CHALMERS
UNIVERSITY OF TECHNOLOGY

OpenDLV Demo Tutorial

version 1.0

CFSD 18

2017.09.15

Task description

Task 1 Get familiar with Github
1. Fork the original repository to your own account
2. Try to pull the branch (if it's not master branch) to your local folder
3. Commit your changes and push them up to github

Task 2 OpenDLV module creation
1. Create a new branch called 'feature.autobrake'
2. Push it to Github
3. Create a time triggered module named proxy-groundspeed (in 'code/proxy-groundspeed' and with namespace 'opendlv.proxy.cfsd18') with no input, but with the output GroundSpeedReading found in https://github.com/chalmers-revere/opendlv.core/blob/master/resources/odvd/ODVDOpenDLVStandardMessageSet.odvd (this file contains the generic message set, but so far only one of these messages are available in the official master branch of opendlv.core)
4. Create one data triggered module named logic-autobrake (in 'code/logic-autobrake' and with namespace 'opendlv.logic.cfsd18') with GroundSpeedReading as input and ActuationRequest as output (https://github.com/chalmers-revere/opendlv/blob/master/resources/odvd/ODVDOpenDLVData.odvd#L57; this is not yet in the generic message set, and you might need to figure out how to include the ODVDOpenDLVData messages into the module). The logic of the module is that, if the input speed is above a certain threshold (given as configuration) then the there will be a negative acceleration (given in configuration), otherwise a positive acceleration (given in configuration).
5. Make sure it compiles, then commit your changes to you branch and push to Github
6. Create a testcase for the data triggered module where you supply different inputs and check the outputs. See how this is done here: https://github.com/se-research/OpenDaVINCI/blob/master/libopendavinci/testsuites/DataTriggeredConferenceClientModuleTestSuite.h There, you find an example how to create an instance of a module in a Testsuite and sending/receiving data.
7. Create a use case (docker-compose file) to start the two module example. Run the proxy-groundspeed module at 10 Hz.
8. When everything compiles and runs, commit to your branch and then push to Github
9. Create a Pull Request to the branch 'pre-release'

1 Task 1 Get familiar with Github

1.1 Fork a Repository

- Go to the origin repository (on website of Github), click "Fork" on up-right corner, then the origin repository will be forked into your own account. Then you should be able to do anything you want on your own development environment (your forked repository). Your committed changes will also be seen by the author of original repository. You can also create a pull request, and the author of original repository will decide if he would like to merge you changes to original one.

- Go to the forked repository under your account, copy the URL, which you will use to download these files to your local computer.

- Go to the folder in your computer where you would like to download the repository. Download the files by: git clone <URL>. You should be able to see the folder appear, for example "opendlv.cfsd18" in this case.

1.2 Pull a certain branch

- The files you cloned from github are from master branch.

- Go into this folder by: cd <folder name>, for example here "cd opendlv.cfsd18"

- Check the branch on your local environment and which branch are you in by: git branch

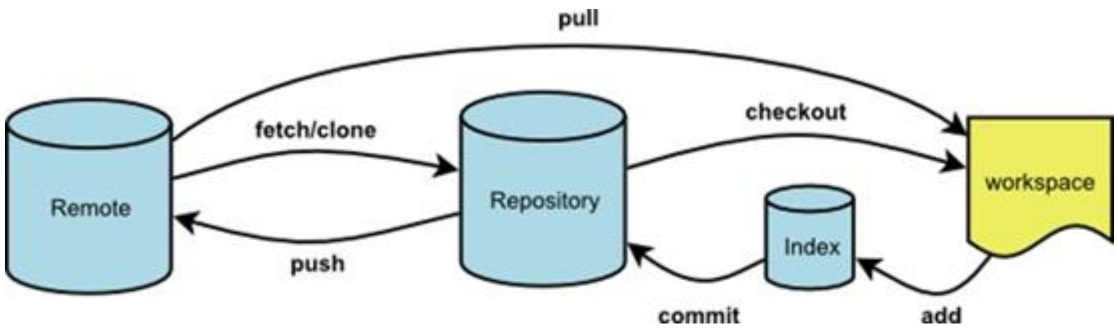
- Create a new branch by: git checkout -b <branch name>, for example, here we use "git checkout -b feature.autobrake". After this you will be automatically switched to this branch.

- The changes you do in this branch will not affect the master branch, so you can later merge the changes to master branch.

- If you want to pull a certain branch from git, for example, the "feature.autobrake" branch in Zijain's repository, use: git pull origin <branch name>.

1.3 Commit your changes and push to git

- Go to the root folder, eg. "opendlv.cfsd18"
- git add . # add all the files
- git commit -m "the message you want to add" # add the description of your changes
- check the URL that you want to add to by: git remote -v
- If no previous pulled repository, you can also add the url by: git remote add origin <URL>
- git push origin <branch name>, for example, branch name can be "master" or "feature.autobrake"
- Type your git username and password



Here is a figure that might help you understand the commands for github

2 Task 2 OpenDLV module creation

Suppose we have already copy the origin master branch from chalmers-revere opendlv.cfsd18 or your forked repository. Go into folder "opendlv.cfsd18"

2.1 Create a time triggered module "groundspeed"

- cd code # change to folder "code"
- cp -r logic-cfsd18 proxy-cfsd18 # copy the folder logic and change its name to proxy
- cd proxy-cfsd18 # go into proxy folder
- gedit CMakeLists # open the CMakeLists in proxy folder, change every "logic" into "proxy", find "# Add subfolders with sources", and add every module within this folder
- Change the name of folder from whatever you have to groundspeed
- cd groundspeed # go into groundspeed folder
- gedit CMakeLists # open the CMakeListes in groundspeed folder, change the module names in it, don't forget to change all logic into proxy
- go into every folder "apps, include, man, src, testsunits", and change all the module names
- NOTE: this is a timeTriggered module, which means the data in this module is sent out at a certain frequency. In time triggered modules, we need to include the head files of TimeTriggeredConferenceClientModule.h, you can find it being included in the head files "groundspeed.hpp" of this groundspeed module. We need to change this into DataTriggered… for a data trigged module. Also note that in time triggered module, the data should be sent out in "body" method. While in data triggered module there should not be a body method. Data are sent out whenever needed (you can image it as infinite high frequency)
- In this specific groundspeed module there are no actual sensor data, so I used a constant value as groundspeed output, which is saved in "m_groundSpeed"
- In body method, send out the data of gourndspeed by:

```
double groundSpeed = m_groundSpeed;

opendlv::proxy::GroundSpeedReading groundSpeedReading;

groundSpeedReading.setGroundSpeed(groundSpeed);
```

```
odcore::data::Container groundSpeedReadingContainer(groundSpeedReading);

getConference().send(groundSpeedReadingContainer);
```

- Don't forget to declare the variables you use in the head file.

2.2 Create a data triggered module "autobrake"

- Logic or control algorithms module should be added in "logic-cfsd18" folder

- Similarly as previous steps, copy the flowbend folder and change everywhere you need to change in CMakeLists in this module and in each folder.

- Since this is a data triggered module, we need to change the TimeTriggered head file in autobrake.hpp into DataTriggered head file.

- NOTE: in time triggered module, if you want to read data from nextContainer while pass it to m_value(global variable) or do something else at the same time, always remember to add mutex::lock in the beginning of nextContainer in order to avoid parallel threads handling conflict. Check previous file for useage of this line of code.

- NOTE: in this module we will use the message type "ActuationRequest" defined in a file called "odvdopendlvdata" in Layer OPENDLV, in which it defined a lot of standard message type, might be useful for us. And this file is not included yet. To be able to use the message type in it, we need to add it, basically we need to change four files to achieve that, and they are:

------------------------------ STEPS to include ODVDOPENDLVDATA-----------------------------------

- docker - completeBuild& incrementalBuild add path of odvopendlvdata in cmake, just behind odvdstandarddataset

- root folder "opendlv.cfds18"- [cmakelists], extranal project add, cmake_args...

- Also in this file,

  add "SET (CMAKE_MODULE_PATH "${ODVDOPENDLVDATA_DIR}/share/cmake-${CMAKE_MAJOR_VERSION}.${CMAKE_MINOR_VERSION}/Modules" ${CMAKE_MODULE_PATH})"

- In folder "logic-cfsd18" -[CMakeLists], add "find package"; also include the directory; then link the library

- include h file of odvdopendlvdata in where it will be used, (in hpp or cpp)

- Go to the cpp file, write a function or whatever you want to implement the logic. Here you can check Zijian's file, in which I send out some acceleration value in message "ActurationRequest"

- It should be able to compile after building there two modules. Go to folder docker, compile and build the image by: `make buildComplete create DockerImage`. (when you only change something in the cpp file, you can also use buildIncremental)

2.3 Create a usecase to run these two modules

- NOTE: to build connections between these two slave micro-servers and the main server, we need the CID of them to be same. There should be two places to check the CID. First, go to docker folder, showhidden files, or in terminal, type gedit .env, in this .env file, check the CID. Then go to the usecase-stable-autobrake, check the CID in .env file as well. They should keep same.

- Create a folder “autobrake” in usecase-stable

- Copy the “configuration”,”docker-compose.yml” and “Dockerfile” from previous projects.

- In configuration files, if you use some configuration values that you would like to preset and then pass into cpp files, you should have the configuration for proxy or logic. Otherwise, just delete these defined values in the end of this file.

- In docker-compose file, should start the micro-servers

```
services:

  odsupercomponent:

    build: .

    network_mode: "host"

    volumes:

    - .:/opt/opendlv.data

    command: "/opt/od4/bin/odsupercomponent --cid=${CID} --verbose=1 --configuration=/opt/opendlv.data/configuration"



  proxy-cfsd18-groundspeeed:

    build: .

    network_mode: "host"

    command: "/opt/opendlv.cfsd18/bin/opendlv-proxy-cfsd18-groundspeed --cid=${CID} --freq=10 --id=1"



  logic-cfsd18-autobrake:

    build: .

    network_mode: "host"

    command: "/opt/opendlv.cfsd18/bin/opendlv-logic-cfsd18-autobrake --cid=${CID} --id=1"
```

- Check the module name and path name to be correct. Note that the time triggered module should define a frequency, while data triggered module don’t.

- In “Docker file”, change the name of image you build as: FROM seresearch/opendlv-cfsd18-on-opendlv-on-opendlv-core-on-opendavinci-on-base-dev:latest

Then you should be able to run everything, if you print out some values, you should be able to see something flashing in the terminal continuously. You can stop it by : Ctrl+C

Hint: to run it, go to usecses-stable-autobrak,
docker-compose up –build
after stopping, remove the docker-compose by: docker-compose rm