The Remote Procedure Call package

Henry Thompson

File: {ivy}<hthompson>lisp>rpc>rpc.bravo, .press
Revised: January 23, 1983 12:19 AM

Remote Procedure Calls

The file `<lispusers>rpc.dcom` implements the (documentation forthcoming - ref. Andrew Birrell) Cedar RPC transport mechanism. Its architecture is copied more or less directly from the Cedar implementation, to which reference should be made in difficulty. It is designed to be used in conjunction with stubs produced by the Interlisp-D version of Lupine (see below) to provide remote service.

To start the RPC world, call `InitRPC()` - to turn it off, use `StopRPC().`

LUPINE - Producing stubs

The goal of all this is to allow an Interlisp-D system to export services to or import services from remote machines. The file `<lispusers>lupine.dcom` implements a stub creation mechanism modeled on the Cedar version (ref. Bruce Nelson's thesis (CSL-81-9) and the Lupine User's Guide - [indigo]<cedar>documentation>LupineUsersGuide.press.). In particular the function `Lupine(packageName functionSpecList signalSpecList lupineTypeString)` will produce a set of server stubs to interface between the RPC transport mechanisms and a set of vanilla Lisp functions, and a set of client stubs to communicate between callers of those functions and the RPC transpost mechanism. This allows the network to intervene between the caller and the callee.

The crucial point to comprehend in the RPC world is that the semantics of a remote call are the same as the semantics of a local call - all the effort is directed at hiding the fact that anything out of the ordinary is going on - pay no attention to the man behind the curtain.

A trivial example will demonstrate the basic story - suppose we have the following Cedar interface spec:

```
Trivial: DEFINITIONS =

BEGIN
Trivial: PROC[arg1: INT, arg2: Rope.ROPE] RETURNS[result: BOOLEAN];

TrivialFailure: TYPE = { badNum, badString };

TrivialFailed: SIGNAL[why: TrivialFailure];

END.
```

Translating this with Cedar Lupine will produce client and server interfaces to the RPC transport mechanisms. To plug into those with from Lisp, or to plug Lisp to Lisp, we must supply the same information as is contained in the Cedar interface spec. to Lisp. The following records from Lisp Lupine are used to specify an interface:

```
(RECORD FunctionSpec (fn args result resultArg))
(RECORD ArgSpec (argName argType argParm))
(RECORD ResultSpec (resRecord . resBits))
(RECORD ResBit (resField resType resParm))
```

The following are the equivalent for Lisp Lupine of the above interface spec:

```
(RPAQQ TrivialSpec (
   (Trivial ((arg1 FIXP)(arg2 STRING)) BOOLEAN)  ))
(RPAQQ TrivialSignals (
   (TrivialFailed ENUMERATION (badNum badString) NIL)  ))
```

The same format is used for both functions and signals, since signals are just functions by another name. Signals are described in <lispusers>signal.press.

The types available are {SSMALLP FIXP BOOLEAN STRING ATOM STREAM ENUMERATION}, where SSMALLP is -2^{15} - 2^{15}-1, and STREAM is like string but goes directly onto a stream. The equivalent Cedar types are {INTEGER INT BOOLEAN Rope.ROPE ATOM Rope.ROPE <enumerated type>}. On the Lisp side, the latter two types take and additional specification - the name of the stream to use in the STREAM case, and the set of values in the ENUMERATION case.

The syntax of the spec is confused by the necessity for distinguishing single from multiple return values, and atomic from compound arguments to signals. Thus the result field may either have a complex specification, giving the record name and field specifications for unpicking a multiple return, or it may simply give the type of a single return. In the latter case, the resultArg field is used for the parameter of the type, if any. This option of single vs. multiple is present for both the args and result fields of a signal spec, as the function Signal itself has only one parameter for the argument(s) to the signal being raised, so if this parameter is compound it too must have a record for unpicking and/or constructing it. This leads to the following additional complexity, that if the single argument to a signal requires a parameter, then the rest of the spec. is displaced to make room for it. This is shown in the example above, where the single argument is an enumerated type. The NIL is the result spec. - any one of function arg, signal arg, function result, or signal result may be NIL.

If we call (Lupine 'Trivial TrivialSpec TrivialSignals "magic-string") we will get back the names of a pair of filecoms for all the necessary stubs etc. The *magic-string* will be used as a default in the process of matching server and client at the time of initial connection. If you want to allow it to default to the same value as the default generated by Cedar Lupine, you can read it out of the file TrivialRpcClientImpl.mesa, at the point in the definition of ImportInterface where it says:

```
      type:  IF ~IsNull[type]
         THEN type ELSE "magic-string"L,
```

To actually export the interface so that clients can call you, you must create an instance of the record InterfaceName, with the field instance set to the name of the service you wish to provide, which should be a Grapevine rName. Then you can call ExportTrivial(<the InterfaceName> <user> <password>), where the user and password are owners of the rName in the interface. Likewise to import an interface exported by someone else, use ImportTrivial(<the InterfaceName>). See the forthcoming Cedar RPC documentation for a discussion of all the options with respect to establishing connections.

A complete example is the easiest way to understand how all this works. {ivy}<hthompson>lisp>rpc>simple and ...>simpleuser are the basis for such an example. The functions in simpleuser call the functions in simple. simple contains the necessary arguments for Lupine. ...>simpleclient and ...>simpleserver contain the stubs generated by Lupine. simpleuser and simpleclient would be loaded into one machine, and simple and simpleserver into another. {ivy}<hthompson>cedar>rpc>simple.df is a directory of all the Cedar files needed to make up the equivalent server and/or client in Cedar.

Messages to HThompson.pa