Appendix to Simulation of Mobile Underwater Communications

Michel Barbeau*, Stéphane Blouin†, Gimer Cervera‡, Joaquin Garcia-Alfaro§ and Evangelos Kranakis*

* School of Computer Science, Carleton University, K1S 5B6, Ottawa, Ontario, Canada
Email: {barbeau,kranakis}@scs.carleton.ca

† Defence R&D Canada - Atlantic, Dartmouth, NS, Canada B2Y 3Z7
Email: stephane.blouin@drdc-rddc.gc.ca

‡ Universidad Tecnológica Metropolitana, 97279, Merida, Yuc., Mexico
Email: gimer.cervera@utmetropolitana.edu.mx

§ Telecom SudParis, CNRS Samovar UMR 5157, Evry, France
Email: joaquin.garcia-alfaro@acm.org

I. OMNeT++/MATLAB INTEGRATION

We show an example of two external applications interfaced from an OMNeT++ simulation. More specifically, we show how an OMNeT++ simulation receives messages from an external C++ application. In turn, the messages are processed by a Matlab function, that returns the computed results to the OMNeT++ simulation.

The purpose of the example is to complement the details provided in Section **??**, about the integration of OMNeT++ simulations with Matlab functionality — via shared libraries — to implement the physical layer reported in Section **??**. The example should not be confused with the final integration of our proposal. The code listings we show are simplified versions, based on the real code.

The general idea is the following. An OMNeT++ module, called *UWModem*, is programmed to receive messages from an external C++ application called *uw_traffic.cc*. The OMNeT++ module calls, in turn, a Matlab function called *PSK.m*. This Matlab function modulates the input message using PSK. Then, the OMNeT++ module acknowledges the reception to the external application.

The communication between the OMNeT++ module and the external application is conducted via an additional component class (not reported in this Section) denoted as *SocketGate*. The use of sockets-based gates (e.g., via socket-like interfaces, such as Berkeley sockets [**?**]) allows to create OMNET++ instances that act as proxies to the external world within OMNET++. It can be complemented to integrate wired and wireless networking protocols, via other libraries, such as MiXiM and INET [**?**], [**?**].

The *SocketGate* is assumed to maintain an active socket, listening and waiting for external connections, and delivering those incoming messages to other simulation components, and the outcoming messages to the external world. More information about the use of this type of components is reported in [**?**].

Let us now describe how to prepare the Matlab function in order to generate libraries shared with OMNeT++. We assume the following function encoded in the Matlab language, and stored under file *PSK.m*. It modulates the input message *x* received as an input parameter:

```
function y = PSK(x)
  y = pskmod(x);
```

We can now transform the previous Matlab function as a shared ANSI C library, via the Matlab Compiler Runtime Toolbox, using the following commands:

```
mcc -B csharedlib:libuwpsk -v PSK.m
```

Under a Linux-based operating system, the previous command creates the shared library file *libuwpsk.so*, as well as the specification and implementation files *libuwpsk.h* and *libuwpsk.c*. These three files contain all the necessary functions, headers and linkable objects to be used by any other standalone ANSI C or C++ application. For instance, we can invoke them from an OMNeT++ module as follows:

```
#include "libuwpsk.h"

int main(int argc, char *argv[]) {
    //Matlab initialization function
    mclInitializeApplication();

    //Shared library initialization function
    libuwpskInitialize();

    //OMNeT++ interface initialization,
    //which starts the OMNeT++ execution.
    setupUserInterface(argc, argv);

    //Shared library termination function
    libuwpskTerminate();

    //Matlab initialization function
    mclTerminateApplication();
}
```

The previous C++ code contains all the necessary system

calls to initialize and terminate the integration of Matlab modules within the execution space of an OMNeT++ library. This is done via the *setupUserInterface* function. In turn, the following OMNeT++ code, denoted as `Simulation-Network.ned`, is required to create the network structure in which the *UWModel* module will be placed during the simulation:

```
import SocketGate;

//Definition of an OMNeT++ node
simple UWModem
{
   gates:
     input in;
     output out;
}

//Definition of an OMNeT++ network
network UWNetwork
{
  submodules:
    proxy: SocketGate;
    uwmod: UWModem;

  connections:
    uwmod.out-->{delay=100ms;}-->proxy.in;
    uwmod.in<--{delay=100ms;}<--proxy.out;
}
```

The above OMNeT++ code defines a new network, called *UWNetwork*, and places two submodules that are connected via a bidirectional link. The only property associated to the communication link is the transmission delay, defined with the parameter *delay=100ms*. The first submodule, called *proxy*, is an OMNeT++ module (based on the INET and MiXiM libraries) that allows the simulation from receiving information from external applications. It provides a socket-like interface (e.g., via Berkeley sockets) to allow inter-process communication. As soon as it receives a new message, e.g., via a local port number, it forwards the message to the second submodule, denoted as *uwmod*. This second submodule *uwmod* is an instantiation of our proposed *UWModem* module class. The associated code to this submodule is defined in a C++ file, that we call *uwmodem.cc*:

```
class UWModem:public cSimpleModule
{
  protected:
   //Module functions
   virtual void initialize();
   virtual void handleMessage(cMessage *msg);
};

//OMNeT++ primitive to register the new module
Define_Module(UWModem);

//Method to handle incoming messages
void UWModem::handleMessage(cMessage *inMessage)
{
    //Matlab variables
    mxArray *x_ptr;
    mxArray *y_ptr = NULL;

    //Matlab function to convert from byte
    //representation to double representation
    x_ptr = mxCreateDoubleScalar(*inMessage);

    //call our Matlab function to modulate
    //the message using PSK modulation
    mlfPSK(1, &y_ptr, x_ptr);

    //print the result
    printf("\%08x\n",mxGetPr(y_ptr)[0]);

    cMessage *outMessage = new cMessage('ACK');
    send(outMessage, "out");

}
```

The previous OMNeT++ code is an example that allows module class *UWModem* to modulate incoming messages via the shared Matlab function *mlfPSK*. The implementation of function *mlfPSK* is contained within the shared library *libuw-psk.so* that we created at the beginning of our example (cf. the *mcc* command-line example). Once modulated the message, the module acknowledges the reception.

We conclude our example by showing the sample code of the external application, in charge of sending messages to our *UWModem* module. It is assumed to use a socket-like interface (e.g., via Berkeley sockets) to provide inter-process communication with the OMNeT++ simulation. The following code, encoded in file *sender.c*, represents just a simplified version of such a third party application. Several parts of the code are omitted (e.g., definition of indexes, constants, socket transmission and reception functions, etc.).

```
int main(void)
{
    connection = tcp_socket(IP_address, port);

    while (index<COUNTER) {

            // Generate a new random message
            for (i=0; i<nbits; i++) {
                message[i] = rand();
            }

            // Send the message
            socket_write(connection,message);

            // Receive acknowledgment
            socket_read(connection,message);

    index++;
     }

    // Close connection
    socket_close(connection);
    socket_free(connection);
}
```

To sum up, the example proposed in this appendix reports

a very simple OMNeT++ simulation that expects incoming messages from a third party application (either from a virtual software application, or from a real device). The communication starts from the third party application (cf. file *sender.c*) and sends messages to the OMNeT++ simulation via a socket-like communication interface. This interface is connected to a new module we called *UWModem*, that uses our own Matlab function *PSK.m* to conduct PSK modulation of the received messages. Upon reception, the module simply acknowledges the messages.