Click Protocol Implementations

Implementing Protocols with Click Modular Router

Bart Braem Johan Bergs

University of Antwerp
iMinds - MOSAIC Research Group

October 2014

Outline

Disclaimer

Everyone has his/her own coding style
This presentation outlines our experiences with coding in Click
Strong suggestions towards our students, may be helpful to others

Click Software Design

Click has two classes that are important for creating new protocols:

- Packet: handles allocation of memory and access to the raw data.
- Element: an abstract class whose children implement specific, preferably bite-sized, actions with Packets.

Protocols are implemented using these basic building blocks configured in the Click script. Your design should take this into account.

Don't start from scratch and try to force your own design into Click.

Working with Packet Formats

Packet formats == structs

- structs are a typical C concept, very low level
- tempting to improve this by wrapping the packets in objects
- attractive to create packet factories

Do not do this, very large overhead:

- In terms of memory and computation (allocate objects, create and delete objects)
- In terms of code base

Use the plain structs

- Requires getting used to
- Straightforward: most packet manipulation is low-level anyway

What about the special cases?

If you really need globally available methods for the struct

- define procedure in file containing the struct (mind the include guards!)

The Infobase

Click architecture is centered around elements, however sometimes information has to be shared between elements

Solutions:

- Global variables
- Handler calls
- Calls on public element methods
- **Create an Infobase element**

Infobase:

- Multiple Elements need to have access to a shared data source
- Infobase Element is only responsible for sharing
- No input or output ports
- Passed to Elements who need it using the configuration

The source-responder pattern

Generating packets can be hard, especially when packet formats must adhere to a standard. Difficult to debug, as packets can get lost in routing complexity.
Solution:

- Debug bit-by-bit, manually
- Introduce a source element, only responsible for packet generation
- MySource -> ToDump(my.dump) -> Discard

Packet generation logic is now in one place, and can easily be tested. Next create a packet responder that parses the packet and responds. When tested plug the elements into a larger Click script.
Other names: generator-receiver, sender-processor

Testing generated packets

Multiple suggested possibilities:

- Avoid plain Print
- Use ToDump, a good generator will allow for easy dumping!
- Take a look at the testie framework (in /test) for Click standard testing

Beware of testing overhead!

God Elements

A single element which implements an entire protocol
Not a good idea:

- Large amount of code in one file
- No reuse of existing Click elements
- Protocol logic becomes hard to debug
- Cooperative development more difficult

Other anti-patterns

- Reinventing the wheel, especially for checksums
- Packet and other factories
- Over-engineering

Conclusion: remain low level!

A big thank you to Michael Voorhaen, one of the original authors of these slides.