DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

1222·2022 800 ANNI

UNIVERSITÀ DEGLI STUDI DI PADOVA

W&M

SIGNET

signet.dei.unipd.it

Exercise: develop a DESERT physical layer module

UNWiS - Padova (Italy)

30th of January – 3rd of February 2023

Filippo Campagnaro, Roberto Francescon, Angela Soldà, Michele Zorzi

filippo.campagnaro@unipd.it

Exercise: module or addon development

Develop your own module and use it in a simulation

Let's define our new cool module

Let's define a new module, called `uwcoolphy` that extends `uwphysical`

- inherits from the class of `uwphysical`
- choose whether module or addon
  - to be located in the PHY folder (`DESERT/physical/uwcoolphy`) M
  - or in the addons folder (`DESERT_Addons/uwcoolphy`) A

- file: `uwcoolphy/uwcoolphy.h`
- file: `uwcoolphy/uwcoolphy.cpp`
- file: `uwcoolphy/Makefile.am`
- file: `uwcoolphy/initlib.cpp`
- file: `uwcoolphy/uwcoolphy-init.tcl`

Warning!

First let's give you a warning:

It is preferred to not copy snippets from PDF files

Usually, PDF files contain other UTF characters that are copied into your sources, preventing you from generating compilable code.

If the snippet is very long, resort to another similar source file, instead, and copy snippets from there.

Definition of UwCoolPhy *.h*

Let's see the basic structure of the header (.h)

```
#include <uwphysical.h>

class UwCoolPhy : public UnderwaterPhysical
{
public:
  UwCoolPhy();
  virtual ~UwCoolPhy();

  virtual int command(int argc, const char *const *argv);
  virtual void recv(Packet *p);
  virtual void endTx(Packet *p);
  virtual void endRx(Packet *p);
};
```

Don't forget include guards!

Implementation of UwCoolPhy[1]

Let's have a look at our .cpp file

```
#include <uwcoolphy.h>

UwCoolPhy::UwCoolPhy()
 : UnderwaterPhysical()
{}

UwCoolPhy::~UwCoolPhy() {}

int UwCoolPhy::command(int argc, const char *const * argv)
{
  return UnderwaterPhysical::command(argc, argv);
}
void UwCoolPhy::recv(Packet *p)
{...}
void UwCoolPhy::endTx(Packet *p)
{...}
void UwCoolPhy::endRx(Packet *p)
{...}
```

Implementation of UwCoolPhy[2]

Let's continue to have a look at our .cpp file

Let's add the method to instantiate an object in Tcl right at the beginning of the .cpp file

```
static class UwCoolPhyClass : public TclClass
{
public:
    UwCoolPhyClass()
        : TclClass("Module/UW/CoolPHY")
    {
    }
    TclObject *
    create(int, const char *const *)
    {
        return (new UwCoolPhy());
    }
} class_module_uwcoolphy;
```

Fill in the initlib.cpp

Let's fill in out `initlib.cpp` file

```
#include <tclcl.h>

extern EmbeddedTcl UwCoolPhyTclCode;

extern "C" int
Uwcoolphy_Init()
{
  UwCoolPhyTclCode.load();
  return 0;
}
```

This must be in camelStyle with TclCode at the end

No camelStyle: only first capital

Fill in the Makefile.am

Let's fill in our `Makefile.am`, that will specify the compilation

```
AM_CXXFLAGS = -Wall -ggdb3

lib_LTLIBRARIES = libuwcoolphy.la

libuwcoolphy_la_SOURCES = initlib.cpp \
        uwcoolphy.cpp

libuwcoolphy_la_CPPFLAGS = @NS_CPPFLAGS@ @NSMIRACLE_CPPFLAGS@
  @DESERT_CPPFLAGS@
libuwcoolphy_la_LDFLAGS =  @NS_LDFLAGS@ @NSMIRACLE_LDFLAGS@
  @DESERT_LDFLAGS@
libuwcoolphy_la_LIBADD = @NS_LIBADD@ @NSMIRACLE_LIBADD@
  @DESERT_LIBADD@

TCL_FILES =  uwcoolphy-init.tcl
initTcl.cc: Makefile $(TCL_FILES)
        cat $(VPATH)/$(TCL_FILES) | @TCL2CPP@ UwCoolPhyTclCode > initTcl.cc
```

Do not put any spaces in Makefiles or config files, *only tabs*. Autotools don't like spaces

A

If an addon: the `@DESERT_LDFLAGS_BUILD@` may be needed

Make DESERT aware

We need to tell the general installer of DESERT about our new module

We need to fill in the `configure.ac` and `Makefile.am` configuration files (in `DESERT_Framework/DESERT`)

```
DESERT_CPPFLAGS="$DESERT_CPPFLAGS "'-I$(top_srcdir)/ranging/uwranging_tokenbus'
DESERT_CPPFLAGS="$DESERT_CPPFLAGS "'-I$(top_srcdir)/ranging/uwranging_tdma'
DESERT_CPPFLAGS="$DESERT_CPPFLAGS "'-I$(top_srcdir)/physical/uwcoolphy'
...
AC_CONFIG_FILES([
Makefile
...
ranging/uwranging_tokenbus/Makefile
ranging/uwranging_tdma/Makefile
physical/uwcoolphy/Makefile
])
```

configure.ac

```
SUBDIRS = m4 \
...
ranging/uwranging_tokenbus \
ranging/uwranging_tdma \
physical/uwcoolphy
```

Makefile.am

configure.ac and autogen

If you're developing and **addon**, also the configure.ac file is needed

Just remember to change the name with the name of your addon

```
AC_INIT(uwcoolphy, 1.0.0)
AM_INIT_AUTOMAKE
AM_PROG_AR

AC_CONFIG_MACRO_DIR([m4])

AC_PROG_CXX
AC_PROG_MAKE_SET

AC_DISABLE_STATIC

...
```

Also, remember to copy the **`autogen.sh`** file from another addon

M4 folder

Copy the entire **`m4`** folder from another addon

Edit the **`desert.m4`** file: put your module dependencies, if any

Both for `AC_ARG_WITH_DESERT`
and `AC_ARG_WITH_DESERT_BUILD`

Compile for first time

Let's see if the compilation files for our module are set up correctly

```
cd ../../..
```

call the installer for the DESERT framework

```
./install.sh --wizard
```

and proceed to install DESERT

To recompile:


```
  DESERT_buildCopy_LOCAL/.buildHost/DESERT_ADDON/uwcoolphy
```

```
  DESERT_Underwater/DESERT_buildCopy_LOCAL/DESERT-3.3.0-build/physical/uwcoolphy
```

Module operations

Let's proceed to add features to our module

- We will implement a physical layer that sets a Packet Error Rate (PER) linear with the distance between transmitter and receiver
- There will be two bounds:
  - `d_min`, at which (and less) PER=0%
  - `d_max`, at which (and more) PER=100%

PER(x) = (x – `d_min`)/(`d_max` – `d_min`), `d_min < x < d_max`

Exercise: suggestions

Few suggestions on how to develop the module

Suggestions: methods

Let's go to our UwCoolPhy class: after having declared the method, we implement the logic

uwcoolphy.h

```
/**
 * Method that computes the PER
 * @param list distance between transmitter and receiver
 */
double getPER(double dist);
```

uwcoolphy.cpp

```
double
UwCoolPhy::getPER(double dist)
{
  if (dist < d_min) {
    return 0;
  } else if (dist > d_max) {
    return 1;
  } else {
    return (dist – d_min)/(d_max – d_min);
  }
}
```

Suggestions: variables

Our previous implementation entails we need some class members holding values for `d_min` and `d_max`

```
double d_min; /**< minimum distance in meters */
double d_max; /**< maximum distance in meters */
```

uwcoolphy.h

```
#include <uwcoolphy.h>

UwCoolPhy::UwCoolPhy(double dmin=0, double dmax=1500)
 : UnderwaterPhysical()
 , d_min(dmin)
 , d_max(dmax)
{
  bind("min_dist", &d_min);
  bind("max_dist", &d_max);
}
```

uwcoolphy.cpp

Suggestions: Tcl

Then, we proceed to make available our new PER method, e.g., making Tcl methods for obtaining the PER

```
int UwCoolPhy::command(int argc, const char *const * argv)
{
  if (argc == 1) {
    if (strcasecmp(argv[1], "getPER")==0) {
      return PER;
    }
  }
  return UnderwaterPhysical::command(argc, argv);
}
```

Suggestions: nuts and bolts

We are going to infer the distance between the node and the transmitter

```
void
UwCoolPhy::endRx(Packet *p)
{
    hdr_cmn *ch = HDR_CMN(p);
    hdr_MPhy *ph = HDR_MPHY(p);

    if (ch->direction() == hdr_cmn::UP) {
        double srcPos = ph->dstPosition;
        double dist = getDist(srcPos);
        getPER(dist);
        // here we can output logs or drop the packet
    }
...
}
```

Finalizing the module

Now that we can compute the PER based on the tx-rx distance, we can add all the features we think are needed or helpful:

- logs, warning messages
- decisions on the physical layers
- AoB