Using the Marine Multivehicle Simulator: uMVS

Paul Newman

March 17, 2009

Abstract

This document will explain how to use the marine multivehicle simulator uMVS

1 Introduction – Global Simulation Mode

There is a mission-file-scope variable simulator . For normal operation (i.e. deployment of a real vehicle) this will be set to be true. However, setting it to false enters MOOS into a different mode – one of simulation.

The idea is that to gain confidence in new code it is a good plan to be able to do dry runs of all the code that will be expected to govern the in-life operation of the vehicle. The simulator flag in conjunction with the behaviour of the instrument applications can achieve this.

Setting the simulator flag to true causes the iProcesses (instruments) to subscribe to one or more SIM_* variables like SIM_X ,SIM_Y ,SIM_DEPTH . These variables are published by a vehicle simulator (see Section 2) and encapsulate the state of a simulated vehicle. The instruments then simulate their output using these values.

So to summarise: running in simulation mode means all the instruments, navigation, Helm applications behave as normal, passing and feeding off the same variables between each other. However, at the lowest level the instrument classes are not talking to hardware via their serial ports etc., but are subscribing to data from a simulated world which they use to generate their measurements.

For example, iGPS normally talks to a GPS sensor via a serial port and outputs GPS_X etc. In simulation mode it also subscribes to SIM_X and SIM_Y which it converts internally (very simply, it turns out, by using the CMOOSVariable class) into GPS_X .

The final part of the story lies with iActuation – when in simulation this process subscribes as usual to DESIRED_RUDDER etc., but instead of sending bytes via a serial port to a lump of hardware, it "re-packages" the commands and bounces them back to the MOOSDB as SIM_DESIRED_RUDDER etc. It is these SIM_ -prefixed control variables that are subscribed to by the simulator and

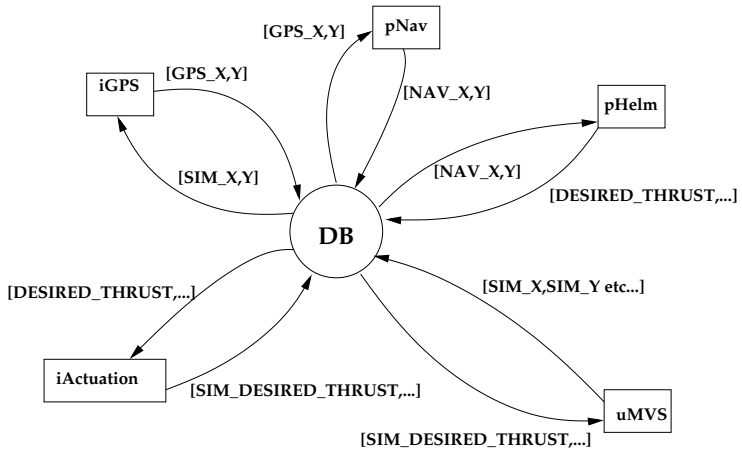

Figure 1: The variable subscription/publication occurring in simulation mode
.

used to control the simulated vehicle. This scenario is illustrated in Figure 1. This shows how messages are bounced around in the simulation mode – notice the additional `SIM_` prefixed subscriptions and publications made by `iGPS` and `iActuation` respectively.

The details of the simulator are described in Section 2.

2 AUV and Acoustic Simulation – `uMVS`

`uMVS` is a multivehicle AUV simulator. It is capable of simulating any number of vehicles and acoustic ranging between them and acoustic transponders. The vehicle simulation incorporates a full 6D.O.F. vehicle model replete with vehicle dynamics, center of buoyancy/ center of gravity geometry, and velocity dependent drag.

The acoustic simulation is also fairly smart. It simulates acoustic packets propagating as spherical shells through the water column. When they intersect with acoustic devices (either on beacons or vehicles) the true time of intersection is calculated by a refinement process. This design allows the real round trip to be calculated when the vehicle is undertakes a trajectory that was not known at the time the initial "ping" was launched.

A typical configuration block is given in Figure 2. The syntax is simple — consider the `ADD_AUV` line in Figure 2. The initial pose of the vehicle is specified as an X, Y, Z, Yaw tuple. The AUV can also be named. The `InputPrefix` and `OutputPrefix` terms are interesting. They allow configuration of the names of the variables which are used to control the actuators of the simulated vehicle and the names of the variables used to describe the state of the vehicle. As in reality, each vehicle is controlled by three actuator settings: rudder, elevator and thrust. On a real vehicle these are typically carried by the variables `DESIRED_THRUST` , `DESIRED_ELEVATOR` and `DESIRED_RUDDER` . Now say a particular simulated vehicle had `InputPrefix = SIM_` , `uMVS` would then subscribe to `SIM_DESIRED_THRUST` , `SIM_DESIRED_ELEVATOR` and `SIM_DESIRED_RUDDER` and use these values as the control parameters for the vehicle. Why might

one like to do this? Well, see Section 1. An alternative use of the prefixes is discussed in Section 2.1.

```
ProcessConfig = uMVS
{
    //add an AUV, starting at Pose (X,Y,Z,Yaw), called AUV1.
    ADD_AUV=  pose=[4x1]{7,3,55,0},name = AUV1,InputPrefix=SIM_,OutputPrefix=SI
    ADD_TRANSPONDER=  name = B1, pose=[3x1]{0,0,0},Rx = CIF, Tx = Ch7,TAT = 0.1

    TideHeight = 60

    //a few variables for the simulator..
    LogFile = SimLog.txt
    InstantLogAcoustics = false

    //what is standard deviation of noise on
    //TOF measurements? 1ms = 1.5 meters
    TOFNoise = 0.00066
}
```

Figure 2: A small configuration block for `uMVS` showing a typical configuration. This would be suitable for use with the topology shown in Figure 1.

```
ProcessConfig = uMVS
{
    ....
    ADD_AUV=  pose=[4x1]{7,3,55,0},name = AUV1,InputPrefix=,OutputPrefix=NAV_
    ....
}
```

Figure 3: Configuring the simulator to work for the scenario shown in Figure 4.

2.1 A More Minimal Simulation

The ability to define the `InputPrefix` and `OutputPrefix` terms for `uMVS` allows a more minimal simulation to be constructed without using the global `simulator` flag discussed in Section 1. In fact, one can eschew the need to use the instruments and `iActuation` altogether. Why would anyone want to do this? Well, the "simulation mode" of Section 1 is invaluable for gaining overall system/architecture confidence; however it can be inconvenient at times. For example, if working on the Helm or action planning process it may be inconvenient to have to launch the instrument processes and the Navigator frequently during development. Figure 4 shows an alternative use of the simulator for exactly this case. The important configuration line (compared to Figure 2) is shown in Figure 3. Note how the simulator is told to publish `NAV_*` data and subscribe to `DESIRED_THRUST` , `DESIRED_ELEVATOR` , `DESIRED_RUDDER` by

having an empty input prefix and NAV_ as an output prefix. A typical topology using this set up is shown in Figure 4.

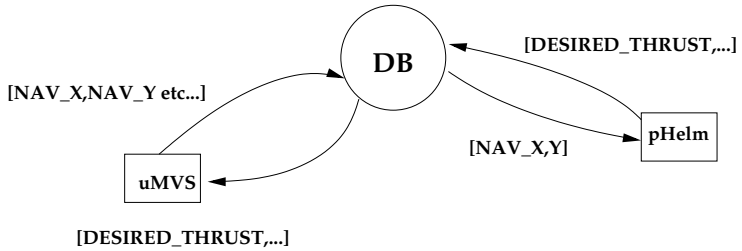

Figure 4: A minimal simulation configuration, for example Helm development. Here uMVS publishes vehicle state as NAV_* data and controls the internal simulated vehicle by direct subscription to the standard Helm outputs.

2.2 Logging

uMVS can be configured (via the LogFile parameter) to write a log file of the simulation. This is a self documenting file that records vehicle state and acoustic ranges for perusal outside of the MOOS environment.

3 Multivehicle Simulation Scenarios

So far we have considered the case of simulating a single vehicle community (one community per vehicle). In the case that there is one mission file for the community, the simulator variable is set to true and all processes run as though this were a real deployment, but behind the scenes the simulator is talking to the instruments to fake reality. This was described in Section 1.

However, what if it was desired to simulate (i.e. prepare for) and experiment with multiple interacting vehicles (i.e. lots of communities), how would this work? Several options are available to the user here and they all involve the use of pMOOSBridge , which is described in the document on MOOSBridge.

One option is to run one simulator for each vehicle (each simulator only running one vehicle) and use pMOOSBridge to bridge whatever variables that need to be shared between communities (it will most likely need to rename the variables as well). For example, consider Figure 5. Here a possible two-vehicle simulation topology is shown. It is created simply by linking two communities, each with their own private uMVS (here each in full simulation mode as described in Section 1 but they could of course be in a reduced form as described in Section 2.1) with an instance of pMOOSBridge as described in the MOOSBridge document.

An alternative approach would be to use the multivehicle simulation capabilities of uMVS and adopt a topology similar to Figure 6. Here each community has its vehicle simulated in a common instance of uMVS which would allow acoustic ranging between vehicles. Figure 7 shows the possible configuration block for uMVS in this scenario. Each vehicle community in Figure 6 runs its own instance of pMOOSBridge to do the relevant data renaming between the "simulating community" and the rest of its own community. For example, with

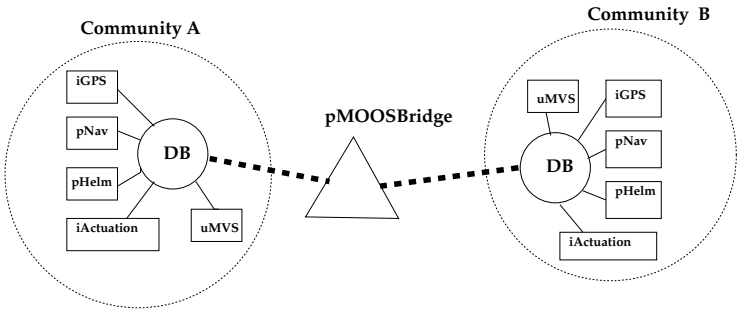

Figure 5: A possible two-vehicle simulation created simply by linking two communities each with a private `uMVS` (here each in full simulation mode as described in Section 1 but they could of course be in a reduced form as described in Section 2.1) with an instance of `pMOOSBridge` .

reference to the configuration snippet in Figure 7 and the topology of Figure 6, the bridge in "Community A" would have to import `AUV_A_X` from the simulation community and map it to `SIM_X` while also exporting `SIM_DESIRED_THRUST` as `AUV_A_DESIRED_THRUST` [1].

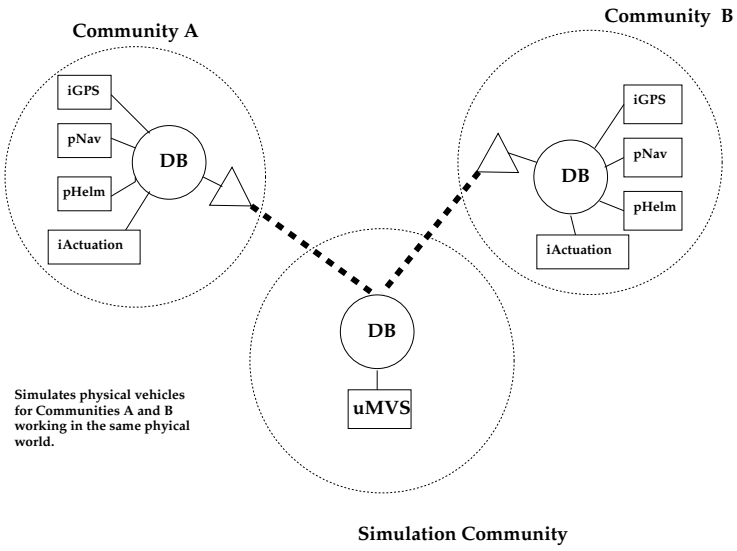

Figure 6: A possible two-vehicle simulation created simply by linking two communities to a single "simulation community" via private instances of `pMOOSBridge` . There is only one `uMVS` running.

Finally, another two alternatives are presented in Figures 8 and 9. Here a single bridge is used to do all the required data routing and name mapping for both communities. In Figure 9, a visualisation community has been added, which uses the `iMatlab` interface to render the simulation in a "fancy fashion". Note that in this case a minimal simulation is being run and so the Bridge will

[1] And similarly for other relevant variables.

```
ProcessConfig = uMVS {
    ....
    ADD_AUV=  pose=[4x1]{7,3,55,0},name = VehA, InputPrefix=AUV_A_, OutputPrefix=
    ADD_AUV=  pose=[4x1]{7,3,55,0},name = VehB, InputPrefix=AUV_A_, OutputPrefix=
    ....
}
```

Figure 7: Configuring the simulator to work for the scenario shown in Figure 6 with two vehicles. Note the values of the input and output prefixes and the message renaming role each `pMOOSBridge` has in Figure 6 because of them
.

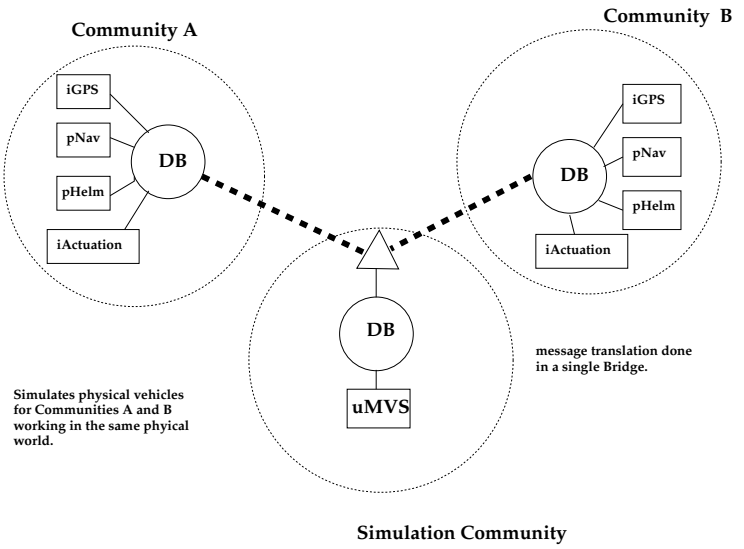

Figure 8: A possible two-vehicle simulation created simply by linking two communities to a single "simulation community" via a *single* instance of `pMOOSBridge` . Again, there is only one `uMVS` running.

be mapping, for example, `AUV_A_SIM_X` on the simulation community to `NAV_X` on community `A` as well as `AUV_B_SIM_X` to `NAV_X` on community `B` .

3.1 Inter-vehicle Ranging

The "ADD_AUV" string can also specify how the vehicle acoustic system responds to acoustic interrogation. By adding something of the form "*ResponderChannel = Ch3,TAT = 0.125*" to the "ADD_AUV" string, the vehicles will act like acoustic beacons (only they move) and respond to "CIF" pulses from other vehicle transceivers on the channel specified with the declared turnaround time. If the "ResponderChannel" is not specified, it will be assumed that inter-vehicle ranging is not wanted and the vehicle's acoustic responders will be turned off.

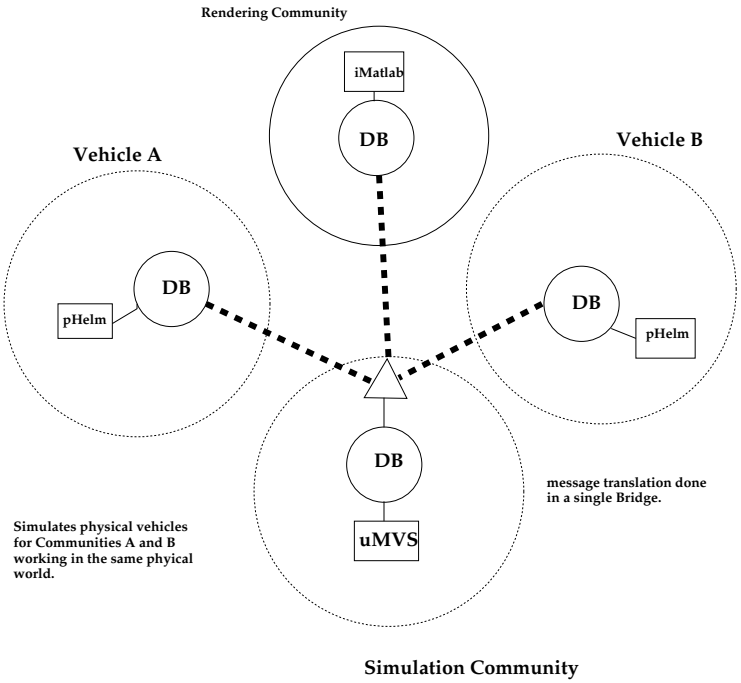

Figure 9: Similar to Figure 8, where a two-vehicle simulation is created simply by linking two communities to a single "simulation community" via a *single* instance of `pMOOSBridge` . Again, there is only one `uMVS` running. The difference between this topology and that shown in Figure 8 is the individual vehicle communities are *not* running in simulation mode (described in Section 1). Instead, in this case, the single instance of `pMOOSBridge` is renaming and routing data such that the need for instruments and the navigator is avoided.