Messaging with Erlang and Jabber

Erlang User Conference '04

21st. October 2004

Mickaël Rémond <mickael.remond@erlang-fr.org>

www.erlang-projects.org

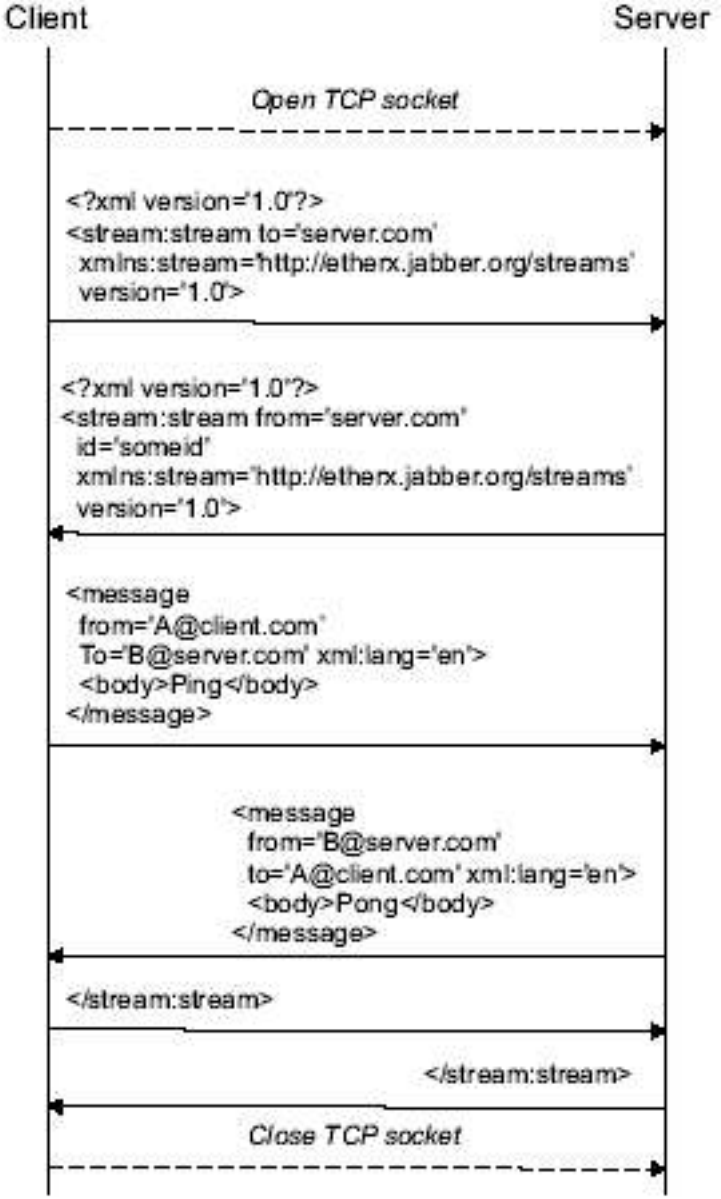
What are XMPP and Jabber ?

- **XMPP** stands for **eXtensible Messaging & Presence Protocol**
- XMPP is a **generic and extensible** messaging protocol based on XML. It is now an IETF standard
- **Jabber** is an **Instant Messaging** protocol that rely on XMPP
- Very active community:
  - Several **server** implementations of XMPP and Jabber
  - Several **client** software and libraries

ERLANG

What does the XMPP protocol looks like ?

- **Interleaved XML document streams**: Client and serverstreams form an XML document.
- First level tag: **<stream>**
- Three types of second levels tags:
  - **message**: asynchronous communications
  - **iq**: Synchronous communications
  - **presence**: presence and status data



Example XMPP telnet session

Step 0: `telnet localhost 5222`

Step 1: Open XMPP stream

Client sends:

```
<?xml version='1.0'?>
<stream:stream
  xmlns:stream="http://etherx.jabber.org/streams"
  to="localhost" xmlns="jabber:client">
```

Server answers:

```
<?xml version='1.0'?>
<stream:stream xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'
  id='3694886828' from='localhost'>
```


ERLANG

Example XMPP telnet session

Step 2: Login

Client send login informations:

```
<iq type='set' id='auth'>
  <query xmlns='jabber:iq:auth'>
   <username>mremond</username>
   <password>azerty</password>
   <resource>TelnetClient</resource></query></iq>
```

Server confirms login:

```
<iq type='result' id='auth'/>
```


ERLANG

Example XMPP telnet session

Step 2: Login

The server can returns an error on failed authentication for example:

```
<iq type='error' id='auth'>
  <query xmlns='jabber:iq:auth'>
   <username>mremond</username>
   <password>D</password>
   <resource>TelnetClient</resource></query>
  <error code='401' type='auth'>
   <not-authorized
    xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error></iq>
```


ERLANG

Example XMPP telnet session

Step 3: Presence

Client sends presence data

```
<presence/>
```

If applications in our client roster are online, our telnet client receives presence packets:

```
<presence from='mremond5@localhost/tkabber'
   to='mremond@localhost/TelnetClient'>
   <priority>8</priority>
  </presence>
```


ERLANG

Example XMPP telnet session

Step 3: Presence

From here, our telnet client is visible online by other clients:

ERLANG

Example XMPP telnet session

Step 4: Sending a message

Client sends a message to another user / application:

```
<message to='mremond5@localhost'>
  <subject>Hello</subject>
    <body>I am chatting with a Telnet client !
    </body>
  </message>
```

No answer from the server

ERLANG

Example XMPP telnet session

Step 5: Receiving a message

We can receive messages from other clients:

```
<message from='mremond5@localhost/tkabber'
  to='mremond@localhost/TelnetClient'
  type='chat'
  xml:lang='fr-FR'>
   <body>Answer from the other side</body>
   <x xmlns='jabber:x:event'>
    <offline/>
    <delivered/>
    <displayed/>
    <composing/>
  </x></message>
```


ERLANG

Example XMPP telnet session

Step 6: Presence update

Presence informations are updated (For example here, when a user in our roster disconnect):

```
<presence from='mremond5@localhost/tkabber'
  to='mremond@localhost/TelnetClient'
  type='unavailable'/>
```


ERLANG

Example XMPP telnet session

Step 7: Closing the XML stream

Client closes XML client stream tag (This end a valid XML document):

```
</stream:stream>
```

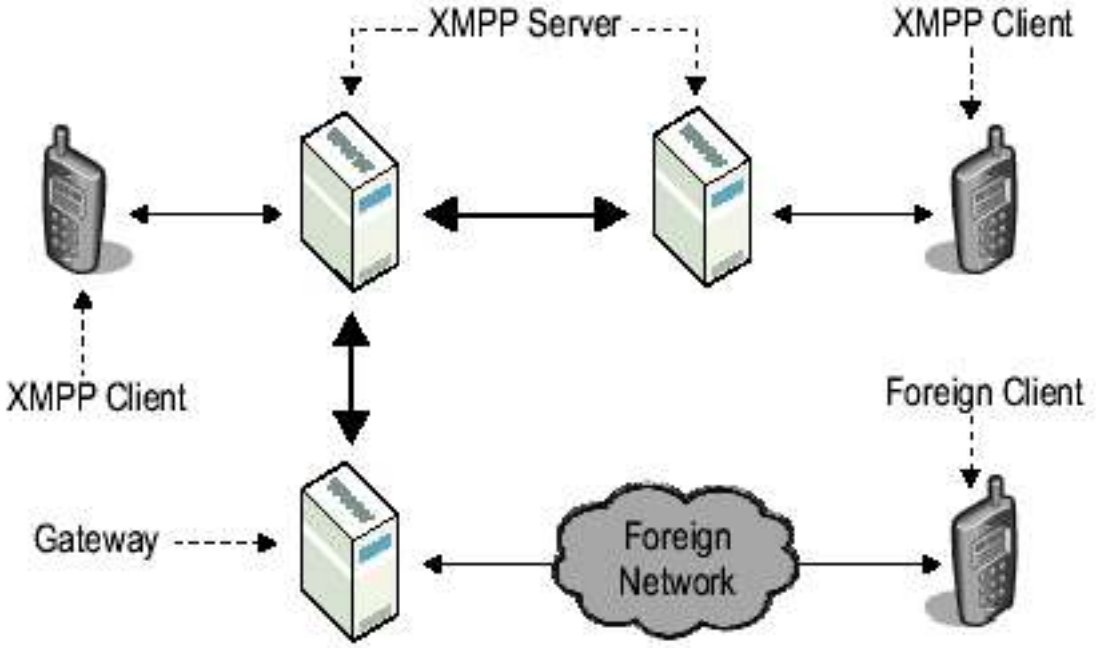
Server then closes XML stream tag and disconnect (This end the second XML document):

```
</stream:stream>
```


ERLANG

XMPP bus design

- XMPP rely on a naturally distributed architecture (**No central server** !)
  - Includes **server to server communications** (with domain routing between servers)
  - Includes **gateways** to various other protocols



ERLANG

XMPP protocol extensions

- XMPP is an extensible protocol (with XML namespaces) that supports many interesting features:
  - Service **discovery** and browsing
  - **Publish & Subscribe**: Allow 1 to n communications.
  - **Reliable messages**: can add confirmations delivery.
  - **Message queues** for offline applications and **messages expiration**.
  - **Users or applications search**.
  - **Geolocalisation**.
  - **Message archiving**.
  - ...

ERLANG

Fun things to do with XMPP and Erlang

- **Implementation**: XMPP and Jabber servers are **massively concurrent**: a Jabber server must handle huge community of users.
  - A Jabber server in Erlang makes sense to handle massive concurrency.
  - It can **prove** Erlang reliability, scalability, ability to handle concurrency
- **Extensions**: XMPP server protocol is build around a complete XML API for **client to server** and **server to server** communications:
  - Developping **Erlang software agents** that plug on the bus is easy: Erlang expressiveness.
  - It allows to use the bus as a **mediation layer** between Erlang and non-Erlang software (Kind of web service but more simple and powerful)
  - Several levels of interaction for Erlang-based extensions: Plug-in in the XMPP bus with a **service protocol** or XMPP client connected to the bus (**client protocol**)
- **Use**: XMPP can be used as an instant messaging platform or integrated with other communication services.

ERLANG

Implementing an XMPP server in Erlang

ejabberd

- ejabberd is an Erlang-based XMPP server implementation.
- It has been designed to support **clustering, fault-tolerance and high-availability**.
- It supports many features and extensions of the Jabber protocol:
  - Built-in **Multi-User Chat** service
  - **Distributed database** (Mnesia)
  - Built-in **IRC** transport
  - Built-in **Publish-Subscribe** service
  - Support for **LDAP** authentification
  - Service **discovery**
- It is more **scalable** than the most used open source implementation (Jabberd1.4 and Jabber2).

ERLANG

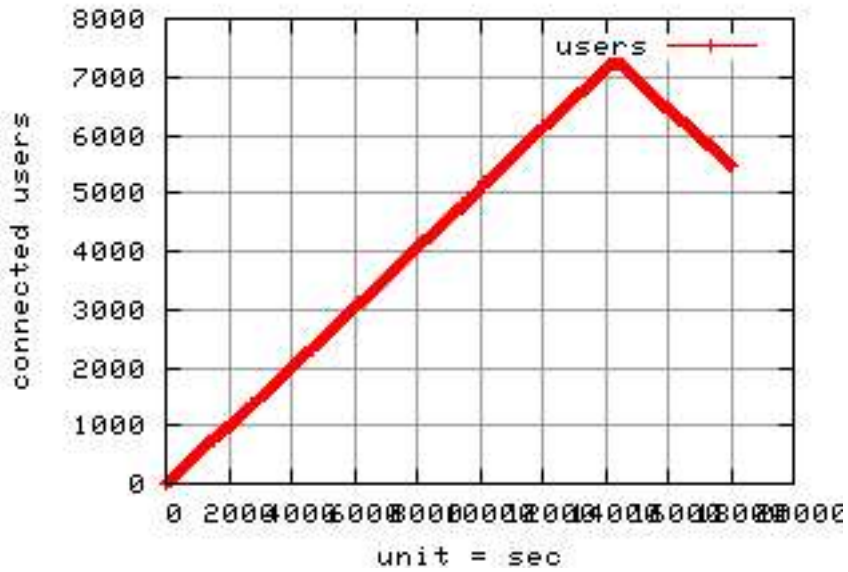
Benchmarks: how does ejabberd perform ?

- Jabber benchmarks realized with the **Tsunami** benchmarking tool.
- Ejabberd performs very well:
  - It can route more than **2000 simple messages per second** (on a single CPU 1,5 Ghz machine).
  - It can handle a huge number of concurrent users on a single machine (Has reached **7000 users** connected without troubles. We need to push the test further).
  - This good performance is achieved while being the most featureful and stable server XMPP.
  - ejabberd design can be improved to achieve better performance.

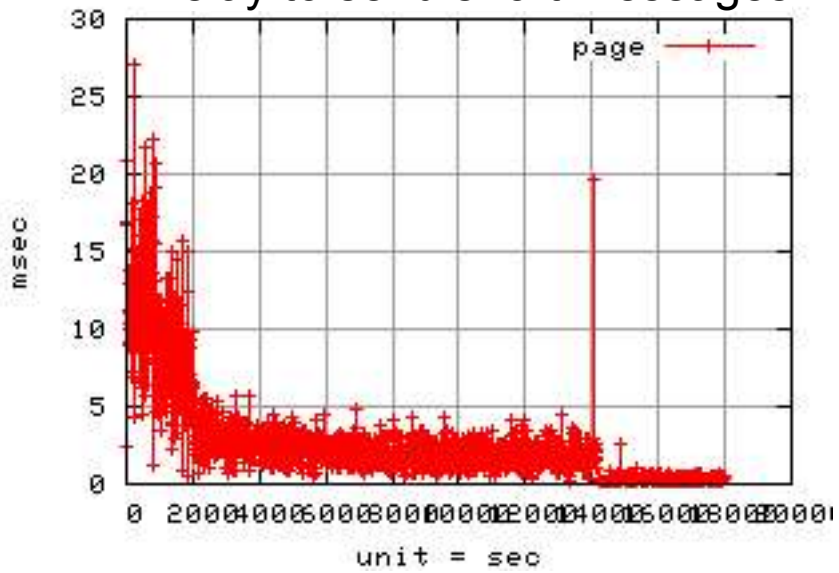
ERLANG

ejabberd

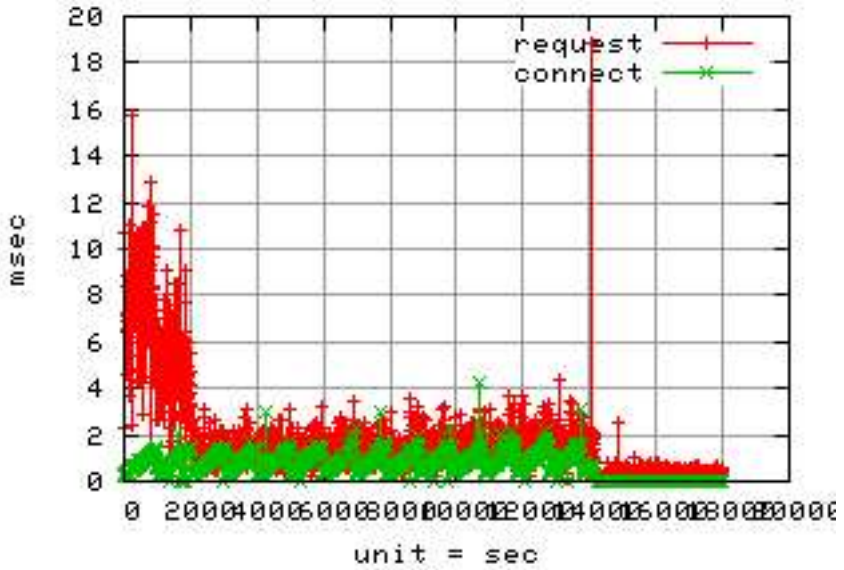
simultaneous users



Delay to send short messages



Response time and connection establishement



ERLANG

The benchmark server

- CPU: Bi-Xeon 2,4 Ghz
- Ram: 1 Go
- Operating System: Linux

Note: The second CPU was not used as only one ejabberd instance was running on the server.

Developing XMPP clients in Erlang

Jabberlang: Helper library to write XMPP client in Erlang

- Jabberlang is a **client library** to write XMPP client in Erlang.
- It can be used both a **library** or as Erlang **behaviour**.
- It allow to write **XMPP services** in Erlang.
- Can be used for inter Erlang programs communication. Several options for inter applications communications:
  - **term to binary** <-> binary_to_term
  - UBF
  - ASN.1

ERLANG

Jabberlang features

- Auto registration
- Subscriptions management
- Roster management
- Message and IQ packet support

ERLANG

Jabberlang: Simple echo (library version)

```
-module(xmpp_echo).
-export([start/1, presence/4, message/7]).
start(Host) ->
    {ok, XMPP} = xmpp:start(Host),
    xmpp:set_login_information(XMPP, "mremond",{password,"azerty"},"ErlangEcho"),
    xmpp:set_callback_module(XMPP, ?MODULE),
    xmpp:connect(XMPP).

%% Ignore presence packets
presence(_XMPP, _Type, _Attrs, _Elts) ->  ok.

%% Echo: Reply to messages with the same message
message(XMPP, Type, From, Subject, Body, Attrs, _Elts) ->
    xmpp:message(XMPP, From, Type, Subject, Body).
```


ERLANG

Jabberlang: Simple echo (behaviour and attributes 1/2)

```
-module(xmpp_echo_behaviour).
-behaviour(gen_xmpp_client).

%% XMPP configuration attributes
-host("localhost").
-port(5222).
-username("mremond").
-authentication({password,"azerty"}).
-resource("Erlang echo behaviour").

%% Optional:
-include("xmpp.hrl").

%% Behaviour callbacks
-export([init/2,
     presence/4,
     message/7]).

-export([start/0]).
```


ERLANG

Jabberlang: Simple echo (behaviour and attributes 2/2)

```
%% Module API
start() ->
    gen_xmpp_client:start_link(?MODULE, [], []).

%% gen_xmpp_client callbacks
init(Args, State) ->
    {ok, State}.

%% Ignore presence packets
presence(_XMPP, _Type, _Attrs, _Elts) ->
    ok.

%% Echo: Reply to messages with the same message
message(XMPP, Type, From, Subject, Body, Attrs, _Elts) ->
    xmpp:message(XMPP, From, Type, Subject, Body).
```


ERLANG

Jabberlang: Simple echo (behaviour and no attributes 1/2)

```
-module(xmpp_echo_behaviour2).
-behaviour(gen_xmpp_client).

%% Optional:
-include("xmpp.hrl").

%% Behaviour callbacks
-export([init/2,
     presence/4,
     message/7]).

-export([start/0]).
```

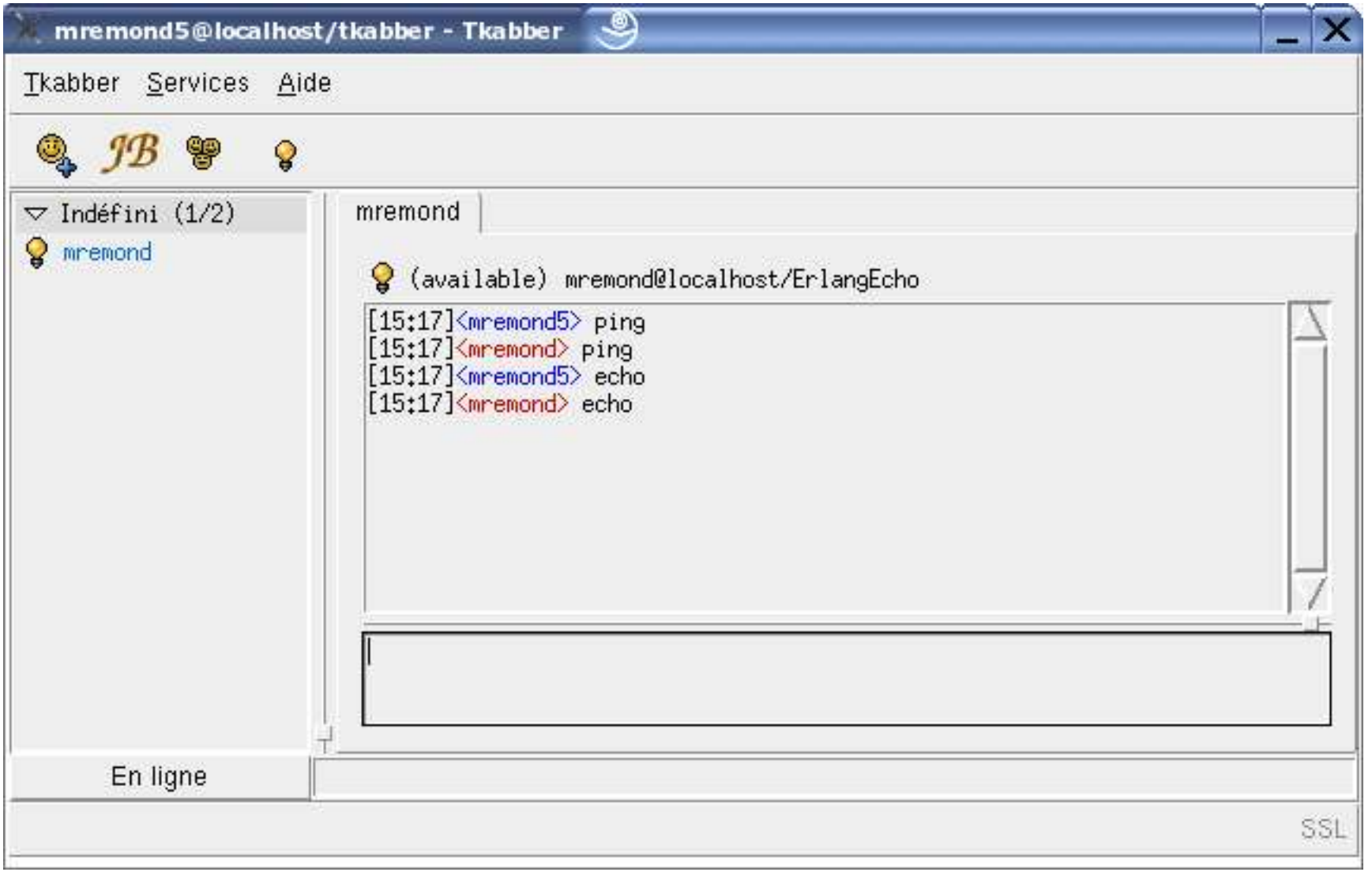

ERLANG

Jabberlang: Simple echo (behaviour and no attributes 2/2)

```
%% Module API
start() ->
    Args = [
        {host, "localhost"},
        {port, 5222},
        {username, "mremond"},
        {authentication, {password, "azerty"}},
        {resource, "Echo XMPP behaviour 2"}],
    gen_xmpp_client:start_link(?MODULE, Args, []).

%% gen_xmpp_client callbacks
init(Args, State) ->
    {ok, State}.

%% Ignore presence packets
presence(_XMPP, _Type, _Attrs, _Elts) ->
    ok.

%% Echo: Reply to messages with the same message
message(XMPP, Type, From, Subject, Body, Attrs, _Elts) ->
    xmpp:message(XMPP, From, Type, Subject, Body).
```


ERLANG

Using our simple echo client from a Jabber client



ERLANG

Erlang to Erlang communication through XMPP

- We launch the Erlang program two time as two differents resources
- The program receives messages, extracts Erlang term and increments the value and send it back to where it came from, and so on (ping – pong).

```
{ok, XMPP}  = xmpp_erlang_example:start("ping").
{ok, XMPP2} = xmpp_erlang_example:start("pong").
xmpp_erlang_example:initialize_counter_pong(XMPP, 1).
```


ERLANG

Erlang to Erlang through XMPP 1/2

```
-module(xmpp_erlang_example).
-behaviour(gen_xmpp_client).

%% XMPP configuration attributes
-host("localhost").
-username("mremond").
-authentication({password,"azerty"}).

%% Optional:
-include("xmpp.hrl").

%% Behaviour callbacks
-export([init/2,
     presence/4,
     message/7]).

-export([start/1,
     initialize_counter_pong/2]).

%% Module API
start(Resource) ->
    gen_xmpp_client:start_link(?MODULE, [{resource,Resource}], []).

initialize_counter_pong(XMPP, Counter) ->
    xmpp:message(XMPP, "mremond@localhost/pong", "chat", "", encode
(Counter)).
```


ERLANG

Erlang to Erlang through XMPP 2/2

```
%% gen_xmpp_client callbacks
init(Args, State) ->
    {ok, State}.

%% Ignore presence packets
presence(_XMPP, _Type, _Attrs, _Elts) ->
    ok.

%% Reply to increment
message(XMPP, Type, From, Subject, Body, Attrs, _Elts) ->
    Value = decode(Body),
    io:format("Value: ~p~n", [Value]),
    xmpp:message(XMPP, From, Type, Subject, encode(Value+1)).

%% Take term and return encoded string
encode(Term) ->
    httpd_util:encode_base64(binary_to_list(term_to_binary(Term))).

%% Take String and return term
decode(String) ->
    binary_to_term(list_to_binary(httpd_util:decode_base64(String))).
```


ERLANG

Extending the services offered by a standard Jabber server

J-EAI: an XMPP based integration tool

- J-EAI is an **Enterprise Application Integration** tool.
- It is intended to **control and organize** data streams in a given information system.
- It allow **transformation, routing, queueing** of all the data exchanges between applications in a given company or outside a company.

ERLANG

J-EAI: an XMPP based integration tool

- J-EAI supports handy features such as:
  - **Messages trace**, to know what is exchanged between applications,
  - **Error message trace** (« hospital »), to be able to take actions upon message problems,
  - **Connectors** to existing protocols and applications,
  - Enhanced **publish & subscribe** mechanism,
  - Full control from a central **console**,

ERLANG

Integrating XMPP with other applications

Integration in Web application

- XMPP can turn web applications into **event-based systems**. You are not anymore limited to pull mode. You can send events to the web browser.
- Those features are being implemented as **browser extensions**.
- **Metafrog**, a project tracker platform, will be the test-bed for this development approach:
  - Go to http://metafrog.erlang-projects.org
  - Look at light bulbs

ERLANG

SIP and XMPP integration

- **SIP / XMPP gateway**: Running together ejabberd and yxa to share user base and presence information. Can provide:
  - Integration between SIP and XMPP client. It could be possible to see if someone is one the phone from a Jabber client.
  - Missed called or voice mail could be moved from the SIP protocol to other (mail or XMPP).

ERLANG

Using XMPP for Erlang distribution ?

- Using XMPP for **Erlang distribution** could allow to develop distributed applications running through the Internet.
- This approach can solve some **security** aspects: Rosters configuration can decide if two process are allowed to exchanged messages.
- **SSL** is supported.
- Performance penalty: only relevant for non heavy-loaded critical **message passing** applications.
- Application **design** in such a way could switch to Erlang standard distribution to get more performance.

ERLANG

Thank you !

- **Alexey Shchepin**: Lead ejabberd developer
- **Jonathan Bresler**: Benchmarks
- **Nicolas Niclausse**: Benchmarks and lead Tsunami developer
- **Christophe Romain**: development on Erlang REPOS
- **Thierry Mallard**: Help for the conference
- **Catherine Mathieu**: Help for preparing the Erlang REPOS EUC 2004 CDROM

ERLANG

References

- http://www.jabber.org
- http://www.jabber.org/press/2004-10-04.php
- http://ejabberd.jabberstudio.org/
- http://www.erlang-projects.org/
- Erlang REPOS CDROM (see Erlang-projects website)

ERLANG

Messaging with Erlang and Jabber

Questions