Erlang and Java - a Happy Marriage

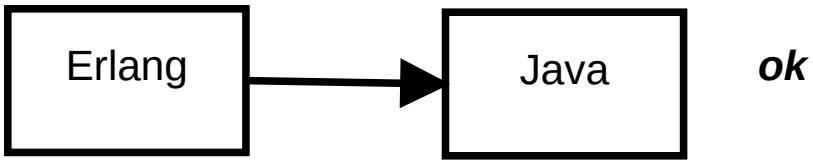
Lars-Åke Fredlund
Universidad Politécnica de Madrid, Spain

`https://github.com/fredlund/JavaErlang.git`

What is the talk about?

The JavaErlang *Erlang* library, which:

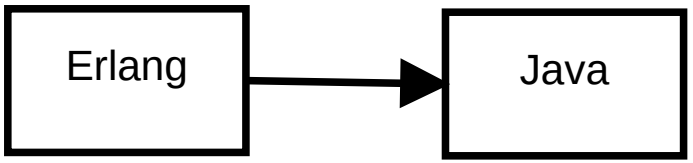
- makes it very easy to call Java from Erlang



What is the talk about?

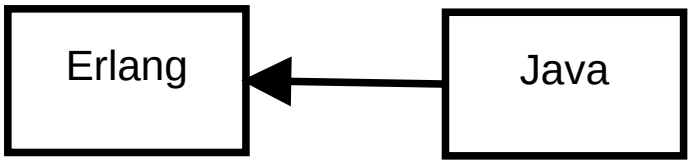
The JavaErlang *Erlang* library, which:

- makes it very easy to call Java from Erlang



ok

- does not help very much if you call Erlang code from Java (what would be useful? we need **your** input)



not much help

`https://github.com/fredlund/JavaErlang.git`

What is the problem we are solving?

What is the problem we are solving?

- **Testing** Java code (using QuickCheck)
  - Teaching Java based course at the Madrid Technical University
    10 (number of exercises) * 125 (student solutions) to correct
  - Web services using JSON schemas: `json-schema-validator`
  - GUIs: `selenium`

What is the problem we are solving?

- **Testing** Java code (using QuickCheck)
  - Teaching Java based course at the Madrid Technical University
    10 (number of exercises) * 125 (student solutions) to correct
  - Web services using JSON schemas: `json-schema-validator`
  - GUIs: `selenium`
- **No desire to write Java code** (we much prefer Erlang)

What is the problem we are solving?

- **Testing** Java code (using QuickCheck)
  - Teaching Java based course at the Madrid Technical University
    10 (number of exercises) * 125 (student solutions) to correct
  - Web services using JSON schemas: `json-schema-validator`
  - GUIs: `selenium`
- **No desire to write Java code** (we much prefer Erlang)

So … – strengths and weaknesses of the library?

What is the problem we are solving?

- **Testing** Java code (using QuickCheck)
  - Teaching Java based course at the Madrid Technical University
    10 (number of exercises) * 125 (student solutions) to correct
  - Web services using JSON schemas: `json-schema-validator`
  - GUIs: `selenium`
- **No desire to write Java code** (we much prefer Erlang)

So . . . – strengths and weaknesses of the library?

\+ Will make it very easy to call Java from Erlang (no Java code needed)

– May not be high-performing

Why not just use Jinterface? (pros)

+ Enables the programming of Java nodes that can communicate with Erlang nodes (Erlang distribution)
+ Easy message passing between Erlang and Java:
  - An Erlang node sends and receives messages containing normal Erlang values
  - A Java node sends and receives messages containing Java objects representing *marshalled* Erlang values
+ Works...

Why not just use Jinterface? (cons)

- *Lots* of **Java** boilerplate code needed for every program tested:

```
OtpErlangObject msg = msgs.receive();

if (msg instanceof OtpErlangTuple) {
  OtpErlangTuple tup = (OtpErlangTuple) msg;
  if (tup.arity() == 2) {
    OtpErlangObject tag = tup.elementAt(0);
    OtpErlangObject arg = tup.elementAt(1);
    if (tag instanceof OtpErlangAtom) {
      String stag = ((OtpErlangAtom) tag).atomValue();
      if (stag.equals("call_m")) {
          int i = ((OtpErlangInt) arg).intValue();
          m(i); // Finally call the method
```

- No support for communicating Java object references to Erlang
- Performance

What about Erjang?

- *Erjang is a virtual machine for Erlang, which runs on Java(tm).*

  `https://github.com/trifork/erjang/wiki`

Does it work?

Yes! It does actually work.

. . .

Larger systems like rabbitmq and riak can boot; and works for basic cases ... but it's not ready for prime time yet.

JavaErlang design & goals

- Built on top of Jinterface (keeping the good parts)
- No Java coding needed (no boilerplate code whatsoever)
- No pre-compilation of Java interfaces needed
  Uses Java reflection for a cleaner solution (easy maintenance)
- Communication of Java references to/from Erlang

JavaErlang design & goals

- Built on top of Jinterface (keeping the good parts)
- No Java coding needed (no boilerplate code whatsoever)
- No pre-compilation of Java interfaces needed
  Uses Java reflection for a cleaner solution (easy maintenance)
- Communication of Java references to/from Erlang
- *Enable safe and automatic garbage collection of Java objects whose references have been sent to Erlang*

Demo Time

Demo Time

- Starting Java:

```
> {ok,N} = java:start_node().
```

Demo Time

- Starting Java:

```
> {ok,N} = java:start_node().
```

- Creating a Java HashSet:

```
> HS = java:new(N,'java.util.HashSet',[]).
```

Demo Time

- Starting Java:

```
> {ok,N} = java:start_node().
```

- Creating a Java HashSet:

```
> HS = java:new(N,'java.util.HashSet',[]).
```

- Interacting with the Hashset:

```
> java:call(HS,add,[1]).
true
> java:call(HS,add,[1]).
false
> java:call(HS,add,[2]).
true
> java:call(HS,contains,[0]).
false
> java:call(HS,contains,[1]).
true
```

API (the `java` module)

```
start_node() → {ok,NodeId}
new(NodeId,Class,[Val]) → Obj
call(Obj,Method,[Val]) → Val
call_static(NodeId,Class,Method,[Val]) → Val
get(Obj,Field) → Val
set(Obj,Field,Val) → Val
get_static(NodeId,Class,Val) → Val
set_static(NodeId,Class,Field,Val) → Val
```

- `Class`, `Method` and `Field` represent Java class, method, and field names as Erlang atoms
- `Object` is the representation of a Java reference in Erlang
- `Val` is an object reference, or an Erlang value that can be interpreted as a Java value (integers, booleans, . . .)

Implementation Sketch – java:new(Node,Class,Args)

Implementation Sketch – java:new(Node,Class,Args)

Erlang node	Java node
1. if `Class` known goto 5	

Implementation Sketch – java:new(Node,Class,Args)

Erlang node	Java node

1. if `Class` known goto 5
2. **Java !** {'req-class-info',`Class`}

Implementation Sketch – java:new(Node,Class,Args)

Erlang node	Java node
1. if `Class` known goto 5	
2. **Java !** {'req-class-info',`Class`}	
	3. $Info \leftarrow lookup_{class}(\texttt{Class})$

Implementation Sketch – java:new(Node,Class,Args)

Erlang node	Java node
1. if `Class` known goto 5	
2. **Java !** {'req-class-info',`Class`}	
	3. $Info \leftarrow lookup_{class}(\texttt{Class})$
	4. **Erlang !** $map_{j \rightarrow e}(Info)$

Implementation Sketch – java:new(Node,Class,Args)

Erlang node	Java node
1. if `Class` known goto 5	
2. **Java !** {'req-class-info',`Class`}	
	3. $Info \leftarrow lookup_{class}(\texttt{Class})$
	4. **Erlang !** $map_{j \rightarrow e}(Info)$
5. $C \leftarrow$ $lookup_{cnstr}(\texttt{Class}, Args, Info)$	

Implementation Sketch – java:new(Node,Class,Args)

Erlang node	Java node
1. if `Class` known goto 5	
2. **Java !** {'req-class-info',`Class`}	
	3. $Info \leftarrow lookup_{class}(\texttt{Class})$
	4. **Erlang !** $map_{j \rightarrow e}(Info)$
5. $C \leftarrow$ $lookup_{cnstr}(\texttt{Class}, Args, Info)$	
6. **Java !** {'call-constructor',C,$Args$}	

Implementation Sketch – java:new(Node,Class,Args)

Erlang node	Java node
1. if `Class` known goto 5	
2. **Java !** {'req-class-info',`Class`}	
	3. $Info \leftarrow lookup_{class}(\texttt{Class})$
	4. **Erlang !** $map_{j \rightarrow e}(Info)$
5. $C \leftarrow$ $lookup_{cnstr}(\texttt{Class}, Args, Info)$	
6. **Java !** {'call-constructor',C,$Args$}	
	7. $Constr \leftarrow map_{e \rightarrow j}(C)$ $CArgs \leftarrow map_{e \rightarrow j}(Args)$

Implementation Sketch – java:new(Node,Class,Args)

Erlang node	Java node
1. if `Class` known goto 5	
2. **Java !** {'req-class-info',`Class`}	
	3. $Info \leftarrow lookup_{class}(\texttt{Class})$
	4. **Erlang !** $map_{j \rightarrow e}(Info)$
5. $C \leftarrow$ $lookup_{cnstr}(\texttt{Class}, Args, Info)$	
6. **Java !** {'call-constructor',C,$Args$}	
	7. $Constr \leftarrow map_{e \rightarrow j}(C)$ $CArgs \leftarrow map_{e \rightarrow j}(Args)$
	8. $Obj \leftarrow invoke(Constr, CArgs)$

Implementation Sketch – java:new(Node,Class,Args)

Erlang node	Java node
1. if `Class` known goto 5	
2. **Java !** {'req-class-info',`Class`}	
	3. $Info \leftarrow lookup_{class}(\texttt{Class})$
	4. **Erlang !** $map_{j \rightarrow e}(Info)$
5. $C \leftarrow$ $lookup_{cnstr}(\texttt{Class}, Args, Info)$	
6. **Java !** {'call-constructor',C,$Args$}	
	7. $Constr \leftarrow map_{e \rightarrow j}(C)$ $CArgs \leftarrow map_{e \rightarrow j}(Args)$
	8. $Obj \leftarrow invoke(Constr, CArgs)$
	9. **Erlang !** $map_{j \rightarrow e}(Obj)$

$lookup_{class}$	–	lookup class information using Java reflection
$map_{j \rightarrow e}(Obj)$	–	map Java values (object references) to Erlang representati
$map_{e \rightarrow j}(V)$	–	map Erlang values to Java values (Object ref)
$invoke$	–	invoke constructor using Java reflection

Under the hood: communicating object references

- A Java node translates object references to special Erlang tuples ("object tuples") when encoding replies ($map_{j \to e}$), and converts them back to object references in incoming messages ($map_{e \to j}$):

  `{'object'`, $ObjectId$, $ObjectRefCounter$, $ClassId$, $NodeId$`}`

  - $ObjectId$ is the object identifier (an integer)
  - $ObjectRefCounter$ is a per-object counter which is incremented every time a reference to the object is sent to Erlang (for GC)
  - $ClassId$ is the class identifier (an integer)
  - $NodeId$ is the Java Node identifier (an integer)

- The Java node maintains a mapping between object references and object tuples to implement the translation, **and** to ensure that objects are not garbage collected prematurely

Garbage Collection

- At the Erlang side the object reference counter is replaced with a new NIF "resource object":

$$\texttt{\{'object'}, ObjectId, ObjectRefCounter, ClassId, NodeId\texttt{\}}$$
$$\Rightarrow$$
$$\texttt{\{'object'}, ObjectId, \texttt{<<robj>>}, ClassId, NodeId\texttt{\}}$$

  where `<<robj>>` is a new NFI resource object

- When Erlang garbage collects the resource object, a custom destructor is called, which communicates to Java that the number of outstanding object references has decreased
- If the count is zero, the Java table entry for the object reference can be removed, and the corresponding Java object becomes a candidate for garbage collection
- No GC if the Erlang platform does not support NIF:s

Process and Threads

- The Java side of the library is multi-threaded – an Erlang process communicates with its own Java thread
- Good for some Java libraries – the Swing GUI library – where the same thread should invoke all operations

Creating "Erlang Classes

The API for some Java libraries require new classes:

- Let's create a Java Swing Button (showing "Hello")

```
Button = java:new(N,'javax.swing.JButton',["Hello"]),
java:call(Pane,add,[Button])
```

- To to change the button text from "Hello" to "World" when pressed one can provide an "action listener" object:

```
java:call(Button,setAction,[ActionListenerObj])
```

- **But** there is no suitable class for creating the action listener – in practice we have to *extend* the *abstract* class `AbstractAction`

Solution: “Proxy classes”

- We use the Javassist byte code manipulation library to enable implementation of Java classes in Erlang

Finishing the example using the JavaErlang API:

```
java_proxy:class
  (N,
   'actListen',                              %% New class name
   'javax.swing.AbstractAction',             %% Superclass
   [                                         %% New methods
    %%% Name, Method Type
    {{actionPerformed,['java.awt.event.ActionEvent']},
     fun actionPerformed/3}                  %% Implementation
   ]).
```

Action Function

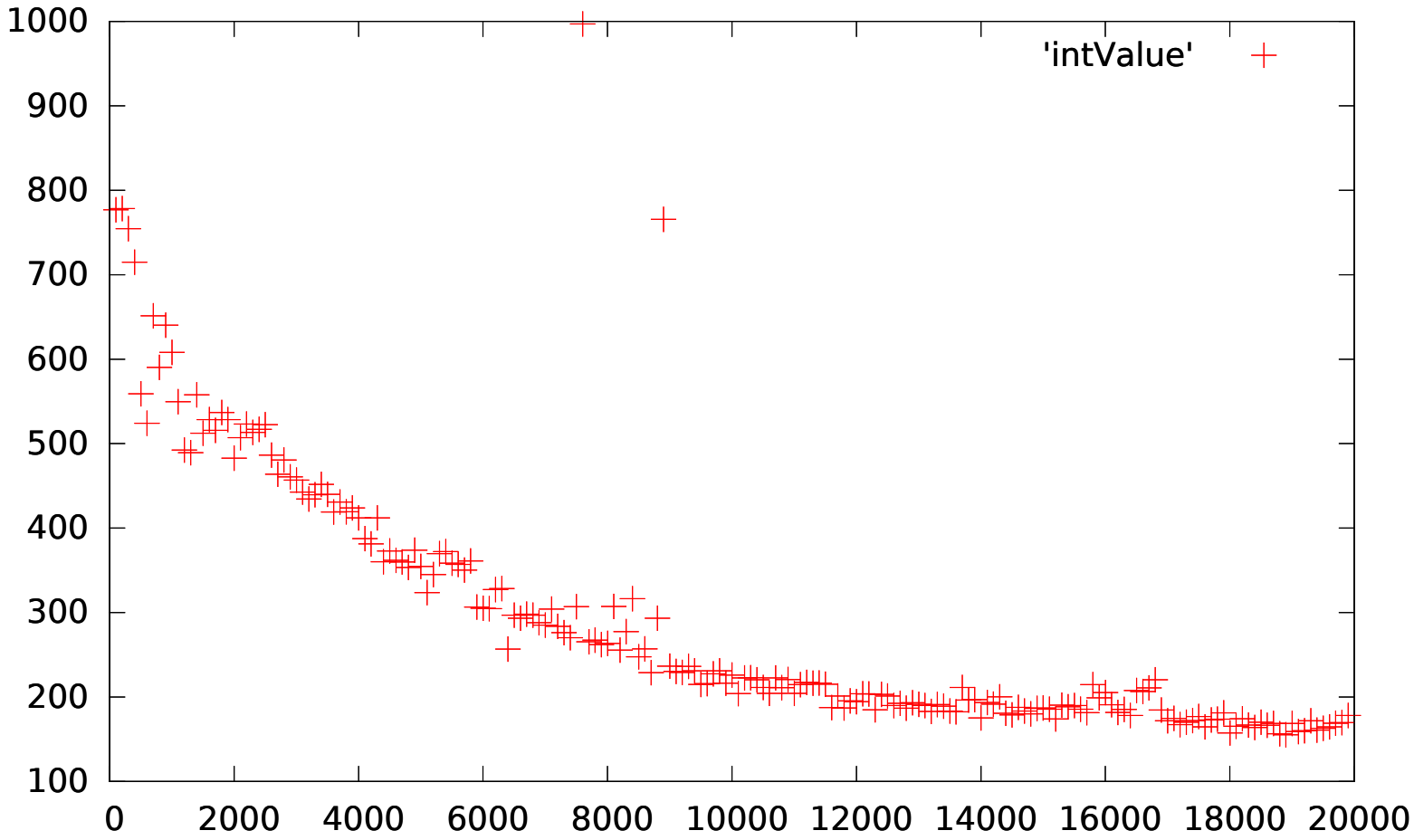
```
actionPerformed(_, _State, Event) ->
  Button = java:call(Event,getSource,[]),
  Text = java:string_to_list(java:call(Button,getText,[])),
  String =
    if
      Text=="Hello" -> "World";
      true -> "Hello"
    end,
  java:call(Button,setText,[String]),
  {reply,void}.
```

- An “Erlang” object has a persistent state which can be updated by “member functions” (second argument above)
- Current limitation: a “member function” should not call methods of its own object directly or indirectly

What is the performance of JavaErlang?

The most important measure is probably the *minimum delay between issuing a call to Java from Erlang, and receiving a reply*

Benchmarking `java:call(Obj,intValue,[])`



- Call time various from 0.16 to 0.8 milliseconds
- Ideas for optimizing the use of Jinterface?

Summary

+ JavaErlang provides an easy-to-use API for calling Java
+ Handles almost all of Java (not synchronized)
+ Little maintenance required
+ Automatic cross language GC

– Still some speed concerns
– The implementation of Java classes using Erlang can be improved
– No beautiful and convenient syntax

Future

- Implement JavaErlang on top of NIF and JNI?
  Should be feasible – and would dramatically improve performance
- But we would still have the performance penalty from using Java reflection (and table lookups, and Erlang NIFs)