**Successful integration of the WAM robotic wrapper class into H3DAPI**

Ian, Christopher, Sharp

Department of Bioengineering

University of Illinois at Chicago and Rehabilitation Institute of Chicago

Isharp2@uic.edu

**Corresponding author:** Ian, Christopher, Sharp

**Keywords:** (minimum 4, maximum 10)

Robotics, haptics, virtual reality, wrapper class, rehabilitation robotics, neural engineering, H3DAPI, C++

**Short Abstract**: Many robots exist in the world which exhibit similar form and functions. In this paper we outline the successful creation and implementation of a wrapper class which would unify the software commands of the most common functions of all robots.

**Long Abstract:** Many robots exist in the world which exhibit similar forms and functions. In the passed, performing these same functions required different software commands to be implemented for each robot. The repercussion of these developmental discrepancies is the cause of unnecessary developmental overhead time. For example, when a haptical/graphical virtual reality environment has been coded for one specific robot to provide haptic feedback, that specific robot would not be able to be traded for another robot without recoding the program. However, recent efforts are being implemented by the open source community to create universal wrapper classes which would elicit the same response on different robots from a single API. The result of this would lead researchers across the globe, who own different haptical robots to perform the same experiment, from the same source code. Therefore switching out the robot would have no effect on development time, and the velocity of research may remain less inhibited. In this paper we outline the successful creation and implementation of a wrapper class into the H3DAPI which integrates the software commands most commonly used by all robots.

**Protocol:**

1.1) Create a wrapper for HAPI the haptics library by creating your own .cpp and header file. For example we will use the name HAPIWAM.cpp and HAPIWAM.h.

1.2) Place HAPIWAM.cpp into the source directory: HAPI/src

1.3) Place HAPIWAM.h into the header file directory: HAPI/include/HAPI

1.4) At the top of HAPIWAM.h, include the main header file(s) of your robot, in the case of the Barrett WAM, that would be:

```
extern "C" {
#include <include/btwam.h>

#include <HAPI/HAPIHapticsDevice.h>

#include <sys/mman.h>
```

Note: extern “C” is required to resolve compiler mangling, this is because the WAM library is written in ‘C’ and the H3DAPI is written in C++.

1.5) In HAPIWAM.h, create your class and include the following 4 functions

```
bool initHapticsDevice(int);
bool releaseHapticsDevice();
void updateDeviceValues(DeviceValues &dv, HAPITime dt);
void sendOutput(HAPIHapticsDevice::DeviceOutput &d, HAPITime t);
```

1.6) Make sure your class inherits publicly from the HAPIhapticsdevice class by declaring your class as follows:

```
class HAPI_API HAPIWAM : public HAPIHapticsDevice
```

1.7) Create a header guard for your class.

1.8) Create the following attributes for your WAM robot

```
int startDone;

wam_struct *wam;

float velocity[3];

float acceleration[3];

btrt_thread_struct can_thd, wam_thd;

static HapticsDeviceRegistration device_registration;
```

1.9) Lastly, create the following 3 functions which are specific to the WAM’s initialization

```
static void can_thd_function(void *thd);
```

```
static int myWAMCallback(btwam_struct *wam );

static DeviceOutput cb_output;
```

1.10) Now that we have set our prototypes in the header file, open HAPIWAM.cpp and include the following header files and namespace.

```
#include < HAPIWAM.h>

#include <math.h>

#include <unistd.h>

#include <stdio.h>

using namespace HAPI;
```

1.11) Define your constructor and destructor

```
HAPIWAM:: HAPIWAM ()

{

  startDone = FALSE;

}

HAPIWAM::~ HAPIWAM ()

{

}
```

1.12) Define how your device will be registered and your static variable.

```
HAPIHapticsDevice::HapticsDeviceRegistration

HAPIWAM::device_registration("WAM",
          &(newInstance<HAPIWAM>),list<string>());

HAPIWAM::DeviceOutput HAPIWAM::cb_output;
```

1.13) Define your 4 inhereted functions and callbacks.

```
void HAPIWAM::updateDeviceValues(DeviceValues &dv, HAPITime dt)

{//WAM has strange coordinate system - y,z,x

  dv.position = Vec3(wam->Cpos->data[1], wam->Cpos->data[2], wam->Cpos->data[0]);

  dv.velocity = Vec3(wam->Cvel->data[1], wam->Cvel->data[2], wam->Cvel->data[0]);

  dv.force    = Vec3(wam->Cforce->data[1], wam->Cforce->data[2], wam->Cforce-
>data[0]);

  HAPIHapticsDevice::updateDeviceValues(dv,dt);

}



bool HAPIWAM::releaseHapticsDevice()

{

  SetPositionConstraint(wam, FALSE);

  cout << "HAPIWAM:: release haptics device" << endl;

  printf("1. Place the WAM into its home folded position,\n");

  printf("2. Press Shift+idle\n");

  printf("3. Press Enter to end the program\n");

  while(getchar()!='\n') usleep(10000);



  wam_thd.done = TRUE;

  can_thd.done = TRUE;

}



int HAPIWAM::myWAMCallback( btwam_struct *wam )
```

```
{
  setval_v3( wam->Cforce, 0, HAPIWAM::cb_output.force.z );
  setval_v3( wam->Cforce, 1, HAPIWAM::cb_output.force.x );
  setval_v3( wam->Cforce, 2, HAPIWAM::cb_output.force.y );
  apply_tool_force_bot(&wam->robot, wam->Cpoint, wam->Cforce, wam->Ctrq);
}


void HAPIWAM::sendOutput(HAPIHapticsDevice::DeviceOutput &d, HAPITime t)
{
  HAPIWAM::cb_output = d;
}
void HAPIWAM::can_thd_function(void *thd)
{
  int err=0;
  btrt_thread_struct *t = static_cast<btrt_thread_struct*>(thd);
  HAPIWAM *wam_device = static_cast<HAPIWAM*>(t->data);
  /*Initiate robot actuators*/
  err = InitializeSystem();
  printf("System Initialized\n");
  if(err)
  {
    printf("  WAM initialization failed!.\nError: %d\n", err);
    exit(1);
  }
  /*get handle to the WAM bus*/
```

```
  printf("Attempting to load wam.conf from location:
/home/robot/btclient/wam.conf\n");

  if((wam_device->wam = OpenWAM("/home/robot/btclient/wam.conf",0)) == NULL)

  {

    printf(" Open WAM wam.conf failed. Quitting program.\n");

    exit(1);

  }

  printf("..Success\n");

  setSafetyLimits(0,1.5,1.5,1.5); //1.5 m/s velocity limits

  setProperty(0, SAFETY_MODULE, TL2, FALSE, 4700);

  setProperty(0, SAFETY_MODULE, TL1, FALSE, 1800);


  wam_device->startDone = 1; //notify the initialization thread we are complete


  while(!btrt_thread_done((btrt_thread_struct*)t))

              usleep(10000);

  //close system

  CloseSystem();

  //remove thread from realtime scheduler

  btrt_thread_exit((btrt_thread_struct*)t);

}


bool HAPIWAM::initHapticsDevice(int)

{

  int err;
```

```
int buscount;


mlockall(MCL_CURRENT | MCL_FUTURE);
#ifdef RTAI
 rt_allow_nonroot_hrt();
 printf("\nrtai\n");
#endif


printf("Turn on the wam, release the E-stops, press shift-idle and press Enter\n");
while(getchar()!='\n') usleep(10000);


//hard-coding the wam.conf path
char *hard_coded_wam_conf_path = "/home/robot/btclient/wam.conf";
err = ReadSystemFromConfig(hard_coded_wam_conf_path, &buscount);
if(err)
{
 printf("Error reading wam.conf - exiting program...\n");
 exit(1);
}
btrt_thread_create(&can_thd,"rtt",45,(void*)can_thd_function,(void*)this);
while(!startDone)
{ usleep(1000000);}


registerWAMcallback( wam, (void *)HAPIWAM::myWAMCallback );
wam_thd.period = 0.002;
```

```
  btrt_thread_create(&wam_thd, "ctrl", 90, (void*)WAMControlThread, (void*)wam);



  printf("Press Shift-Activate on the WAM control panel and hit return on the
keyboard.\n");

  while(getchar()!='\n') usleep(10000);



  SetGravityComp(wam, 1.0);

  SetCartesianSpace(wam); //Imperative for updateDevice values to work

}
```

Look for "/home/robot/btclient/wam.conf" in the above code and replace this with the actual location of wam.conf on your local machine.

2.1) Now that we have created the wrapper class, we need to compile your wrapper into the HAPI library. Go to HAPI/HAPI/build, and edit CMakeLists.txt and add the dependant libraries after the line that says 'SET(OptionalLibs)'. If you are using the XENOMAI kernel, a PEAK card, and PCI

```
SET(HAVE_WAMAPI 1)
INCLUDE_DIRECTORIES(/home/robot/btclient/include
                    /usr/include
                    /usr/lib
                    /usr/local/lib
                    /lib
              /usr/xenomai/include
              /usr/xenomai
              /usr/xenomai/lib
                    )

#include WAM libraries Ian Sharp
SET(requiredLibs)
SET(requiredLibs ${requiredLibs} ${LDFLAGS}
#WAM common.mk recommended linking order
  -lpthread -lncurses -lm
  /home/machine/btclient/lib/libbtwam.a
```

```
  /home/machine/btclient/lib/libbtsystem.a
  -Wl,-Map=$(TARG).map,--cref
  -lpcan
  -rdynamic
  /usr/xenomai/lib/libnative.a
  -rpath
)
```

Note: If your computer is not named 'robot', then you will need to replace the path '/home/robot/btclient/include' with the location of the btclient/include on your local machine.

2.2) Next go into the HAPI/HAPI/build and type the following 3 commands in this order:

```
cmake .

sudo make

sudo make install
```

3.1) Create the wrapper class for the H3DAPI library. Place WAMDevice.cpp into the source directory: H3DAPI/src

3.2) Place WAMDevice.h into the header file directory: H3DAPI/include/H3D

3.3) WAMDevice.h should contain the following:

```
//////////////////////////////////////////////////////////////////////////////

//    Copyright 2009, Ian Sharp, PhD candidate

//    isharp2@uic.edu

//

//////////////////////////////////////////////////////////////////////////////


#ifndef __WAMDEVICE_H__

#define __WAMDEVICE_H__


#include <H3D/H3DHapticsDevice.h>
```

```
#include <HAPI/HAPIHapticsDevice.h>

extern "C"{

#include <HAPI/HAPIWAM.h>

}

#include <H3D/MFString.h>

#include <H3D/SFString.h>

#include <H3D/SFDouble.h>

#include <H3D/MFVec3f.h>


namespace H3D

{

class WAMDevice: public H3DHapticsDevice{

 public:

   WAMDevice(

         Inst< SFVec3f          > _devicePosition        = 0,

         Inst< SFRotation       > _deviceOrientation     = 0,

         Inst< TrackerPosition  > _trackerPosition       = 0,

         Inst< TrackerOrientation > _trackerOrientation  = 0,

         Inst< PosCalibration   > _positionCalibration   = 0,

         Inst< OrnCalibration   > _orientationCalibration = 0,

         Inst< SFVec3f          > _proxyPosition         = 0,

         Inst< WeightedProxy    > _weightedProxyPosition  = 0,

         Inst< SFFloat          > _proxyWeighting        = 0,

         Inst< SFBool           > _mainButton            = 0,

         Inst< SFBool           > _secondaryButton       = 0,
```

```
        Inst< SFInt32          > _buttons              = 0,
        Inst< SFVec3f          > _force                = 0,
        Inst< SFVec3f          > _torque               = 0,
        Inst< SFInt32          > _inputDOF             = 0,
        Inst< SFInt32          > _outputDOF            = 0,
        Inst< SFInt32          > _hapticsRate          = 0,
        Inst< SFInt32          > _desiredHapticsRate   = 0,
        Inst< SFNode           > _stylus               = 0,
        Inst< SFString         > _deviceName           = 0 );


  virtual void initialize();
  /// Node database entry
  static H3DNodeDatabase database;
  /// The name of the device as specified in the servers.db file.
  auto_ptr< SFString > deviceName;


};
}
#endif
```

3.4) WAMDevice.cpp should look like this:

```
//////////////////////////////////////////////////////////////////////////////
//    Copyright 2009, Ian Sharp, PhD candidate
//    isharp2@uic.edu
```

```
//
//////////////////////////////////////////////////////////////////////////////



#include <H3D/WAMDevice.h>

#include <HAPI/HAPIWAM.h>


using namespace H3D;


H3DNodeDatabase WAMDevice::database("WAMDevice",
&(newInstance<WAMDevice>), typeid( WAMDevice), &H3DHapticsDevice::database );


namespace WAMDeviceInternals

{

  FIELDDB_ELEMENT( WAMDevice, deviceName, INITIALIZE_ONLY );

}


void WAMDevice::initialize()

{

  H3DHapticsDevice::initialize();

  hapi_device.reset(

  new HAPI::HAPIWAM() );

}
```

```
/// Constructor.
WAMDevice::WAMDevice(
        Inst< SFVec3f       > _devicePosition,
        Inst< SFRotation    > _deviceOrientation,
        Inst< TrackerPosition > _trackerPosition,
        Inst< TrackerOrientation > _trackerOrientation,
        Inst< PosCalibration  > _positionCalibration,
        Inst< OrnCalibration  > _orientationCalibration,
        Inst< SFVec3f       > _proxyPosition,
        Inst< WeightedProxy   > _weightedProxyPosition,
        Inst< SFFloat       > _proxyWeighting,
        Inst< SFBool        > _mainButton,
        Inst< SFBool        > _secondaryButton,
        Inst< SFInt32       > _buttons,
        Inst< SFVec3f       > _force,
        Inst< SFVec3f       > _torque,
        Inst< SFInt32       > _inputDOF,
        Inst< SFInt32       > _outputDOF,
        Inst< SFInt32       > _hapticsRate,
        Inst< SFInt32       > _desiredHapticsRate,
        Inst< SFNode        > _stylus,
        Inst< SFString      > _deviceName )
:H3DHapticsDevice( _devicePosition, _deviceOrientation, _trackerPosition,
        _trackerOrientation, _positionCalibration,
        _orientationCalibration, _proxyPosition,
```

```
        _weightedProxyPosition, _proxyWeighting, _mainButton,
            _secondaryButton, _buttons,
        _force, _torque, _inputDOF, _outputDOF, _hapticsRate,
        _desiredHapticsRate, _stylus ),
  deviceName( _deviceName )
 {


 type_name = "WAMDevice";
 database.initFields( this );


 hapi_device.reset(0);
}
```

3.5) Now that the wrapper class has been created, recreate the H3DAPI library with your new classes. Once this is completed, the WAM will be able to be used as a node. Edit CMakeLists.txt in the same way that was performed in step 2.1, only under the directory: H3DAPI/build

3.6) Rebuild the H3DAPI library with the same steps as in 2.3 under the directory H3DAPI/build

```
cmake .

sudo make

sudo make install
```

**Part 7: Representative Results:**

When the protocol is done correctly, then once the <AnyDevice> node is loaded into the H3DViewer or H3DLoad, the WAM device should be recognized and initiated. If the WAM were replaced with another robot, the code itself would not need to be changed.

**Discussion:** It is critical that the extra 'include directories' and libraries are added into the CMakeLists.txt file, otherwise when the HAPI and H3DAPI library are be created, there will be unresolved symbols to your wrapper class functions. Possible modifications to integrating your wrapper class into your latest distribution of the H3DAPI would be to create your wrapper class, and put your wrapper class into a *.so library file. This way, your wrapper class would be isolated from the original H3DAPI distribution. Once your wrapper class is complete, people who do not own a WAM will be able to perform the same virtual reality experiments that you create without changing your source code or their robot.

**Acknowledgments:** There was no funding source for this project.



**References:**

**Tables**

**Figure legends**

**Figures:**