

# Getting Erlang to talk to C and C++

Hal Snyder  
Vail Systems, Inc.  
570 Lake Cook Rd, STE 408  
Deerfield, IL 60015  
hal@vailsys.com

Rick Pettit  
Vail Systems, Inc.  
570 Lake Cook Rd, STE 408  
Deerfield, IL 60015  
rpettit@vailsys.com

## ABSTRACT

Three approaches to the interfacing of C and C++ applications to resources running on OTP (Open Telecom Platform) are discussed: ad hoc TCP, ei, and UBF.

## 1. CURRENT PLATFORM

Vail Systems is a computer telephony applications service provider and hosting company. We do custom IVR (interactive voice response) as well as hosting of customer-based VoiceXML applications. The majority of the platform uses voice over IP, specifically SIP and RTP. We operate two sites, handling several million calls per day. OTP is used primarily for distributed services:

- least cost routing of outbound calls
- call detail record data collection
- resource management
- gathering platform statistics

SIP applications are typically written in threaded C++, using one thread per call, plus additional housekeeping threads.

## 2. STATEMENT OF THE PROBLEM

We have in-house expertise in C, C++, and OTP, with a significant amount of software already developed in all three.

Unfortunately, interaction between C/C++ and OTP systems has been limited and costly. In addition, new software development presents an unpleasant either/or choice, instead of allowing us to take advantage of a combined approach.

We want to improve the situation. For example, in many instances it would be convenient for programmers in the C and C++ camp to use Mnesia tables with minimal interfacing effort and Erlang code, with the usual benefits of fault tolerance, distribution, SNMP-watchability, etc.

This presentation discusses three generations of work on the problem. Other interesting discussions of the matter have appeared on line, for example [1].

## 3. AD HOC PROTOCOL

For the first approach, an ad hoc query-response protocol was developed, allowing a C++ client to invoke selected OTP M,F,A (module, function, argument list) triples. The C++ client connects to an OTP node running on the same host, the request broker. Using the localhost OTP node

reduces resource discovery issues. The request broker is part of an OTP cluster including servers capable of servicing the C++ client's requests.

A sender issuing a `sync` request will await a reply, while `async` requests are fire-and-forget. In a second protocol revision, `handoff` was added in which the sender awaits acknowledgment from the request broker node but not the target OTP server hosting the desired module.

```
request: <Id><sync|async|handoff> <M,F,A>  
reply: <Id><reply data>
```

Use of this interface adds a thread to the application for sustaining a heartbeat with the request broker.

The code for this interface is part of a large C++ utility system which makes extensive use of the STL (standard template library), object oriented smart wrappers for data and function pointers, etc.

The above architecture was the result of extensive and heated discussion regarding the most efficient and reliable ways of doing things.

At the time, there was much higher confidence in C++, threads, STL, and our own ad hoc TCP protocols than there was in OTP, to us a less known and less tested environment.

The first application to use this interface was LCR, the least cost routing outbound calling subsystem. It took about 5 months to write and put through extensive testing. The C++ code comprising the OTP interface consists of 1274 lines, with 1521 more lines of test code.

This facility has been in use for over a year. It has handled more than 48 million requests, with no downtime and zero errors detected.

## 4. EI AND C NODES

The second approach was used when, almost a year after beginning the LCR project, we needed to interface a new C++ client to an existing service which ran on OTP and already had OTP (actually, jinterface) clients.

Existing clients use intentional names in DNS to locate the servers; a name in DNS indicates the service provided and resolves to a list of A records for all hosts offering that service.

A C++ client interface could be constructed from three layers:

1. Ericsson's `ei` library
2. a C language shim doing keep-alive and query-response matching

### 3. C++ application-specific objects

The C++ client is a typical multithreaded telephony application. We expect many threads to access the OTP resource concurrently during normal operation.

There were several reasons for using a different approach the second time.

1. The protocol (OTP transport) was already done. [2]
2. It would not be necessary to disturb OTP servers in production.
3. Our confidence in OTP had increased.
4. The ad hoc approach previously used suffered from build system issues.
5. No middle-tier request broker OTP node would be involved.

The first application to use the `ei`-based interface is a C++ call control engine. It is presently undergoing integration testing.

Requirements for servicing `ERL_TICK` messages from the remote OTP node and the multithreaded nature of the application mandated a threaded approach to this interface. Despite our experience with the first interface, we found threaded programming complicated things [4]. While prototype code was doing successful transactions by the second day of programming, we were still finding problems relating to threads four weeks later.

Another problem that arose along the way relates to the absence of intrinsic transaction identifiers in the `ei_rpc` family of functions. Because we did not want to allow client threads to block indefinitely, it was necessary to use `ei_rpc_to` followed by `ei_rpc_from` with a finite timeout.

Using the following server function

```
add1(Num) ->
    timer:sleep(2000),
    Num + 1.
```

with a client timeout of 1 second in `ei_rpc_from`, we saw the following:

```
>./ei_rpc_cli
connected to node1@fafner.vail fd=3
0 + 1 = ? ERL_TIMEOUT
1 + 1 = ? ERL_TIMEOUT
2 + 1 = 1
3 + 1 = 2
4 + 1 = ? ERL_TIMEOUT
5 + 1 = 4
6 + 1 = 3
7 + 1 = 5
8 + 1 = ? ERL_TIMEOUT
```

Work-around was to patch `ei_connect.c`, deleting occurrences of

```
self->num = fd;
```

from `ei_reg_send()` and `ei_rpc_to()`.

The C shim consists of 2698 lines of C source plus 1571 lines of test, and is still undergoing debugging. The C++ interface is 433 lines of source plus 77 line of test.

## 5. UBF

Although the `ei` interface is not yet in full production, there is still interest in a third approach, using UBF. This is ongoing investigation as an after-hours research project.

Proposed architecture is as in Joe Armstrong's groundbreaking and provocative UBF paper, [3]. Each UBF session will be handled by a single TCP connection and a single client thread.

Our C and C++ applications do not use UBF- or Erlang-style terms internally. For this reason, we make the generic part of a C UBF driver at the lexical level, allowing sending and receiving of UBF tokens. Applications create specific UBF(B) types, for which there will be C and C++ I/O operations.

On the UBF side, we want to separate UBF driver from contract checker so that they may be used independently.

Reasons for interest in the UBF approach are:

- better security than `ei_rpc`
- going beyond N,M,F,A (node, module, function, argument list) call model
- more varied endpoints: such as Java client, C++ server
- UBF + Erlang possibly more effective than XML + ECMAScript
- contract checker is a concise, executable document of system interactions

Our initial application for UBF is a call bridging engine. Resource management and work requests from outside will be handled by OTP. Telephony such as call setup and mixing of audio streams will be done by a C++ executable.

While the project is still in initial stages, we already have a few observations about UBF(A):

1. Reverse ordering of list entries forces a store-and-forward approach, which might be undesirable if streaming is desired or copying is to be minimized. A suggestion is to use [ and ] as delimiters for non-reversed lists.
2. It might be better for semantic tags to precede the values they modify. If that is done, the recipient could use the semantic tag to decide what action to take on a following binary without having to wait until the entire value is received.
3. Because decimal notation seems so unnatural for some things, and because we like looking at raw packet traces, it would be nice to support alternate radices with integers, at least a prefix of 0x for hexadecimal.

## 6. CONCLUSIONS

- We need a powerful yet convenient interface mechanism to get the most out of a hybrid platform of OTP and non-OTP systems. It must be readily acceptable by non-OTP enthusiasts.
- Ad hoc TCP protocol and localhost request broker yielded a very reliable system, but the system was not easy enough to replicate.
- Using an `ei` interface involved some one-time coding costs, but provides a convenient way to access existing OTP code without modification

- UBF holds promise of broader range of application than the other two approaches.

NOTE: For supporting source code and follow-up results, please consult the website

<http://www.drxyzzy.org/>

## 7. ACKNOWLEDGMENTS

The authors would like to thank Leon Smith for several interesting discussions of UBF via email during the writing of this paper, and specifically for the suggestion that semantic tags precede the values they modify.

## 8. REFERENCES

- [1] Distribution by another means. In *erlang-questions mailing list*.  
<http://www.erlang.org/ml-archive/erlang-questions/200006/msg00020.html>.
- [2] ERLANGS EXTERNAL FORMAT and distribution protocol. In *otp\_src\_R9C-0 source distribution*, 2003.  
[erts/emulator/internal\\_doc/erl\\_ext\\_dist.txt](http://erts/emulator/internal_doc/erl_ext_dist.txt).
- [3] J. Armstrong. Getting Erlang to talk to the outside world. In *ACM SIGPLAN Erlang Workshop '02 Pittsburg, PA USA*, October 2002.  
<http://www.sics.se/~joe/ubf/site/ubf.pdf>.
- [4] J. Ousterhout. Why Threads Are A Bad Idea (for most purposes). Invited Talk, 1996 USENIX Technical Conference, January 1996.  
[http://www.cc.gatech.edu/ccg/people/rob/software/threads/ousterhout\\_threads.html](http://www.cc.gatech.edu/ccg/people/rob/software/threads/ousterhout_threads.html).