Brief on the state of the new framework (version 0.1 alpha)

Eugenio Bargiacchi

January 15, 2014

1 Rationale

The are a number of constraints that this framework needed to fulfill. The current structure is not meant to be final, as what has been done was done only with the purpose of finding out whether a switch to a new framework was possible or not. Since the current framework is still relatively small, such decisions are not meant to be final and are up for discussion.

The problems that I thought were to be addressed are:

- Everything needs to be as modular and as independent as possible. Given the fact that the codebase is bound to be huge for a reason or another, we need to make sure that anybody can go and improve something without forcing him/her to learn the whole thing by scratch. At the same time, the structure of the whole needs to be clearly written somewhere so that it is self-documenting. It must be easy to understand which part goes where, who references it and so on.
- Portability. Hopefully we could use this new framework in different projects. As it is now, the framework is completely platform independent, as it is only concerned with modules and their internal communication.

2 Structure

The idea behind the framework, which is pretty much the same idea behind the other frameworks, is to have multiple threads that concurrently do stuff, and to make them talk. The way the current framework is structured is that we have a Brain class which handles these threads, which are currently

called BrainWave(s). Each BrainWave represents a thread, and you can put as many as you wish in the program. The naoTH (for what I know) uses two (Motion and Cognition), and BHuman (for what I read), has at least 4 (Motion, Cognition, Debug, and a small server detached that moves the robot).

Each BrainWave has the task of managing a bunch of modules, and it calls them in a loop forever. In this framework modules are loaded as shared libraries, which are assumed to be extended from the DynamicModuleInterface class. Another class Blackboard, has the task of making everything communicate. More info about these classes is in the code (you can doxygen it).

The way this framework is intended to be used is to have some big library of data-types, which are simple structs (hopefully no methods) that contain data, and that are used to share informations by multiple modules. This library would reside in a separate repo. Each module would then also reside in their separate repo, and would import the necessary headers from the framework and the big library of data-structures that are needed, and include whatever it needs to do its work. It would then be compiled as a dynamic library, which would then be fed to the framework in order to do its job.

You can see that having modules be dynamic libraries makes it really easy to run arbitrary code in the framework, while at the same time keeping an extremely clean workspace as the runnable project would only need to include the dll's of the modules, the executable of the framework and whatever code you are working with, and not ALL the modules's code.

Currently there are two prototype modules in the repo, in the folder moduleExamples. Each can be compiled using the script in the folder, like so:

```
./compileModule Writer
./compileModule Reader
```

The idea is that their code would in the future reside in different repos. Also it would be cool to have a general-module-repo where a default module is kept with all the possible documentation ever about how to do things, so one can fork that and in 15 minutes already do work on new things with minimal experience.

3 Future Work

There is a LOT to be done. Here is a very incomplete list:

- Networking. The current framework has a console-line interface, but this needs to change for a number of reasons. The first is that our tools use sockets to connect and communicate with the old framework, so if we want to keep (at least a part of) them we need to do that. Secondly it makes easier to develop general tools to interface with the framework, so we could for example have python scripts to handle configurations and loading of modules, and python would simply communicate to the framework what needs to be loaded, options and so on.

- Network protocol. Networking implies some sort of protocol. I'm not sure what BHuman and the others use, we could copy them or make a new protocol altogether, that may be expanded in the future.

- Module Options. Currently there is no direct way to set options for modules. Sure a module can open its own file and read a config from there, but it could be nice to give direct instructions to the modules via the framework.

- Module loading. Currently it is only possible to add modules to the ones already loaded, due to a limitation of run-time type checks upon the data that is requested/provided by the modules. It may be useful to implement this checking in a more complex manner, so that modules could theoretically be moved/removed within a thread's chain at runtime and still have some type checking to avoid random breakage (or we may want to remove typechecking altogether..).

- Framework Options. It may be useful to specify certain conditions for a thread to stop working, keep working and so on. For example, BHuman uses a separate process to move the robot, using information reported via shared memory. It does so so that in the event of a crash the separate process could keep working (for example to display the yellow led of "hey everything crashed"). We could do that, or we could implement a way to say to the framework "hey keep running this thread even if some other thread dies", or "stop this thread if this other thread dies", or "execute this code if this happens".

- Really anything you may want from the framework.