Modern Languages and OE

‘There is no magic’

Bruce Ashfield

Overview

- Issues
- Requirements / constraints
- Approaches
- Specific Language Examples
- Future work

Some Things to Note

- Presenting the challenges, requirements and options
- Engage and generate discussion
- Not trying to solve the problem(s), or critique approaches
- I am not an expert in most (or any) of the languages in this talk
- Discussed on the mailing list(s) multiple times, over multiple years

The “problem”

- Many new/modern languages have built-in / language specific dependency management (and tools)
  - Not widely understood by recipe creators (nor should they need to be)
- Integration with the Yocto Project’s features can be challenging
  - More than just source code retrieval and building
- Solutions must be maintainable
- Don’t want to “fight” the upstream
  - often more interested in the spec, versus the tools
- Consider it somewhat like the Bitbake / OE interactions with build systems (meson, etc)

A Few Questions !!

- Scalability and performance
- Recipe clarity
- Transparency of sources
- Debugability / Maintainability
- Reproducible builds
- Long Term support
- Broad usability / Developer Experience
- Developer Workflow
- Licensing / security

Solution requirements

- Many answers require a full / proper integration with the fetcher and related functionality
- Solutions should work for ‘x’ years
- Trusted upstream(s) / proxies / package repositories

General Requirements

- As seen on the OE architecture list:
  - Network:
    - Network access for sources is only expected to happen in the do_fetch step
      - In particular, no access in do_unpack
    - DL_DIR should be in a format suitable for creating an offline build mirror
    - Network access is only performed when explicitly enabled
    - Network access works with the standard linux proxy variables (firewall support)
    - Network access is limited during parsing, and only when explicitly enabled (i.e. AUTOREV)
  - Versions:
    - Revision information is available during parsing (for recipe version construction)
    - Versions must increase in a sortable way (for package feeds)

General Requirements

- As seen on the OE architecture list:
  - Reproducibility:
    - Fetcher output is deterministic
  - Desirable:
    - API to query for possible upgrades to enable automatic upgrade code potential
  - Bitbake
    - selftests for functionality and maintenance
- Not all fetchers support all features (i.e. they don’t make sense), but the guidelines are still valid

Solution Challenges

- Balancing competing requirements
  - Developers versus integrators versus system build versus …
  - Simple recipes (and complex fetchers) versus complex recipes and well tested fetchers
  - Long term maintenance versus bleeding edge
  - Generated versus explicit / static
  - Loosely integrated versus tightly integrated tools
  - Build performance and re-use, versus fine grained dependency management
- Don’t want to reinvent what the languages are doing
  - .. and we also don’t want language fetchers to re-implement what OE does
- While the approaches are similar between languages, the implementations are often quite different
  - Reason to expose the revision control systems backing everything ?
  - Downloads, unpack, etc often use standard tools (wget, git, svn, tar, etc)

Types of solutions

- Flexibility is important
  - Support the different approaches, without being prescriptive
- Dynamic versus static information
- Generated
  - Recipes
  - Data
  - Configuration

Examples in the ecosystem

- npm / npmsw
  - npmsw -> npm fetcher
- rust (cargo)
  - crate:// fetcher
  - meta-rust and cargo-bitbake (crate:// generation)
- go
  - Has changed over time … has it finally reached a ‘stable’ approach ?
- python
  - Largely avoided .. but requirements on very specific package versions is a similar problem
- ruby
  - Gems ?

go

- go has changed dependency management a few times
- proxy.golang.org, sum.golang.org, index.golang.org
  - Will they be around in 10 years ?
- setup parallel / hosted infrastructure for go mod builds ?

go recipes

- Various proposals on the OE architecture list (most recently in May 2022)
  - Generate recipes versus data during build
  - What level of integration with bitbake, classes, etc.
- Packaged dependencies
  - Hundreds or even thousands …
- SRC_URI
  - Explicitly listing source dependencies
  - Reference go.mod/go.sum
- vendor’d build
  - populated by go or by the fetcher
- non-vendor’d / go mod build
  - with env setup for fetching, source organization, etc
- Note: Different solutions can co-exist

meta-virtualization and go

- Meta-virtualization has a number of very large go projects
  - ‘vendoring’ was common, is not so common now
- K3S
  - ~ 515 source dependencies
  - Recipe’s SRC_URI and vendor creation are generated via external script
  - Uses existing fetcher infrastructure during build
- nerdctl
  - ‘Only’ 153 source dependencies
- The current approach works, and has been proven through upgrades, etc
- Recipes are large (but source organization can help with this)

Future Work

- Look for commonality between languages
- Implement core functionality, leaving flexibility for recipes
- Tests and documentation
- Adapt to the inevitable corner cases and language evolution
- Address long term support and reproducibility concerns