroTel
Exceptional Innovation

I'll Cover

- Real, concurrent embedded product
- Enterprise Telephones
- The product
- Architecture overview
- Challenges
- Learnings, including porting a C application to Elixir

Who am I?

Stephen Pallen

- steve.pallen@emetrotel.com
- github.com/smpallen99
- Ottawa, Ontario, Canada
- 26+ years in telecommunications, software development, project management, agile, and leadership
- Partner, R&D Lead at E-MetroTel

E-MetroTel
Exceptional Innovation

- Startup, providing cost effective communications solutions
- 4+ years in business
- Offers customers with Nortel Enterprise phone systems a cost effective upgrade path to unified communications

www.emetrotel.com

Background

- We originally wrote the application in C with a couple production deployments currently using the C version
- Rewrote the complete C version in Elixir (1 component left to rewrite)
  - More maintainable
  - Much less code
  - More reliable and scalable
- Elixir version is in our product verification lab

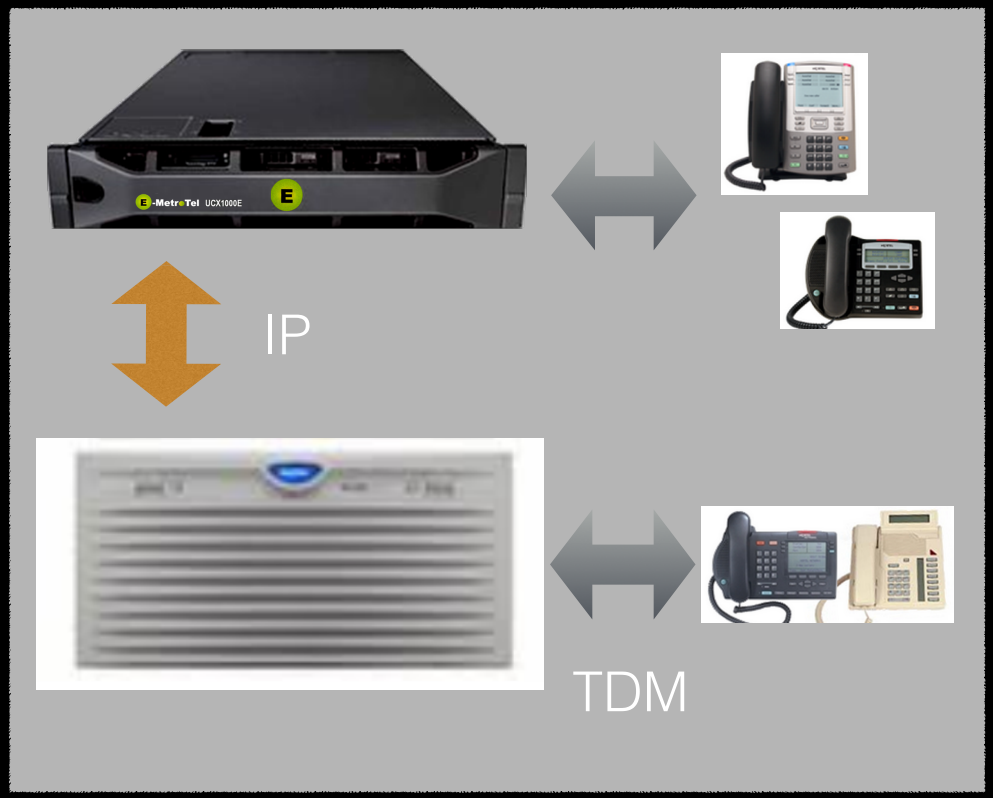
Product - The Starting Block

- Open source IP PBX
- Supports SIP and other vendor IP phone protocols
- Already supports legacy PBX IP phones

- End of life Legacy PBX
- Large installed base of Digital and Analog phones (not IP)
- Vendor hardware cabinets hardware supports IP interface that drive non-IP phones
- Customers want modern phone features, but don’t want to buy new phones

Product - End Game

- Digital set emulator software
  - Protocol converter
  - Converts IP protocol to messaging understood by the legacy gateway controller
  - Supports up to 60 gateways and 10,000 phones

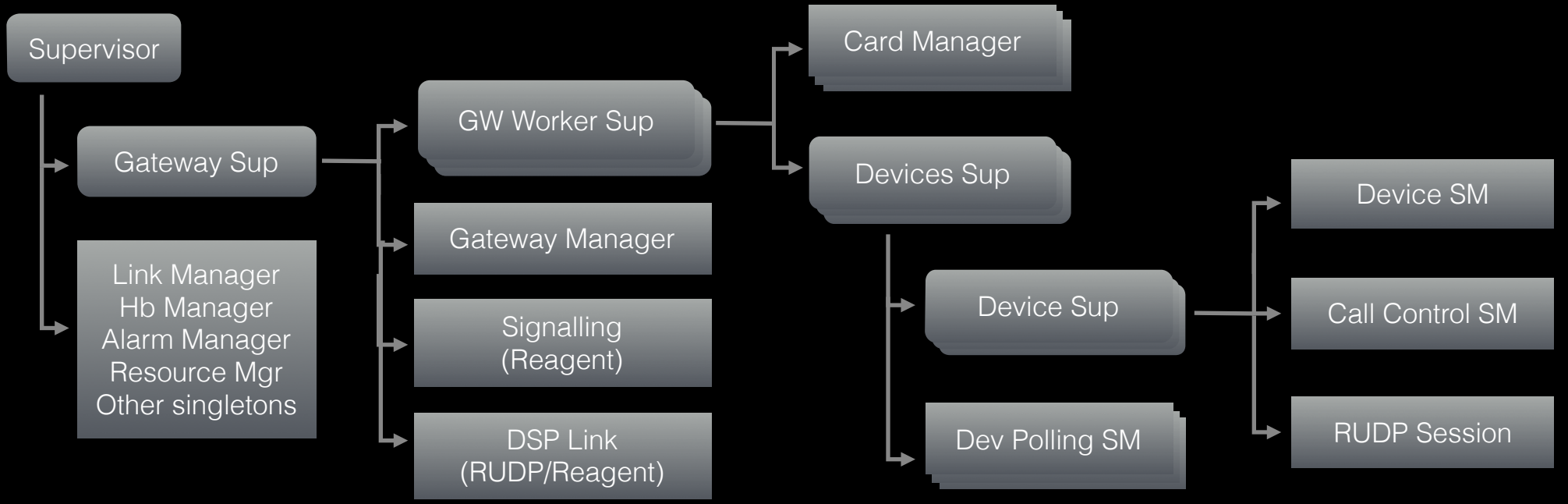


Application Features

- Support Sun RPC for gateway registration
  - Decided to keep C version of this protocol. Used Elixir TCP to communicate with the C service
  - Plans to move to Erlang Sun RPC library in the future
- Accept TCP & RUDP connections from gateway
  - Use Reagent for TCP; wrote an RUDP stack
  - Line card install / remove detection

- Handle phone features
  - Phone ID and type detection; Key press detection
  - Phone Display messages
  - Start & stop tones; operational parameters; Lamp Control; Key features
  - Call features - state machines
  - Switching tones (dial tone, ring back tone), Switching speech path
  - Handle DSP resources, tone generator and tone detector resources
  - Digital phone polling - detects when a phone starts/stops responding

Architecture - Summary



Performance

- Application co-resident with Asterisk PBX
- Needs to support
  - Up to 8 calls per second (setup and tear down messages the bottle neck)
  - Approximately 50 messages per call
  - House keeping like phone polling, heartbeat, etc.
- Less capacity than C (use multiple servers in the future)

Application Logging

- Existing C product uses syslog
  - Ported Erlang Twig package to Elixir to get syslog capabilities
  - Plans to extend new Elixir logger to support syslog
- Application complexity requires a fine level of log control
  - Two tier level / category to provide finer control
  - Macros allow logging levels/categories to be compiled out for performance

Configuration & Administration

- Use same config file schema as C version
  - Wrote configuration file parser and configuration server
- Config files generated by a Web Application
- Custom command line tools
  - Accessible from web page (not complete)

Command Line Console

- IEX based, remote console for production
- iex.exs - import Console
- help and help command
- Much easier to extend then C version
- Implemented with macros
- Full power of the elixir console

- Always bothered me the amount of effort to add a new commend in the C version
- Example command implementation:

```
command :hex, desc: “Convert decimal to hex”,
                    number: “An integer” do
    ExPrintf.sprintf “04x”, [number]
end
```

Multi Module Constants

- Port many lines of C #define statements
- Elixir does not support shared include files
- Wrote a define macro which solves the problem
  - matching supported
- Easy to port from C with Sublime text multi cursor support

```
#define NORMAL_MSG        1
#define SPECIAL_MSG       2
#define SOMETHING_ELSE    0x42
```

```elixir
defmodule MyConsts do
  use Constants
  define answer,  42
end

defmodule MyModule do
  import MyConsts
  def the_answer(answer), do: true
  def the_answer(_), do: false
end
```

```elixir
defmodule Constants do
 defmacro __using__(_opts) do
   quote do
     import Constants
   end
 end
 defmacro define(name, value) do
   quote do
     constant unquote(name), unquote(value)
   end
 end
end
```

Working with Hex Numbers

- inspect_hex
  - Most state and messaging logs must be printed in hex
  - Tried adding hex by default, but ended up creating inspect_hex
- sigil_H
  - Wireshark output of binary messages displayed in hex
    - i.e 1234 abcd 99ff 5678 fedc 9999 ffff …
  - Very useful for testing. Copy hex output from other tools

```elixir
@doc """
  Handles the sigil ~H. It takes a sequence
  of hex numbers and returns a list of numbers.

  ## Examples

      iex> ~H(1234 000f 8000 ab)
      [4660, 15, 32768, 171]
  """
def sigil_H(string, _opts) do
  string
  |> String.split(" ", trim: true)
  |> Enum.map(&(binary_to_integer(&1, 16)))
end
```

C Type Structure Serialization

- Common pattern C code to map binary data over the wire with C structure and bit fields
- Had to port many messages to Elixir
- Wrote C Structure module
  - Then ported from records to structs :)

```
struct binary_message {
  unsigned short msg_type;
  unsigned int field_1 : 3;
  unsigned int field_2 : 5;
  ...
}
```

```elixir
defmodule Test.CStruct do
  defmodule SubData3 do
    defstruct f1: 0, f2: 0
    use CStructure, endian: :big, schema: [ f1: 16, f2: 8 ]
  end
  defmodule MyListRecords2 do
    defstruct one: [],  two: [], three: 0
    use CStructure, schema: [one: [list: {[integer: 8], 4}],
                             two: [list: {[record: SubData3], 3}],
                             three: [list: {[string: 4], 2}]],
                             endian: :big
  end

  test "it works" do
    msg = <<0xfefdfcfb::32, 1::16, 2::8, 0xabcd::16, 0xbd::8,
                    0xffff::16, 0xdd::8, "abcd", "good">>
    data = MyListRecords2.load msg
    assert Enum.at(data.one, 1) == 0xfd
    assert Enum.at(data.two, 1).f1 == 0xabcd
    assert Enum.at(data.three, 0) == "abcd"
  end
end
```

Packaging & Installation

- Exrm to create releases
  - Creates a package with the Application and the Erlang and Elixir Run Time
  - Live upgrades, RPC capabilities
- RPM generation (exrm-rpm on hex)
  - Wrote a exrm plugin to generate RPMs
  - Easy to customize
  - Generates service script and start on server boot

Testing

- Choose amrita early in the project
  - Built in mocking
  - Supports grouping with facts, fact, describe, and it keywords
  - Has not been updated for a while - I'm still running 0.13.2 :(
- Excellent test coverage for platform / framework modules, less coverage for our problem domain code

- Wrote hardware simulator in Elixir
  - Simulates 100's of phones
  - Allows automated load testing of phone traffic

Elixir Benefits

- Supervisors!!!
- Call processing error drops 1 phone - not 1000s of phones
- Live upgrade (development and production)
- Powerful remote console
- Multi-node scailability
- Easier to maintain (much less code, no semaphores, pointer corruption, etc)

Challenges

- Had to train developers in Elixir
- Team buy-in, effort to port and performance concerns
- Defining the architecture while learning Elixir/Functional programming
- Elixir language churn
  - Porting 25+ records with OO modeled customer functions
  - Immature eco system (syslog, Sun RPC, etc)
  - Choose test framework not actively being maintained

Elixir |> experience? == :true

- Great community. Great to be involved pre 1.0
- Love Elixir, hate C
- Modest transition from Ruby
- Built a better product
- Looking forward to 1.0
- José Rocks!


Questions?