

Getting Erlang to talk to the outside world

Joe Armstrong
Swedish Institute of Computer Science
joe@sics.se

October 7, 2002

Abstract

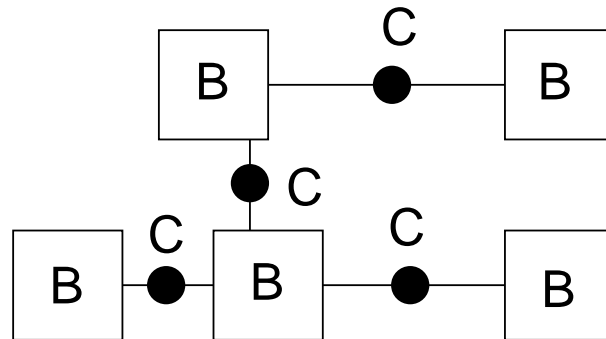
We describe a method for Erlang to communicate with programs written in different languages.

We assume that these programs can operate anywhere in a distributed network.

We describe a transport format, a type system and a contract language for specifying the behaviour of communicating programs in a distributed network.

http://www.sics.se/~joe/talks/pittsburgh_2002_ubf.pdf

Contracts



B = Black box, C=Contract checker

Components in a distributed system communicate by following some protocol.

Question: How can we ensure that all components in a distributed system obey the rules of the protocol?

Answer: Contracts.

Question: How should programs written in different languages talk to each other?

Answer: UBF.

Universal Binary Format

- UBF(A) = An language independent encoding scheme.

Approximately equivalent to well-formed XML - but much more efficient.

- UBF(B) = A type scheme and a contract language.

Approximately equivalent to XML schemas + WSDL but more expressive.

UBF(A)

We assume a stack machine.

Primitive type encodings (human readable).

- **integers** - 3987597834, -348576 ...
- **strings** - "this is a string"
- **memory buffers** - <Int>~~
- **constants** - 'monday', 'november' etc.

Constructed type encodings:

- **tuples** - {Arg1,Arg2,...,Argn}
- **lists** - #Arg1&Args2&...&ArgN

White space (tab, nl, blanks) is ignored.

When an item is recognized it is pushed onto the recognition stack.

$\$$ denotes *end of item* at which time the recognition stack should contain exactly one item.

All other characters are used for caching.

- $>C$ - pop the top item of the recognition stack and store into `reg[C]`
- C - push `reg[C]` onto the recognition stack

Semantic tags

- Obj 'Tag'

Means that Obj has the semantic value Tag.

Example

1234 ~ ~ 'jpeg'

Means that the chunk of 1234 bytes is of type "jpeg" - the application is supposed to know what this means.

Simple types

UBF(B) has a simple type system.

Primitive types:

- `int()` - means a UBF(A) integer
- `string()` - means a UBF(A) string
- `constant()` - means a UBF(A) constant
- `bin()` - means a UBF(A) memory buffer
- `X()` - means an object of type X

Primitive UBF(A) literals are also types.

Constructed Types

$\{T_1, T_2, \dots, T_n\}$ Is the *tuple type* if $T_1 \dots T_n$ are types. We say that $\{x_1, x_2, \dots, x_n\}$ is of type $\{T_1, T_2, \dots, T_n\}$ if x_1 is of type T_1 , x_2 is of type T_2 , ... and x_n is of type T_n .

$[T]$ Is the *list type* if T is a type. We say that $\# x_n \ \& \ x_{n-1} \ \& \ \dots \ x_2 \ \& \ x_1$ is of type $[T]$ if all x_i are of type T .

$T_1|T_2$ Is the *alternation type* if T_1 and T_2 are types. We say that x is of type $T_1 \ | \ T_2$ if x is of type T_1 or if x is of type T_2 .

New types

This defines several new types in UBF(B).

```
+TYPES
  person()      = {person,
                   firstname(),
                   lastname(),
                   sex(),
                   age()};
  firstname()   = string();
  lastname()    = string();
  age()         = int();
  sex()         = male | female;
  people()     = [person()].
```

Here is an example of the `people()` type in a UBF(A) encoding:

```
'person' >p
  # {p "jim" "smith" 'male' 10} &
    {p "susan" "jones" 'female' 14} & $
```

And in XML

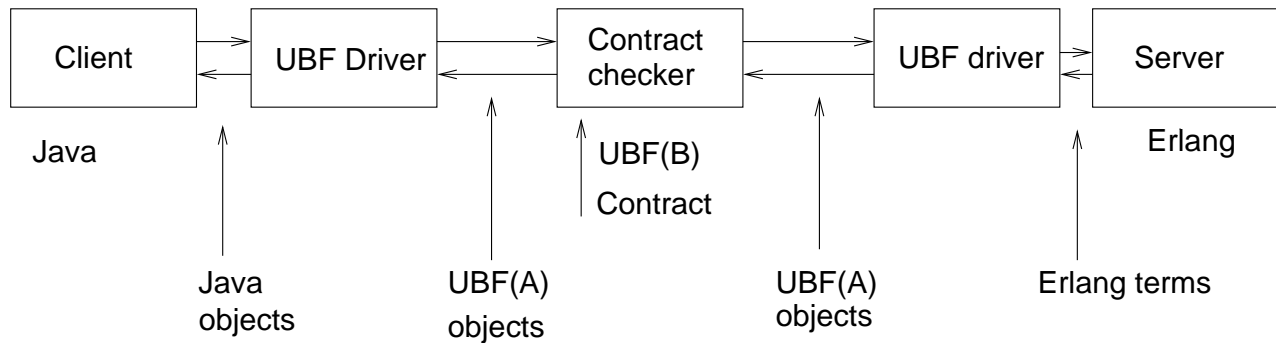
```
<people>
  <person>
    <firstname>jim</firstname>
    <lastname>smith</lastname>
    <sex>male</sex>
    <age>10</age>
  </person>
  <person>
    <firstname>susan</firstname>
    <lastname>jones</lastname>
    <sex>female</sex>
    <age>14</age>
  </person>
</people>
```

Size of XML = 215 characters

Size of UBF(A) = 65 characters

imagine what the schema looks like!

The Contract Language



The contract is a set of four tuples of the form:

$$\{S_{in}, T_{in}, T_{out}, S_{out}\}$$

This means that if the server is in state S_{in} and it receives a message of type T_{in} then it may possibly respond with a message of type T_{out} and change its state to S_{out} .

File Server contract

```

info()           = info;
description()   = description;
services()     = services;
contract()     = contract;

file()          = string();
ls()           = ls;
files()        = {files, [file()]};
getFile()      = {get, file()};
noSuchFile()  = noSuchFile.

+STATE start
    ls()        => files()      & start;
    getFile()   => binary()     & start
                | noSuchFile() & stop.

+ANYSTATE
    info()      => string();
    description() => string();
    contract()  => term().

```

IRC Types

```
info()           = info;
description()    = description;
contract()      = contract;
bool()          = true | false;
nick()          = string();
oldnick()       = string();
newnick()       = string();
group()         = string();
logon()         = logon;
proceed()       = {ok, nick()}
listGroups()    = groups;
groups()        = [group()];
joinGroup()     = {join, group()}
leaveGroup()    = {leave, group()};
ok()           = ok;
changeNick()    = {nick, nick()}
msg()          = {msg, group(), string()}
msgEvent()     = {msg, nick(), group(), string()};
joinEvent()    = {joins, nick(), group()};
leaveEvent()   = {leaves, nick(), group()};
changeNameEvent() = {changesName,
                    oldnick(), newnick(), group()}.
```

IRC Rules

```
+STATE start logon() => proceed() & active.
```

```
+STATE active
```

```
  listGroups() => groups() & active;
```

```
  joinGroup()  => ok() & active;
```

```
  leaveGroup() => ok() & active;
```

```
  changeNick() => bool() & active;
```

```
  msg()        => bool() & active;
```

```
  EVENT => msgEvent();           % Message from group
```

```
  EVENT => joinEvent();         % Nick joins group
```

```
  EVENT => leaveEvent();       % Nick leaves group
```

```
  EVENT => changeNameEvent(). % Nick changes name
```

```
+ANYSSTATE
```

```
  info()          => string();
```

```
  description()  => string();
```

```
  contract()     => term().
```

Experience

- Easy to implement (1100 lines of Erlang)
- Space efficient (UBF(A) files were on average 59% of the size of files in the Erlang external term format)
- Fast parsing (compared to XML)
- Contracts were brilliant for debugging applications written in different languages

Observations

- Much easier than XML, XML-schemas, WSDL
- Fast parsing
- Easy to port