Introduction

somFree supports DSOM technology to produce distributed system. At the present time, somFree confirms CORBA 1.1 C language mapping as published in CORBA 1.1 Specification and IIOP 1.1 as published in CORBA Specification ??? (link here...). This mean you can write client and server application which will interoperable with ORB's which confirm at least CORBA Specification. This document describes practical approaches of interoperability with some known ORBs.

As example base, ORBit2 ORBit Beginners Documentation V1.2 (orbit-docs.tar.gz (add file here because hard to find now)) was chosen. First, we will create a simple IDL file, compile it and write somFree client which will communicate to corresponding server.

Defining IDL

ORBit2 example contains simplest example of class is Echo, which allow sending messages from client to server. Here is original echo.idl from ORBit Beginners Documentation V1.2:

```
// MyFirstOrbit program - The Echo object
//
// All this does is pass a string from the
// client to the server.

interface Echo
{
        void echoString(in string input);
};
```

To provide SOM specific information, we will modify it as follows:

```
// MyFirstOrbit program - The Echo object
//
// All this does is pass a string from the
// client to the server.

#ifdef __SOMIDL__
#include <somobj.idl>
#endif

interface Echo
#ifdef __SOMIDL__
  : SOMObject
#endif
{
        void echoString(in string input);
#ifdef __SOMIDL__
    implementation
```

```
    {
        releaseorder: echoString;
        majorversion = 1;
        minorversion = 0;
    };
#endif
};
```

Here we added include for root SOM class SOMObject, added Echo parent (optional) and implementation section to provide binary compatibility of future versions of Echo classes.

somFree client implementation

Now we need to compile interface and implementation bindings as well as add Echo class information to Interface Repository (required for SOM 2.1 and before):

```
sc.exe -sh;ih;ir echo.idl
```

Compiler will produce echo.h, echo.ih files.

Now, to make DSOM happy and success creation of Echo proxy object, we need to make empty class implementation. In IBM SOM it is enough to create an implementation template and compile it. But somFree compiler doesn't provide implementation template, so we need to write one:

```
#define Echo_Class_Source

#include <echo.ih>

SOM_Scope void SOMLINK echoString(Echo SOMSTAR somSelf, Environment *ev, /*
in */ string input)
{
}
```

As you see, nothing here but empty method implementation. It is a good idea to trow exception about not implemented method. This allows to prevent class usage as real implementation. On object creation, DSOM will construct a proxy object which will forward all methods calls to the remote object.

And now, ported to SOM client from ORBit example:

```
/*
 * Echo client program.. Hacked by Ewan Birney <birney@sanger.ac.uk>
 * from echo test suite, update for ORBit2 by Frank Rehberger
 * <F.Rehberger@xtradyne.de>. Ported to SOM by Yuri Prokushev.
 *
 * Client reads object reference (IOR) from local file 'echo.ior' and
 * forwards console input to echo-server. A dot . as single character
 * in input terminates the client.
 */
```

```
#include <stdio.h>
#include <somd.h>

/*
 * This header file was generated from the idl
 */

#include "echo.h"

/**
 * test for exception
 */
static
boolean
raised_exception(Environment *ev)
{
    return ((ev)->_major != NO_EXCEPTION);
}

/**
 * in case of any exception this macro will abort the process
 */
static
void
abort_if_exception(Environment *ev, const char* mesg)
{
    if (raised_exception (ev)) {
        somPrintf ("%s %s", mesg, somExceptionId(ev));
        somExceptionFree(ev);
        abort();
    }
}

/*
 * main
 */
int
main (int argc, char *argv[])
{
    FILE * ifp;
    char * ior;
    char filebuffer[1024];

    Environment ev;
    Echo SOMSTAR echo_client;         /* the service */

    /*
     * Standard initalisation of the orb. Notice that
     * ORB_init 'eats' stuff off the command line
     */
```

```
    SOM_InitEnvironment (&ev) ;
    SOMD_Init (&ev) ;
    abort_if_exception(&ev, "init ORB failed");

    EchoNew();

    /*
     * Get the IOR (object reference). It should be written out
     * by the echo-server into the file echo.ior. So - if you
     * are running the server in the same place as the client,
     * this should be fine!
     */

    ifp = fopen("echo.ior","r");
    if( ifp == NULL ) {
        somPrintf("can not open \"echo.ior\"");
        abort ();
    }

    fgets(filebuffer,1023,ifp);
    ior = strdup(filebuffer);

    fclose(ifp);

    /*
     * Actually get the object. So easy!
     */

        echo_client = ORB_string_to_object(SOMD_ORBObject, &ev, ior);
        somPrintf("testec=%d", (long)echo_client);
    abort_if_exception(&ev, "bind failed");

    /*
     * Ok. Now we use the echo object...
     */

    somPrintf("Type messages to the server\n"
        "a single dot in line will terminate input\n");

    while( fgets(filebuffer,1024,stdin) ) {
        if( filebuffer[0] == '.' && filebuffer[1] == '\n' )
            break;

        /* chop the newline off */
        filebuffer[strlen(filebuffer)-1] = '\0';

        /* using the echoString method in the Echo object
         * this is defined in the echo.h header, compiled from
         * echo.idl */
```

```
        Echo_echoString(echo_client,&ev,filebuffer);
        abort_if_exception(&ev, "service not reachable");
    }

    /* Clean up */
    SOMDObject_release(echo_client, &ev);
    abort_if_exception(&ev, "releasing service failed");

    SOMD_Uninit (&ev) ;
    SOM_UninitEnvironment ( &ev)

    /* successfull termination */
    exit (0);
}
```

As you see, here just downgraded code to CORBA 1.1 C Language mapping from CORBA 2.x C Language mapping. ORB initialization and finalization replaced by DSOM initialization and finalization. And, this is matter, EchoNew() call added to create Echo class to make it known to DSOM. Now you can compile it and all mostly ready.

Now you have SOM client with stub class implementation and interface registered in interface repository. You are ready now.

ORBit2

somFree was tested with the latest available ORBit2 implementation (version 2.14.19). ORBit2 shipped as packages on various operational systems, including various Linux distributions, and Cygwin package on Windows system. So, install ORBit2 first using your package manager (add step-by-step guide for Cygwin at least).

Now configure ORBit2 to communicate using IPv4 protocol and disable (optional) ORBit2-specific options. Edit or create /etc/orbitrc (can't find normal desription of file) with following content:

```
ORBIIOPIPv4=1
ORBIIOPIPSock=16000 (is it ok? Found something about standard port)
ORBIIOPIPName=127.0.0.1
ORBIIOPIPv6=0
ORBLocalOnly=1 (restriction for external connections???)
ORBIIOPUNIX=0
```

Now ORBit2 ready to talk via IPv4 with clients. Let's build echo server now. Most of the information are available in ORBit Beginners Documentation V1.2. Here minimal information about build:

Call ORBit2 IDL compiler:

```
orbit-idl-2 --skeleton-impl echo.idl
```

It will produce:

- echo.h
- echo-common.c
- echo-stubs.c
- echo-skels.c
- echo-skelimpl.c

Some of them not required because it is client implementation. Now compile echo-server.c with echo-common.c and echo-skels.c. Provided Makefile not well works with cygwin, but can be changed easily. Here is our version of Makefile for cygwin (make some cleanup of makefile):

```
PREFIX=/usr
CC = gcc
TARGETS=echo-server
ORBIT_IDL=orbit-idl-2
CFLAGS= -v -DORBIT2=1 -D_REENTRANT -I$(PREFIX)/include/orbit-2.0 \
       -I$(PREFIX)/include/glib-2.0 \
       -I$(PREFIX)/lib/glib-2.0/include

LDFLAGS=--verbose -t -Wl,-Bdynamic /bin/cygORBit-2-0.dll \
          /bin/cygglib-2.0-0.dll /usr/lib/libdl.a \
          /usr/lib/libglib-2.0.dll.a /usr/lib/libORBit-2.dll.a \
          --enable-auto-import --enable-runtime-pseudo-reloc

IDLOUT=echo-common.c echo-stubs.c echo-skels.c echo.h

all: $(IDLOUT) echo-server

echo-server : echo-server.o echo-common.o echo-skels.o

$(IDLOUT): echo.idl
    $(ORBIT_IDL) echo.idl
```

Now your server is ready. Execute echo-server and you will have console with waiting echo. To make client know about Echo server copy generated echo.ior to client directory.

Executing somFree client

Nothing hard here. Configure SOM environment (if not done yet) and execute client. It will write:

```
Type messages to the server
a single dot in line will terminate input
```

Now type something and press . to finish client:

```
Hello from somFree!
.
```

On server console, you must see:

```
Hello from somFree!
```

To show some information, you can add to client (before text input loop) something like this:

```
_somDumpSelf(echo_client,0);
```

So you will see after compiling, linking and execution:

```
An instance of Echo__Proxy at address 01219738
{
  type_id    IDL:Echo:1.0
  protocol   SOMD_TCPIP, host:127.0.0.1, port:16000
  object_key 0000000048a4ca0b334633e33a2115ee1080a940010000008a6ea430
}
```

All source code can be found here (cleanup and publish code)

somFree server implementation

(Implement server first, then write chapter)

(omniORB and TAX also must be tested as most available OSS. May be some other ORBs?)

C Language Mapping differences

somFree uses CORBA 1.1 C Language Mappings. Most of CORBA compatible ORBs uses later specifications, at least at CORBA 2.0 level. Some changes was introduced from CORBA 1.1 to CORBA 2.0:

1. All CORBA base types now prefixed by CORBA_ prefix
2. All items from module CORBA prefixed by CORBA_ prefix (as well as other items from other modules also must be prefixed)
3. Environment and Context moved to last function arguments

So, to simplify migration from CORBA 2.0+ mapping to CORBA 1.1 mapping follow include file (not finished yes, contains only some conversions) can be used:

```
/*
 *  CORBA 1.2+ compatibility layer
 */

#ifndef CORBA_Environment
#define CORBA_Environment Environment
#endif
```

```
#define CORBA_exception_init(ev) SOM_InitEnvironment(ev)

#ifndef CORBA_ORB
#define CORBA_ORB ORB
#endif

#ifndef CORBA_ORB_init
CORBA_ORB SOMSTAR CORBA_ORB_init ( int * argc, char ** argv, char * orb_id,
CORBA_Environment * env);

CORBA_ORB SOMSTAR CORBA_ORB_init ( int * argc, char ** argv, char * orb_id,
CORBA_Environment * env)
{
  SOMD_Init(env);
  return SOMD_ORBObject;
}
#endif

#ifndef CORBA_ORB_destroy
#define CORBA_ORB_destroy(orb, ev) SOMD_Uninit(ev);
SOM_UninitEnvironment(ev);
#endif

#ifndef CORBA_OBJECT_NIL
#define CORBA_OBJECT_NIL NULL
#endif

#ifndef CORBA_NO_EXCEPTION
#define CORBA_NO_EXCEPTION NO_EXCEPTION
#endif

#ifndef CORBA_exception_id
#define CORBA_exception_id(ev) somExceptionId(ev)
#endif

#ifndef CORBA_exception_free
#define CORBA_exception_free(ev) somExceptionFree(ev)
#endif

#ifndef CORBA_ORB_string_to_object
#define CORBA_ORB_string_to_object ORB_string_to_object
#endif

#ifndef CORBA_Object_release
#define CORBA_Object_release SOMDObject_release
#endif

#ifndef CORBA_double
#define CORBA_double double
#endif
```

```
#ifndef CORBA_Object
#define CORBA_Object SOMObject
#endif
```

If you convert from CORBA 1.2 mapping such approach mostly enough. But for CORBA 2.0+ mapping you also need to change methods arguments order. For example:

```
echo_client = CORBA_ORB_string_to_object(orb, ior, ev);
```

must be changed to

```
echo_client = CORBA_ORB_string_to_object(orb, ev, ior);
```