Erlang Training and Consulting Ltd

A Comparative Evaluation of

Imperative and Functional Implementations of the IMAP Protocol

ACM SIGPLAN Erlang Workshop
Victoria (BC), Canada
September 27th, 2008

Francesco Cesarini,
Viviana Pappalardo
Erlang Training & Consulting

Corrado Santoro
University of Catania

Agenda

- Purpose
- IMAP Protocol Overview
- Evaluation Criteria
- Evaluated Solutions
- Result and Conclusions

Purpose

Comparative analysis of several IMAP implementations evaluating

- Performance
- Capability to meet user requirements
- Effort needed to develop the library

We use the metrics

- Source Lines Of Code
- Functionality of primitives
- Memory Usage
- Execution Time

IMAP Protocol Overview (1/2)

Client-Server protocol communication

- Standardized by means of the RFC3501

POP3 heir

- Data storing and email processing on the remote server
- Email are not loaded locally by the client

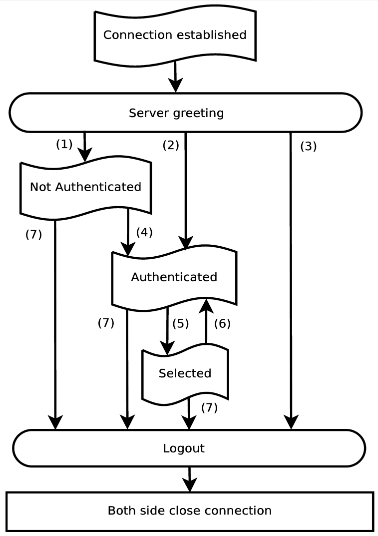
Request-Reply model

- The client starts each activity sending a request
- The server generates a reply carrying data required by client's query

IMAP4 Session

- FSM client sends commands in some specific order
- When request are successful, client enters in a new state

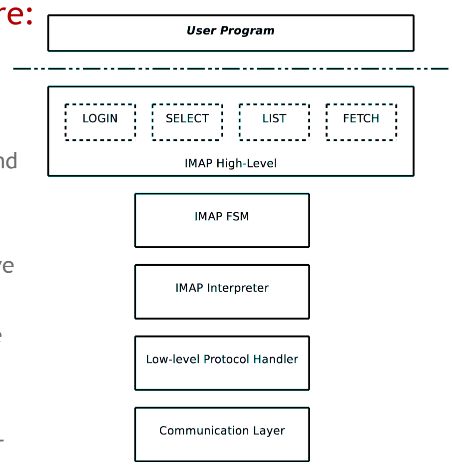
IMAP Protocol Overview (2/2)



/* **LOGIN INTERACTION** */
C: A001 LOGIN username password
S: A001 OK LOGIN Ok
/* **SELECT INTERACTION** */
C: A002 SELECT INBOX
S: * FLAGS (Junk NonJunk ...
S: * OK [PERMANENTFLAGS (Junk NonJunk ...
S: * 4270 EXISTS
S: * 0 RECENT
S: * OK [UIDVALIDITY 1186814135] Ok
S: * OK [MYRIGHTS "acdilrsw"] ACL
S: A002 OK [READ-WRITE] Ok
/* **FETCH INTERACTION** */
C: 4 fetch 4 body[header.fields (date subject)]
S: * 4 FETCH (BODY[HEADER.FIELDS (date subject)] {54}
Subject: es
Date: Sat, 16 Dec 2006 18:19:45 +0100)
S: 4 OK FETCH complete

Architecture of an IMAP Client Library

- Basic IMAP client library architecture:
  - ***Communication Layer***, handles socket connectivity.
  - ***Low-level IMAP protocol handler***, managing request-reply communication (tags handling and untagged/tagged response identification)
  - ***IMAP interpreter***, parsing role (transforms server's reply into programming language native types)
  - ***IMAP FSM***, handles IMAP's FSM (manages state transitions)
  - ***IMAP high-level interface***, implements the various IMAP commands by means of *IMAP low-level interface's services* and *IMAP parsing*.



Evaluation Criteria (1/2)

Number of source lines of code (SLOC)

- No blank lines and comments
- Count the numbers of lines of all library's source files
- "The more the SLOC the more the time required to write the software and to test it."

Functionality of primitives

- Qualitative parameters...
- ...to make sense of the capability of the library to provide a complete and transparent implementation of the IMAP commands
- Related to the number and type of function/method calls

Evaluation Criteria (2/2)

Memory

- Performance parameter
- Expresses the memory usage in a language specific IMAP library at run-time
- Test program: logging and fetching a bunch of messages

Execution-Time

- Performance parameter
- Test program: executing different commands and **measuring** time required by each library to perform its activities.

Evaluated Solutions (1/5)

Erlang Library

- Supervisor architecture, gen_server behaviour
- Stand-alone application managing the most important IMAP commands
- TCP/SSL connection support
- IMAP Protocol handling and Parsing activity carrying out Erlang native types
- Three modules:
  - **im_client** (front-end interface)
  - **scanning** (lexical analysis)
  - **parsing** (parses server's response)
- **im_client** module's outcome: *{ResultValue ('ok', 'not' or 'bad'), ParsedReply}*, ParsedReply is the server's reply data
- Check socket connection and re-instantiate it if needed, preventing brutal closing

Evaluated Solutions (2/5)

Java Library

- JavaMail package, provided officially by SUN Microsystems
- **Platform-independent and multiprotocol** (it supports IMAP and POP3 either) framework to implement email and messaging applications
- New objects
  - ***Session***, to retrieve the proper ***Store*** object, ***Store*** object performs socket connection mechanism, ***Folder*** object models email messages folder (r-w and r-o access mode), ***Message*** object models an email message.
- Two policies
  - "***getXXX***" method of Message object, to retrieve specific message's data (e.g. text,flags...), ***"getMessages"*** (cache all messages locally) and allows client to keep data readily available, therefore it is closer to POP3 than IMAP.

Evaluated Solutions (3/5)

C# Library

- Uses native library of .Net platform
- Three main source files:
  - **ImapBase.cs** (handles low-level client-server connection),
  - **Imap.cs** (protocol's engine, handles IMAP commands),
  - **ImapException.cs** (manages protocol's fault and internal error)
- Custom policy to fetch email message, three methods:
  - ***FetchPartHeader*** fetches only the Header section of the original email message,
  - ***FetchPartBody*** fetches only the Body section of the original email message,
  - ***FetchMessage*** performs a complete parsing and retrieve the whole message, generates an XML file such as output of its parsing activity. Client program must re-interpreter XML file to get meaningful data.

Evaluated Solutions (4/5)

Python Library

- Transparency in function declaration: 1:1 mapping of IMAP commands to methods.
- Authentication mechanism supported (IMAP AUTHENTICATE command)
- Client program must create the IMAP command (library manages Tag numbering only)
- No Parsing...client program must parse server's reply
- All methods returns a tuple: *{ Tagged response, Untagged Response }*

Evaluated Solutions (5/5)

Ruby Library

- Transparency in function declaration: 1:1 mapping of IMAP commands to methods.
- Authentication mechanism supported (IMAP AUTHENTICATE command)
- Client program must create the IMAP command (library manages Tag numbering only)
- Partial Parsing activity, only FETCH /BODYSTRUCTURE/BODY[TEXT]/BODY[section]/ENVELOPE replies are parsed
- Al methods returns a native Ruby types containing query's sequence_number and a tuple *{key,value}*, the former is FETCH command keyword(FLAGS...TEXT) whereas the latter is associated data.

Results – SLOC (1/4)

Languages	Lines
Erlang	1,189
Python	472
Ruby	1,612
C#	1,089
Java	n/a

Erlang and Ruby perform a full parsing of server's replies:

- **Erlang Parser** counts **818 lines** of code
- **Ruby Parser** counts **1048 lines** of code

Results – Functionality of primitives (2/4)

Command/ Primitive	Erlang	Python	Ruby	C#	Java
LOGIN	Yes	Yes	Yes	Yes	Yes
SELECT	Full parsing	No parsing	Partial parsing	No parsing	Internal w/full parsing
FETCH	Parametric Query Flexible expression of range Full parsing	Parameters as string Ranges as string No parsing	Parameters as string Range or single messages Partial parsing	Only some queries supported Single messages Partial parsing	Encapsulated Queries Single Messages Full parsing
LIST	Full parsing	No parsing	Full parsing	n/a	Full parsing Supported in current context

Results – Amount of memory (3/4)

Language	Total	Code	Data/Stack
Erlang	8,056	2,868	4,656
Python	6,748	4,400	1,684
Ruby	14,942	4,332	10,386
C#	18,800	11,148	2,748
Java	212,852	14,428	190,060

© 2008 - Erlang Training and Consulting

Results – Execution Time (4/4)

Command/ Primitive	Erlang	Python	Ruby	C#	Java
LOGIN	28.6	45.7*	26.1	32.7	30.7
SELECT	2.5	2.1*	44.3	13.2	4.5
FETCH BODY[TEXT]	43.0	41.1*	80.2	40.0	40.2
FETCH BODY[HEADER]	2.5	0.2*	44.5	5.3	2.9
FETCH BODY[HEADER.FIELDS]	2.3	0.2*	43.0	n/a	1.3*
FETCH BODYSTRUCTURE	1.8	0.2*	80.2	n/a	1.2*
FETCH (ENVELOPE SIZE BODY.PEEK[HEADER.FIELDS] ...)	n/a	n/a	n/a	n/a	1.9
LIST	0.7	0.9*	52.1	n/a	6.3

* = command executed without any parsing of the response

Results Summary

Full Parsing *(Erlang Library vs Ruby Library)*

- Erlang implementation outdoes Ruby library features, indeed its performances (such as Memory consumption, Execution-time) are better than values estimated for Ruby.
- Erlang library counts less lines of code of Ruby implementation, as demonstrated through SLOC analysis.

Partial Parsing *(Java Library vs C# Library)*

- Java implementation consumes huge amount of memory.
- Execution-time are comparable even if Java has better performances in the FETCH command handling, probably it depends on internal proprietary implementation.

NO Parsing *(Python Library)*

- Python has best performances (memory consumption and execution-time) but they depend on the lack of parsing activity and Python JVM characteristics.

Conclusions

The aim is...

– ...to evaluate Erlang ability to meet requirements of productivity and performance for a distributed system

We conclude that the Erlang library...

– ...can deliver the requirements of functionality and performance for a distributed system
– has high execution-time without high memory cost