9.2.2 Worker Routines

Now, let's consider the routines which implement the interactive initiator. There are only two: `ryinitiator` and `getline`.

The `ryinitiator` routine does most of the work:

```
int     ryinitiator (argc, argv, myservice, mycontext,
                mypci, ops, dispatches, quit)
int     argc;
char  **argv,
       *myservice,
       *mycontext,
       *mypci;
struct RyOperation ops[];
struct dispatch *dispatches;
IFP    quit;
```

The parameters to this procedure are:

`argc/argv:` the argument vector (and its length) that the program was invoked with;

`myservice:` the non-host portion of the application-entity information;

`mycontext:` the application context name of the service;

`mypci:` the abstract syntax of the service;

`ops:` the operations defined for the service;

`dispatches:` a pointer to a `dispatch` table; and,

`quit:` a routine to call when the association is to be terminated.

The function of this routine is straight-forward though tedious. First, `myname` is initialized to the name that the program was invoked with. Next, a rudimentary argument check is done, and the application-entity information and presentation address for the desired service are computed. This is followed by computing the application context and default presentation context information for the service, along with a session reference identifier. Finally, some diagnostic information may be printed out if the program will operate in interactive mode.

The routine `AcAssocRequest` is called to establish an association for the service. If successful, the routine `RoSetService` is used to tell the remote operations library to use the presentation service as the underlying service.

The interactive loop is then entered: a line is read from the standard input (the `getline` routine is called), it is broken into components, a search is performed on the dispatch table to find a matching keyword, and then finally the user-supplied routine is called. When end-of-file is reached (or if the user-supplied routine returns the manifest constant `DONE`), the assocation is released.

If additional arguments were present on the invocation line, then the named operation is invoked instead of entering an interactive loop.

```
/* ryinitiator.c - generic interactive initiator */

#include <stdio.h>
#include <varargs.h>
#include "ryinitiator.h"

/*  DATA */

static char *myname = "ryinitiator";

extern char *isodeversion;

/*  INITIATOR */

ryinitiator (argc, argv, myservice, mycontext, mypci, ops, dispatches, quit)
int     argc;
char  **argv,
      *myservice,
      *mycontext,
      *mypci;
struct RyOperation ops[];
struct dispatch *dispatches;
IFP     quit;
{
   int      iloop,
```

```
        sd;
char    buffer[BUFSIZ],
        *vec[NVEC + 1];
register struct dispatch   *ds;                                    30
struct SSAPref sfs;
register struct SSAPref *sf;
register struct PSAPaddr *pa;
struct AcSAPconnect accs;
register struct AcSAPconnect   *acc = &accs;
struct AcSAPindication  acis;
register struct AcSAPindication *aci = &acis;
register struct AcSAPabort *aca = &aci -> aci_abort;
AEI     aei;
OID     ctx,                                                       40
        pci;
struct PSAPctxlist pcs;
register struct PSAPctxlist *pc = &pcs;
struct RoSAPindication rois;
register struct RoSAPindication *roi = &rois;
register struct RoSAPpreject *rop = &roi -> roi_preject;

if (myname = rindex (argv[0], '/'))
    myname++;
if (myname == NULL || *myname == NULL)                             50
    myname = argv[0];

if (argc < 2)
    adios (NULLCP, "usage:  %s host [operation [ arguments ...  ]]", myname);

if ((aei = _str2aei (argv[1], myservice, mycontext, argc < 3,
                     NULLCP, NULLCP)) == NULLAEI)
    adios (NULLCP, "unable to resolve service:  %s", PY_pepy);
if ((pa = aei2addr (aei)) == NULLPA)
    adios (NULLCP, "address translation failed");                  60

if ((ctx = ode2oid (mycontext)) == NULLOID)
    adios (NULLCP, "%s:  unknown object descriptor", mycontext);
if ((ctx = oid_cpy (ctx)) == NULLOID)
    adios (NULLCP, "out of memory");
```

```
if ((pci = ode2oid (mypci)) == NULLOID)
        adios (NULLCP, "%s:  unknown object descriptor", mypci);
if ((pci = oid_cpy (pci)) == NULLOID)
        adios (NULLCP, "out of memory");
pc -> pc_nctx = 1;                                                    70
pc -> pc_ctx[0].pc_id = 1;
pc -> pc_ctx[0].pc_asn = pci;
pc -> pc_ctx[0].pc_atn = NULLOID;

if ((sf = addr2ref (PLocalHostName ())) == NULL) {
        sf = &sfs;
        (void) bzero ((char *) sf, sizeof *sf);
}

if (argc < 3) {                                                       80
        printf ("%s", myname);
        if (sf -> sr_ulen > 2)
            printf (" running on host %s", sf -> sr_udata + 2);
        if (sf -> sr_clen > 2)
            printf (" at %s", sf -> sr_cdata + 2);
        printf (" [%s, ", oid2ode (ctx));
        printf ("%s]\n", oid2ode (pci));
        printf ("using %s\n", isodeversion);

        printf ("%s...  ", argv[1]);                                  90
        (void) fflush (stdout);

        iloop = 1;
}
else {
        for (ds = dispatches; ds -> ds_name; ds++)
            if (strcmp (ds -> ds_name, argv[2]) == 0)
                break;
        if (ds -> ds_name == NULL)
            adios (NULLCP, "unknown operation \"%s\"", argv[2]);      100

        iloop = 0;
}
```

```
    if (AcAssocRequest (ctx, NULLAEI, aei, NULLPA, pa, pc, NULLOID,
                    0, ROS_MYREQUIRE, SERIAL_NONE, 0, sf, NULLPEP, 0, NULLQOS,
                    acc, aci)
            == NOTOK)
        acs_adios (aca, "A-ASSOCIATE.REQUEST");

    if (acc -> acc_result != ACS_ACCEPT) {
        if (iloop)
            printf ("failed\n");

        adios (NULLCP, "association rejected:  [%s]",
                    AcErrString (acc -> acc_result));
    }
    if (iloop) {
        printf ("connected\n");
        (void) fflush (stdout);
    }

    sd = acc -> acc_sd;
    ACCFREE (acc);

    if (RoSetService (sd, RoPService, roi) == NOTOK)
        ros_adios (rop, "set RO/PS fails");

    if (iloop) {
        for (;;) {
            if (getline (buffer) == NOTOK)
                break;

            if (str2vec (buffer, vec) < 1)
                continue;

            for (ds = dispatches; ds -> ds_name; ds++)
                if (strcmp (ds -> ds_name, vec[0]) == 0)
                    break;
            if (ds -> ds_name == NULL) {
                advise (NULLCP, "unknown operation \"%s\"", vec[0]);
                continue;
            }
```

```
            invoke (sd, ops, ds, vec + 1);
        }
    }
    else
        invoke (sd, ops, ds, argv + 3);

    (*quit) (sd, (struct dispatch *) NULL, (char **) NULL, (caddr_t *) NULL);
}

/*  */

static  invoke (sd, ops, ds, args)
int     sd;
struct RyOperation ops[];
register struct dispatch *ds;
char  **args;
{
    int     result;
    caddr_t in;
    struct RoSAPindication  rois;
    register struct RoSAPindication *roi = &rois;
    register struct RoSAPpreject  *rop = &roi -> roi_preject;

    in = NULL;
    if (ds -> ds_argument && (*ds -> ds_argument) (sd, ds, args, &in) == NOTOK)
        return;

    switch (result = RyStub (sd, ops, ds -> ds_operation, RyGenID (sd), NULLIP,
                        in, ds -> ds_result, ds -> ds_error, ROS_SYNC, roi)) {
        case NOTOK:             /* failure */
            if (ROS_FATAL (rop -> rop_reason))
                ros_adios (rop, "STUB");
            ros_advise (rop, "STUB");
            break;

        case OK:                /* got a result/error response */
            break;
```

```
        case DONE:                  /* got RO-END? */
            adios (NULLCP, "got RO-END.INDICATION");
            /* NOTREACHED */

        default:
            adios (NULLCP, "unknown return from RyStub=%d", result);
            /* NOTREACHED */
    }                                                                   190

    if (ds -> ds_free && in)
        (void) (*ds -> ds_free) (in);
}

/*  */

static int  getline (buffer)
char   *buffer;
{                                                                       200
    register int    i;
    register char  *cp,
                   *ep;
    static int  sticky = 0;

    if (sticky) {
        sticky = 0;
        return NOTOK;
    }
                                                                        210
    printf ("%s> ", myname);
    (void) fflush (stdout);

    for (ep = (cp = buffer) + BUFSIZ - 1; (i = getchar ()) != '\n';) {
        if (i == EOF) {
            printf ("\n");
            clearerr (stdin);
            if (cp != buffer) {
                sticky++;
                break;                                                  220
            }
```

```
            return NOTOK;
        }

        if (cp < ep)
            *cp++ = i;
    }
    *cp = NULL;

    return OK;
}

/* */

void    ros_adios (rop, event)
register struct RoSAPpreject *rop;
char    *event;
{
    ros_advise (rop, event);

    _exit (1);
}


void    ros_advise (rop, event)
register struct RoSAPpreject *rop;
char    *event;
{
    char    buffer[BUFSIZ];

    if (rop -> rop_cc > 0)
        (void) sprintf (buffer, "[%s] %*.*s", RoErrString (rop -> rop_reason),
                rop -> rop_cc, rop -> rop_cc, rop -> rop_data);
    else
        (void) sprintf (buffer, "[%s]", RoErrString (rop -> rop_reason));

    advise (NULLCP, "%s:  %s", event, buffer);
}
```

```
/*  */

void    acs_adios (aca, event)
register struct AcSAPabort *aca;
char   *event;
{
    acs_advise (aca, event);

    _exit (1);
}


void    acs_advise (aca, event)
register struct AcSAPabort *aca;
char   *event;
{
    char    buffer[BUFSIZ];

    if (aca -> aca_cc > 0)
        (void) sprintf (buffer, "[%s] %*.*s",
                AcErrString (aca -> aca_reason),
                aca -> aca_cc, aca -> aca_cc, aca -> aca_data);
    else
        (void) sprintf (buffer, "[%s]", AcErrString (aca -> aca_reason));

        advise (NULLCP, "%s:  %s (source %d)", event, buffer,
                aca -> aca_source);
}

/*  */

#ifndef lint
void    _advise ();


void    adios (va_alist)
va_dcl
{
    va_list ap;
```

```
                                                                    300
    va_start (ap);

    _advise (ap);

    va_end (ap);

    _exit (1);
}
#else
/* VARARGS */                                                       310

void    adios (what, fmt)
char   *what,
       *fmt;
{
    adios (what, fmt);
}
#endif


                                                                    320
#ifndef lint
void    advise (va_alist)
va_dcl
{
    va_list ap;

    va_start (ap);

    _advise (ap);
                                                                    330
    va_end (ap);
}


static void _advise (ap)
va_list    ap;
{
    char    buffer[BUFSIZ];
```