Middleware Usage

Why Do I Need To Use The Middleware?

The Middleware offers an abstraction layer for common HLC tasks and should thus be used in combination with your HLC implementation. More information can be found here (theory). Information on implementation can be found on this page.

Command Line Parameters

Parameter Usage Example

```
./middleware --vehicle_ids=1,7,8 --node_id=middleware --wait_for_start=true --dds_domain=1 --simulated_time=false
```

Important Command Line Parameters

- node_id: *Identification string of the middleware instance in the DDS network*
- simulated_time: *True if the current setup uses simulated time, else false*
- middleware_domain: *Domain ID for communication between middleware and HLC via shared memory, default is 1*
- wait_for_start: *Whether the middleware should wait for the LCC's start signal, should be true (false only for debugging purposes)*
- vehicle_id: *Single ID of a vehicle the middleware is supposed to be responsible for*
- vehicle_ids: *IDs of the vehicles the HLC(s) that the middleware 'controls' are responsible for (usually just one ID)*
- vehicle_amount: *Set amount of vehicles instead of vehicle ID, sets IDs 1 - vehicle_amount (restricted to 256)*
- dds_domain: *Domain ID of the domain in which the LCC, vehicle etc. participate*
- domain_number: *Local domain ID of the middleware and the HLC*
- dds_initial_peer: *Should be explained in Lab Control Center (Usage)*

Important ParameterServer Parameters

- `middleware_period_ms`: *Length of each period in which the HLC is triggered, in milliseconds*

Other Command Line parameters

- offset_nanoseconds: *Set the offset of the communication w.r.t. ~1970 (real time) or 0 (simulated time); should usually not be changed*

QOS_LOCAL_COMMUNICATION.xml.template

The communication between Middleware and HLC takes place locally and should not be shared with the other participants in the network. Thus, this DDS QoS file makes sure that messages via both of them are only exchanged locally and that other participants do not get discovered for that. .template is used as the current IP must be used for this to work, build.bash fills in the template and creates the .xml file in the build folder. This file is also relevant for the HLC scripts, which need to use the same QoS settings to communicate with the Middleware.

How Do I Use The Middleware?

The Middleware is usually started by the LCC (with the right parameters, i.e. vehicle IDs) if you deploy it via the LCC. Don't forget that, after you have deployed the Middleware and after it has been initialized and communicated with the HLC script, you also need to send a start signal from within the timer tab.

You can also start the Middleware manually given the parameters above (a start signal is still required though) - the most important parameters here are:

- *vehicle_ids* You need to tell the Middleware which vehicles your program is responsible for. It will then forward information about the vehicles to your HLC program, and your HLC program can send planned trajectories for these vehicles to the Middleware, which then sends these to the vehicles themselves.
- *middleware_period_ms* This parameter is set using the parameter service (theory, usage). It tells the middleware how often it should send your program new vehicle information, which you are supposed to use to start the computation of a new trajectory (and thus as timing signals).

There are some steps involved in using the Middleware after it has been started. They are mentioned below and are further elaborated in the Matlab Tutorial. Even if you use another programming language, you can use that tutorial as an implementation guideline (as DDS communication should not depend on the programming language).

Communication between HLC and Middleware

Several IDL types are used for the communication between the HLC and the Middleware. **In all cases**, HLC and Middleware communicate using DDS over shared memory with the **DDS Domain ID 1**. Only communication between HLCs should take place on the ID that is used in the rest of the Lab. The following communication types are used in domain 1. They are:

ReadyStatus

After initialization, the Middleware waits for the initialization of the HLC. Afterwards, it tells the LCC that the HLC is ready to operate (i.e. the Middleware appears in the Timer Tab, waiting for a start signal). The Middleware only continues execution (and thus only appears in the Timer Tab) if it receives a ReadyStatus by the HLC. This signal should only be send by your implementation after every possible initialization has taken place, so that, as a next step, you can start with the computation as soon as the HLC tells you to do so.

In the message, only the ID string matters, which must be of the form `"hlc_" + vehicle_id`, where the latter is the ID of the vehicle the HLC is responsible for. As you can see, the Middleware does not expect a single HLC script to be responsible for more than one vehicle. It can communicate with more than one script though (depending on the set vehicle IDs), in which case it waits for all HLCs to send a ReadyStatus before it continues execution.

The ReadyStatus is defined here: https://git.rwth-aachen.de/CPM/Project/Lab/software/-/blob/master/cpm_lib/dds_idl/ReadyStatus.idl

SystemTrigger

The LCC regularly sends timing information in the network if simulated time is used, or only start or stop signals (to start and stop the simulation) in case of real time. The Middleware takes care of all timing signals for you, and also forwards timing information within the VehicleStateList, which you should use to obtain the current system time. But it is not able to stop the HLC directly in case a stop signal is received. Thus, SystemTrigger messages are forwarded to the HLC as well. Your implementation should only look for stop signals regularly, which are defined here. If such a signal is received, the HLC program must be stopped.

CommonroadDDSGoalState

We use commonroad to define scenarios for the simulation. All relevant commonroad information will get passed to the HLC. Currently, this only includes information regarding the goal state. Due to the usage of TransientLocal settings and a restart of the writer for each simulation, the goal state information is always up to date and can always be received in case a commonroad scenario is in use. The structure of the datatype is similar to its Commonroad definition.

You can find out more about it here.

VehicleStateList

These messages are meant as timing signals for your program, but they also include all required information about the current state of all vehicles in the system. You can find out more here:

https://git.rwth-aachen.de/CPM/Project/Lab/software/blob/master/hlc/middleware/idl/VehicleStateList.idl

Thus, you have to receive VehicleStateList messages, which include the current states and observations of the vehicle as well as the current time. These signals are supposed to be the start signal for the HLC, so computation should start using this data after the message was received. You thus need to:

1. Wait for the next VehicleStateList message (and look for stop signals in SystemTrigger)
2. Read the message, use the vehicle and timing information, compute a trajectory for the vehicles your program is responsible for
3. Send the trajectory to the Middleware
4. Go back to step 1

The history for this signal is set to 1, but you may still get an outdated signal here if you missed a period during your computation and read the next VehicleStateList in the middle of the next computation period. In that case, it may be better to skip that period as well and wait for the following one to start. Thus, if directly after you have sent a trajectory there is another VehicleStateList message in the reader, you should ignore that message and wait for the next one.

VehicleCommand[...]

You need to send vehicle command messages as a result of your computation - including the vehicle ID - to the Middleware, which propagates these to the vehicle. The Middleware currently supports VehicleCommandTrajectory, VehicleCommandSpeedCurvature and VehicleCommandDirect. The first of these is the message type that you are most likely to use:

https://git.rwth-aachen.de/CPM/Project/Lab/software/-/blob/master/cpm_lib/dds_idl/VehicleCommandTrajectory.idl