Region Digraphs

Basing Subsystems on Framework hooks

Glyn Normington

Background

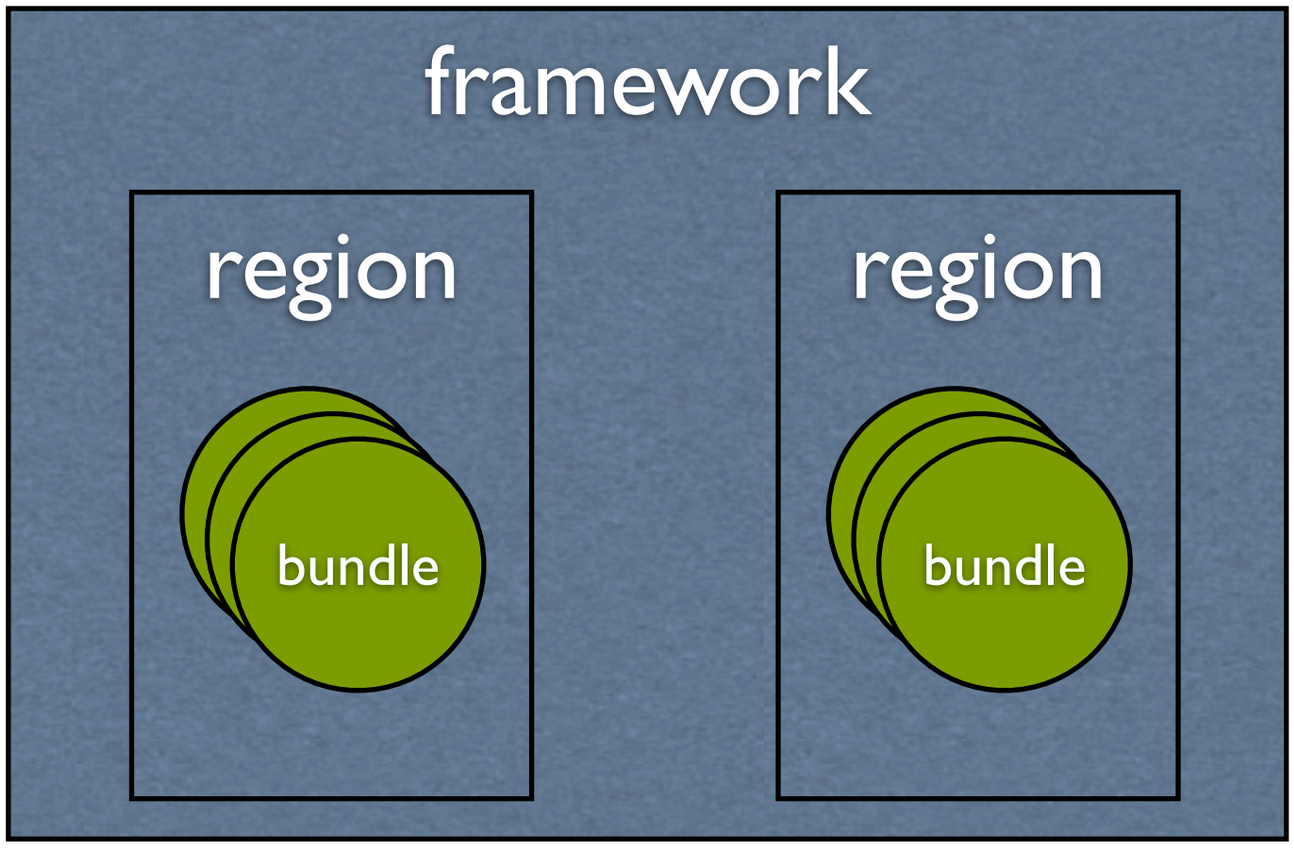
- Virgo isolates “regions”:
  - Kernel
  - Applications in a “user region”
- Virgo 2.1 uses a composite bundle
  - Deprecated!
- Virgo 3.0 will use framework hooks

Framework Hook Usability

- Experimented with direct use of framework hooks
  - Hard to ensure consistency
  - Various limitations, e.g. in region membership
  - Difficult to use, e.g. plan scoping would require additional hook instances
- New implementation uses an intermediate directed graph ('digraph') abstraction
  - Hooks use the digraph for consistency
  - Limitations removed
  - Simple digraph API

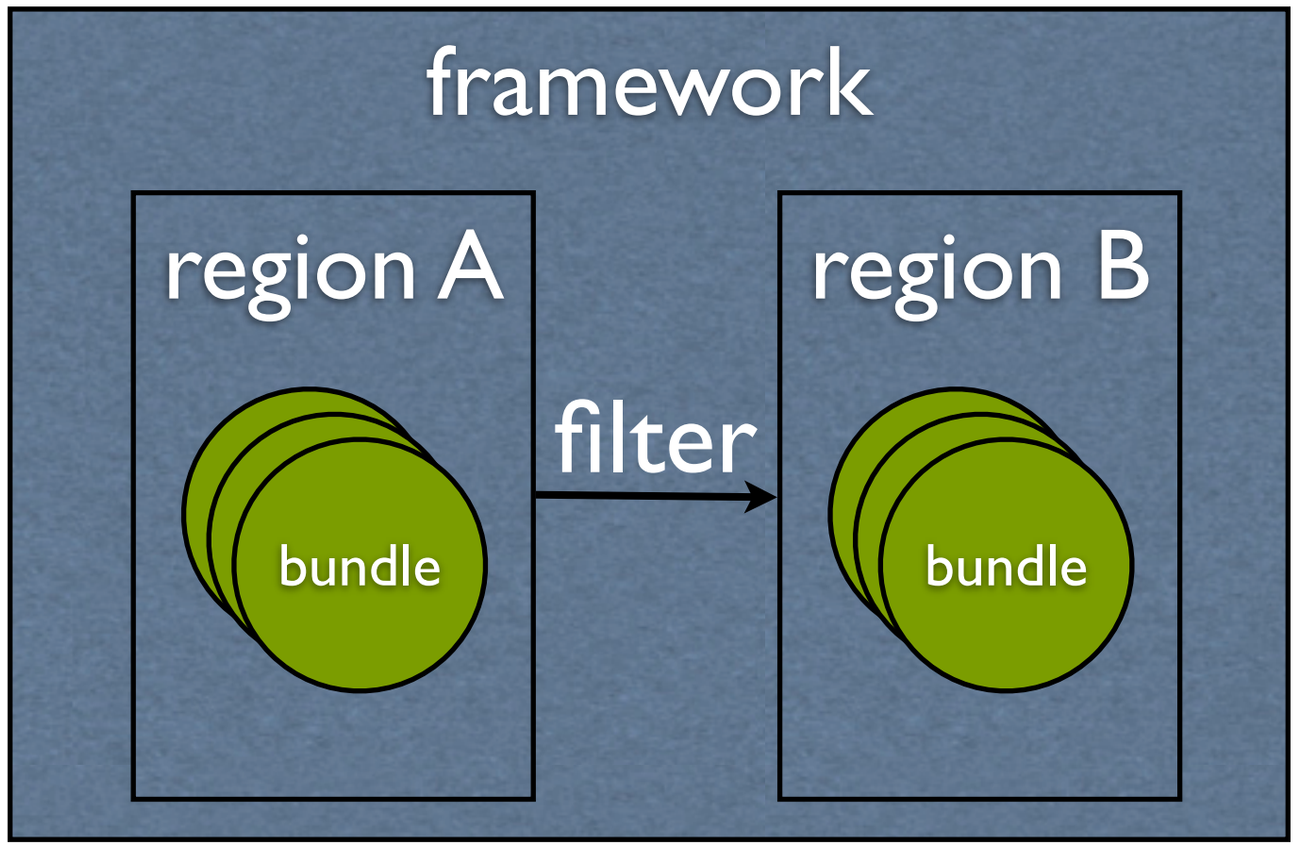
Regions

- A *region* is a collection of bundles
- The regions partition the bundles in the framework:
  - each bundle belongs to a region
  - regions do not overlap
- To install the "same" bundle in two distinct regions, distinct locations must be used



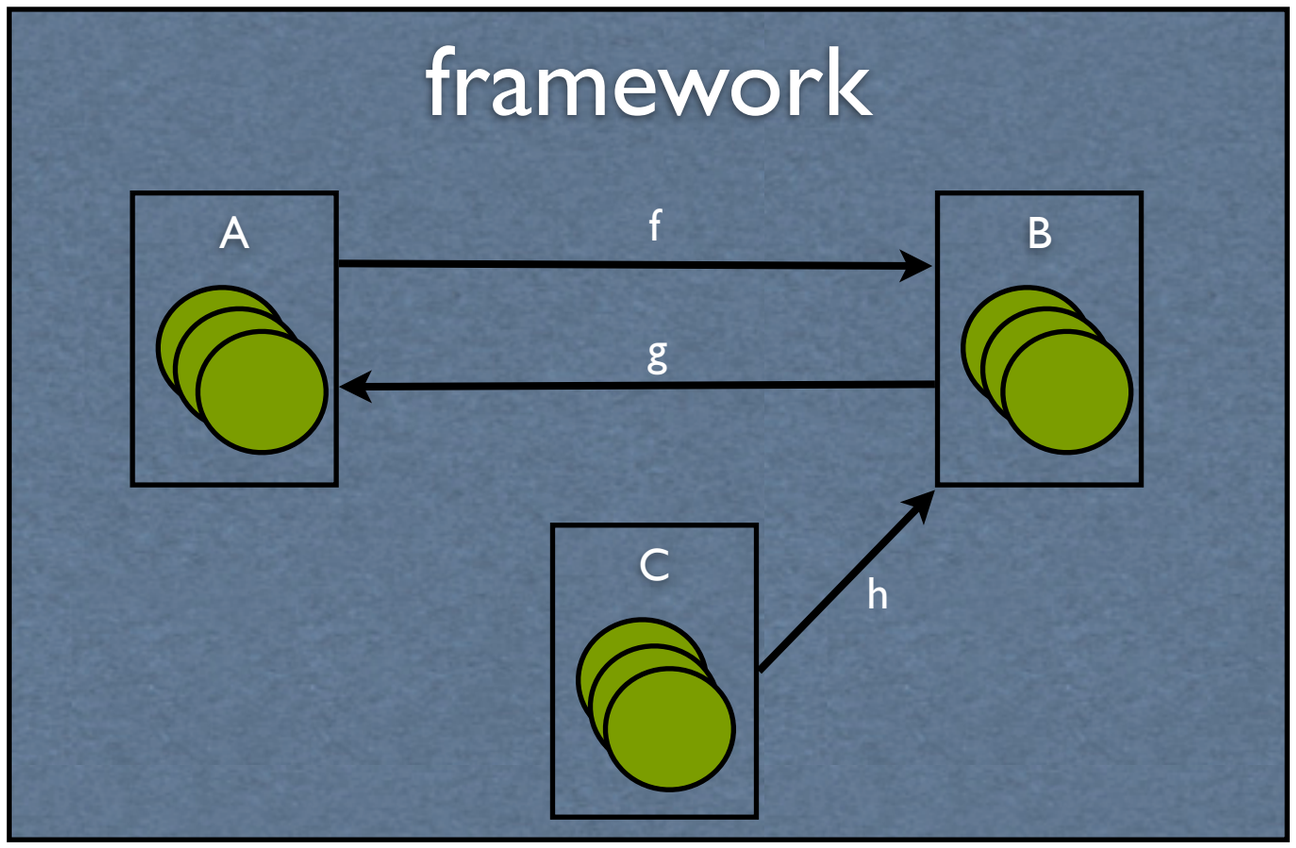
Connections

- Regions can only see the "contents" of other regions via *connections*
  - bundles
  - packages
  - services
- Each connection has a filter

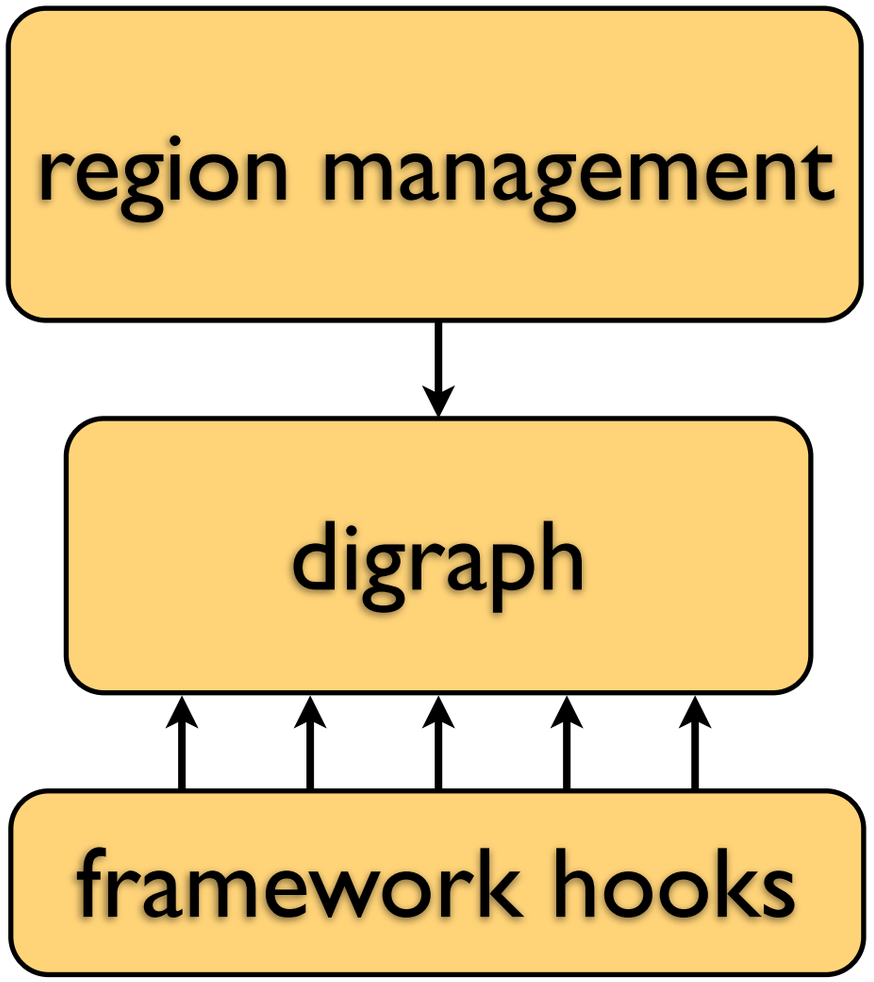


Digraph

- Regions and connections form a digraph
- Cycles are necessary (but harmless)



Layering



Observations

- Bundle locations need to be made ‘unique’, e.g. by prepending region name
- Bundle event hook is useful for assigning newly installed bundles to regions
- Resolution diagnostics must respect regions

Conclusions

- Framework hooks are usable as a replacement for composite bundles
  - Function more flexible
  - Isolation somewhat weaker

Conclusions

- Digraph sufficient basis for Virgo's region support
  - Appears sufficient for plan scoping too
- Digraph addresses subsystem scoping requirements

References

- Formal model (with English intro)
  - **http://tinyurl.com/virgoregions**
- Eclipse bug 330776
  - https://bugs.eclipse.org/bugs/show_bug.cgi?id=330776