TSP

XMLRPC command channel

Ruby TSP consumer library

Stephane GALLES stephane.galles@free.fr

Presentation content

- XMLRPC channel in TSP
- Why Ruby + TSP
- Ruby consumer library for TSP
- TSP meets the Web

XMLRPC channel

- Original author Frederik DEWEERDT
- Breaks dependency on ONCRPC channel
- Almost all the new languages have an
  - XMLRPC library
  - not necessarily an ONCRPC library

XMLRPC channel

- TSP is a specification, not an implementation
- Before XMLRPC channel
  - ONCRPC
  - wrapping of C implementation
- XMLRPC channel allows to implement the TSP specification with most languages

XMLRPC channel

- In order to compile the TSP C provider library with XMLRPC channel activated :
  - Install **xmlrpc-c** library compiled with **libwww** support
  - Compile the TSP C Provider library with cmake option : **-DBUILD_XMLRPC=ON**
- XMLRPC channel starts on port 8000

Why Ruby + TSP

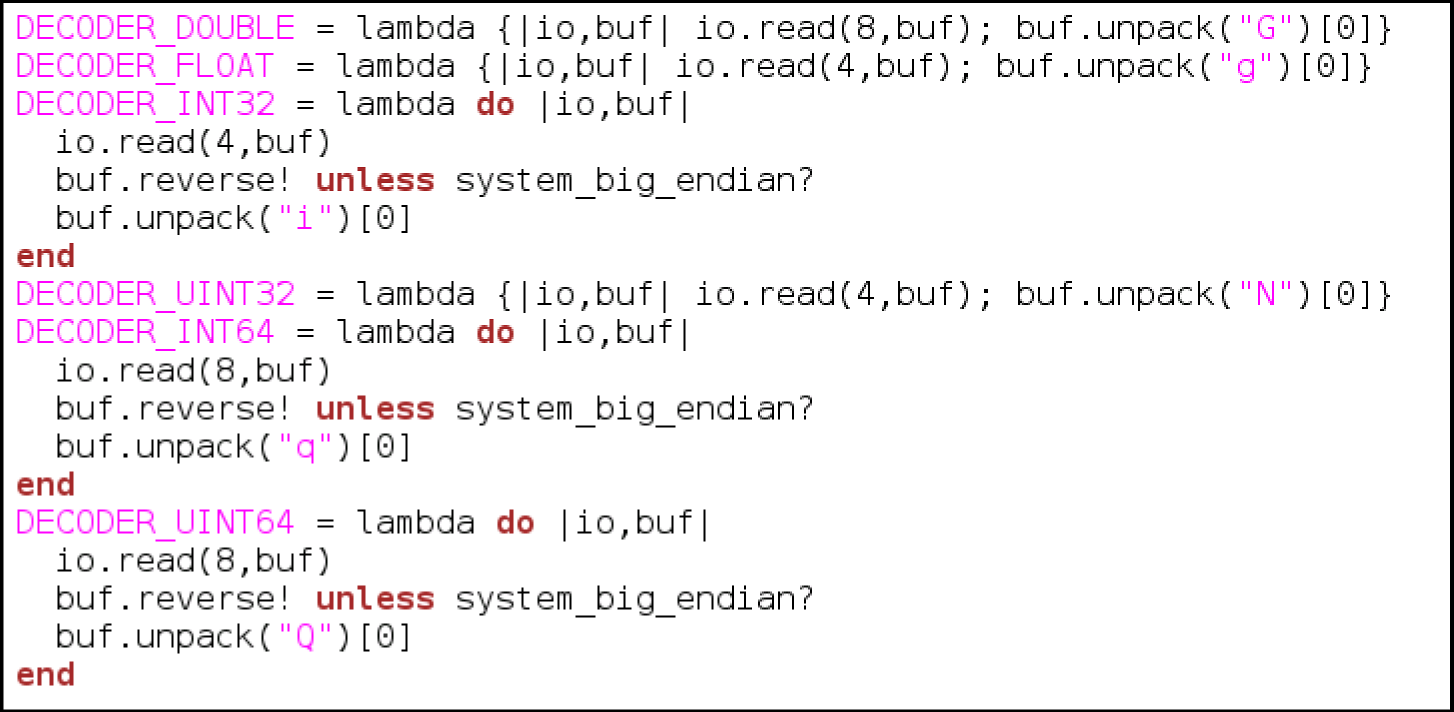
- Ruby is a very nice OO scripting language !
- To code a complete TSP consumer was a very good way to start to program in Ruby
  - Use of the Ruby XMLRPC standard library
  - Multithreading
  - Socket
  - Platform independent byte stream decoding

Ruby TSP consumer library

- Full Ruby implementation of a TSP consumer
- Power of the ruby core library and idioms kept the code length short

Ruby TSP consumer library

- A very nice surprise in the Ruby code library :
  - An API to decode in a platform independent way a stream of bytes :



```
DECODER_DOUBLE = lambda {|io,buf| io.read(8,buf); buf.unpack("G")[0]}
DECODER_FLOAT = lambda {|io,buf| io.read(4,buf); buf.unpack("g")[0]}
DECODER_INT32 = lambda do |io,buf|
  io.read(4,buf)
  buf.reverse! unless system_big_endian?
  buf.unpack("i")[0]
end
DECODER_UINT32 = lambda {|io,buf| io.read(4,buf); buf.unpack("N")[0]}
DECODER_INT64 = lambda do |io,buf|
  io.read(8,buf)
  buf.reverse! unless system_big_endian?
  buf.unpack("q")[0]
end
DECODER_UINT64 = lambda do |io,buf|
  io.read(8,buf)
  buf.reverse! unless system_big_endian?
  buf.unpack("Q")[0]
end
```

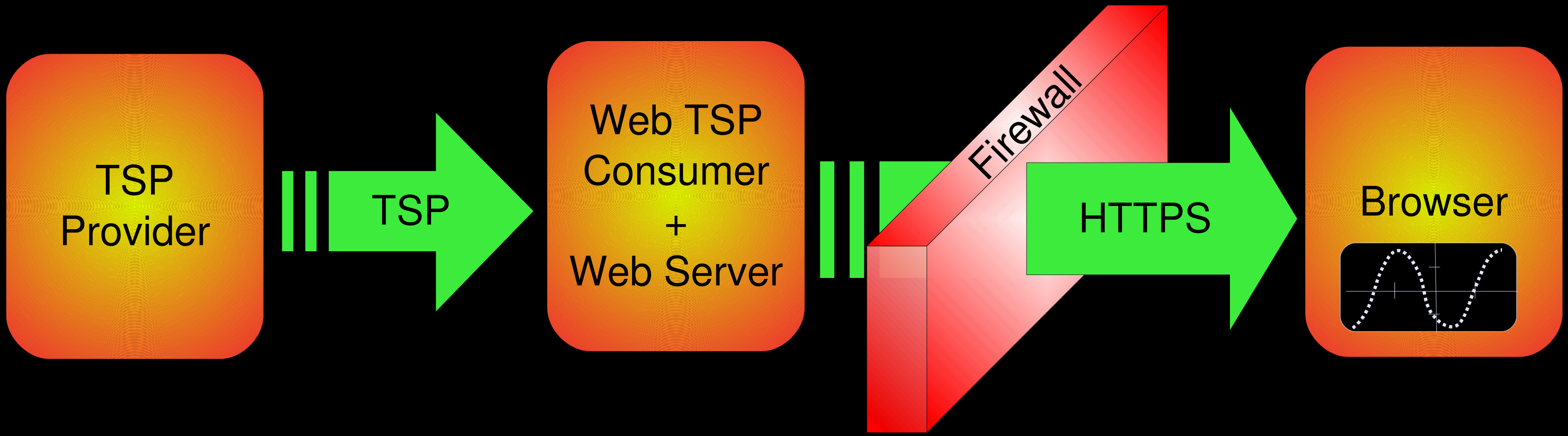
TSP meets the Web

- What about being able to monitor data connected to a TSP provider trough the Internet
- TSP is a great protocol but
  - It is not firewall friendly
  - Anyway, you don't really want to try a real time data connection to a TSP provider via Internet
- We only need the GUI to be distributed, not the consumer itself

TSP meets the Web

- For a distributed GUI, the obvious answer is a Web application
  - can be safely accessed trough a firewall
  - data can be encrypted with SSL (HTTPS)
  - Not need to install anything, a browser is enough
  - OS agnostic

TSP meets the Web



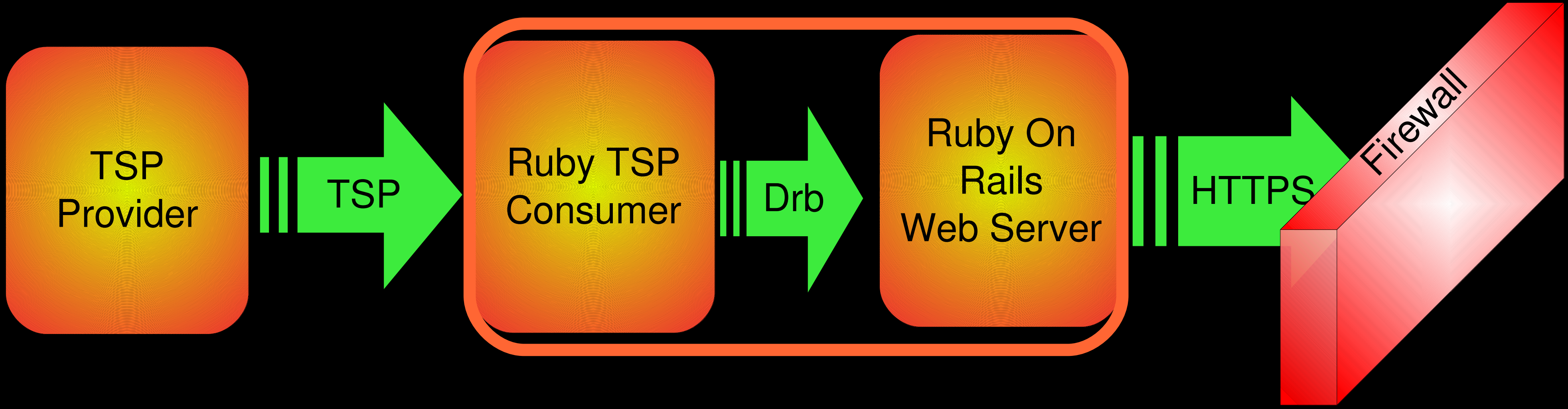
TSP meets the Web

- With a Ruby consumer library, the next natural step was to use this library in a **Ruby On Rails** program
- Ruby On Rails is a very productive framework to develop web applications
- Ruby On Rails basically is what made Ruby be famous, whereas Ruby itself is older than JAVA

TSP meets the Web

- A Ruby On Rails application is stateless
- We need a stateful Ruby consumer that keeps a buffer of recent symbol data
- The Ruby On Rails process can use Drb, the Ruby distributed method call protocol in order to fetch the symbol data

TSP meets the Web


TSP Provider
TSP
Ruby TSP Consumer
Drb
Ruby On Rails Web Server
HTTPS
Firewall