REST on Routers?

Prelim. Lessons for Language Designers, Framework Architects, and App Developers

Joe Kaylor
Konstantin Läufer (speaker)
George K. Thiruvathukal

Emerging Technologies Laboratory
Department of Computer Science
Loyola University Chicago, USA

Position Paper at ICSOFT 2011 Sevilla
Objectives

- Assess hardware and software solution stack choices available to developers of resource-oriented web services on low-power equipment
- Generalize experience gained from ongoing project
- Share early lessons with the community
- Similar in spirit to implementing-rest project (on Google Groups)
Context

- Chicago Clean Air Clean Water: ongoing interdisciplinary research project on air and water quality in a major urban ecosystem
  - sensors
  - analyzers
  - aggregators
  - publishers

- Required information infrastructure: role-based hierarchy of distributed, individually addressable, interconnected resources
Representational State Transfer (REST): Background

- Software architectural pattern described by Roy Fielding (2000)
- The web as a platform for building distributed systems from interconnected resources
- Richardson Maturity Model (RMM)
  - RMM 0: remote procedure call e.g. SOAP
  - RMM 1: addressable resources URLs
  - RMM 2: uniform interface HTTP verbs
  - RMM 3: hypermedia representations HATEOAS
REST: RMM Examples

- **SOAP:**
  always POST to http://myhost/mySoapService

- **URL:**
  http://cacw.luc.edu/lakeshore/baumhart/ts42i/no2/current

- **HTTP verbs:** PUT, GET, POST, DELETE, HEAD, OPTIONS, etc.

- Hypertext as the engine of application state (HATEOAS): representations include links that represent the next actions, corresponding to state transitions, currently available to the
Representational State Transfer (REST): Assessment

● Drawbacks
  ○ less tool support available

● Benefits
  ○ simplicity
  ○ high scalability through caching
  ○ high flexibility and maintainability through loose coupling between services and clients
  ○ many frameworks for various platforms and languages
  ○ less tool support required
Hardware Requirements

- low power consumption
- low cost
- wired or wireless network connectivity
- always-on operation
- reliability
- ease of software development
- active community support
Server Hardware Spectrum

- Conventional servers: x86 (or low-power Atom, C7), several GB RAM, 30-200 W, US$200+, highly extensible
- Plug computers: usually ARM, 0.5+ GB RAM, 5-15 W, US$100+, extensible
- WiFi routers, NAS, and similar devices: ARM or embedded MIPS, 0-32 MB flash and 2-64 MB RAM, 1-5 W, US$40+, somewhat extensible (USB)
- Single-board embedded computers and microcontrollers: ARM, Atmel, or similar, < 1 MB memory, < 1 W,
Green Hardware: Routers and Friends

- Very active communities around these devices
- Can run uClibc-based Linux, e.g., OpenWrt
- Can run on NiMH batteries, 4h in WLAN client mode
OK, cool, but how to develop the desired application software?
What Doesn't Work

● Java/JVM: Java Micro Edition (ME) Connected Device Configuration (CDC) still based on 1.4.2 version of the language => does not support even lightweight modern REST and other frameworks

● .NET/Mono: recent versions (2.x) not (yet?) supported on ARM or MIPS
Better Choices

- C/C++ with cross-compilation and Boost and POCO libraries

- Interpreted/scripting languages
  - *Erlang*: works but need mid-range router
  - *Lua*: well supported and very small footprint
  - *Perl*: probably works
  - *PHP*: well supported, looks promising
  - *Python*: issues with packages but probably works
  - *Ruby*: memory footprint too large
Example in Lua
(fits in router's remaining flash mem)

* Configuration

sensors = {
  baumhart = {
    ts42i = {
      nitrogen = {
        no = {
          current = function()
            return read_sensor(device, 7)
          end,
          ...
        }
      }
    }
  }
}

* URL-to-resource mapping

function map_path_to_resource(path, resource)
pos = resource
  for word in string.gfind(path or "", "[^/]+") do
    pos = pos[word]
  end
  if type(pos) == "table" then
    header_ok()
  end
  print ...
end
Preliminary Lessons

- App developers should consider alternate solution stacks (besides Java, Mono) for embedded target platforms.
- Language designers should be more supportive of embedded target platforms.
- Framework architects should be more aware of the limitations of current language support on these targets.
- Most observations generalize to other types of distributed computing projects.
Future Work

● Systematic evaluation of the various language and framework combinations
  ○ web server performance using httperf and siege
  ○ light-weight memory profiling.

● Evaluate languages and frameworks for single-board embedded computers and microcontrollers

● Apply RESTful thinking to novel hardware architectures such as GPU

● Study “true cost of computing” over entire life of device
Acknowledgments

- Loyola University Chicago's Graduate School and Center for Urban Environmental Research and Policy for internal grant support
- Sergio Fanchiotti for valuable discussions

Contact:
http://laufer.cs.luc.edu
http://etl.luc.edu
Questions?