Regions in Virgo (v0.81)

Glyn Normington
Steve Powell

February 10, 2011

This document provides a formal model of how Virgo divides the OSGi framework into a directed graph of *regions*.

The model will inform the work on **Bug 330776** "Re-implement user region using framework hooks instead of nested framework".

The model is defined using the Z specification language. An introduction provides an informal description of regions and how they influence framework hooks for readers who are not familiar with Z.

We are grateful to Rob Harrop for reviewing this model and raising several corrections and points of clarification and to Dave Kemper for pointing out an inconsistency.

Contents

1 Introduction

The Virgo kernel is isolated from applications by the use of *regions*. The kernel runs in its own region and applications run in a *user region*.

Virgo 2.1.0 implemented the user region as a nested framework, but Equinox has deprecated the nested framework support in favour of *framework hooks* which are being defined for OSGi 4.3. **Bug 330776** re-implements the user region using the OSGi framework and service registry hooks.

Framework hooks are used to limit which bundles can 'see' particular bundles and exported packages, and service registry hooks are used to limit which bundles can 'see' particular services. 'Seeing' includes both finding and being notified via lifecycle events.

Rather than allowing arbitrary hook behaviour, we limit the hooks to operate on regions which are connected together with filters.

A *region* is then a set of bundles and a region can see bundles, exported packages, and services from another region via a *connection*. Each connection has a *filter* which may limit what can be seen across the connection. Hence regions and connections form a directed graph decorated by filters.

For example, Figure 1 shows three regions connected by three connections. Each connection has a filter which limits what bundles, exported packages, and services are visible across the connection.

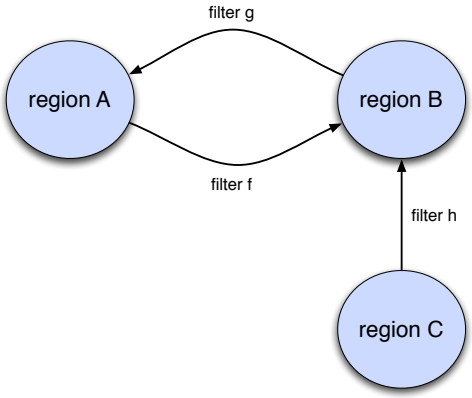

Figure 1: Connected Regions

A connection may be thought of as an import. So region C imports from region B. The imports are filtered, so filter h may limit what region C sees from region B. Similarly region A imports from region B through filter f and region B imports from region A through filter g.

Unlike OSGi package imports between bundles, imports between regions are transitive. So region C can see bundles, exported packages, and services from region A, subject to filters g and h.

We now consider visibility from the perspective of each of the framework hooks.

1.1 Bundle Find Hook

The bundle find hooks limits the bundles that a bundle in a given region sees when listing bundles using `BundleContext.getBundles`.

For example, Figure 2 shows a bundle W which finds bundles W, X, and Y. It does not find Z which is filtered out.

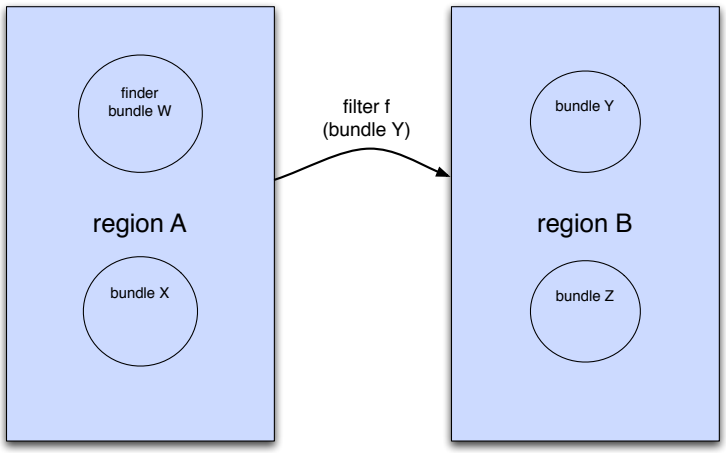

Figure 2: Bundle Find

1.2 Bundle Event Hook

The bundle event hooks limits the bundle events that a bundle listener in a given region receives.

For example, Figure 3 shows a bundle W which has a bundle listener. Bundle W's bundle listener receives events for W, X, and Y. It does not receive events for bundle Z which is filtered out.

1.3 Service Find Hook

The service find hooks limits the services that a bundle in a given region sees when looking up services.

For example, Figure 4 shows a bundle W which can look up services s and t but not u which is filtered out.

1.4 Service Event Hook

The service event hooks limits the service events that a service listener in a given region receives.

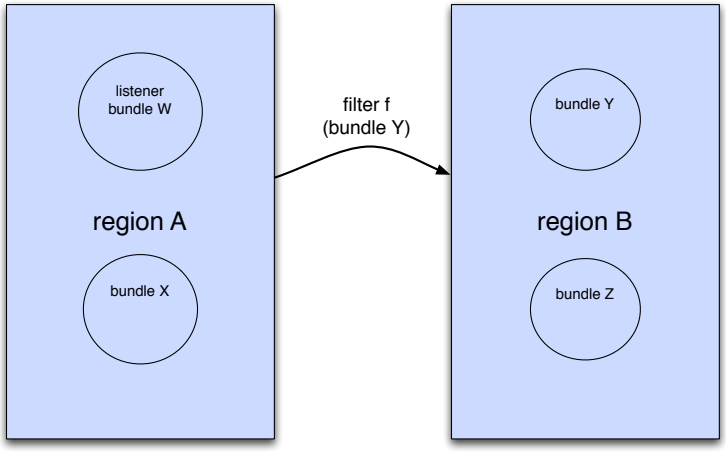

Figure 3: Bundle Event

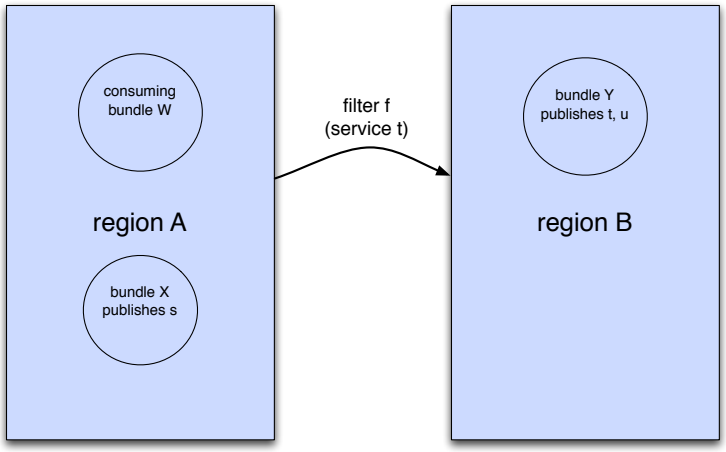

Figure 4: Service Find

For example, Figure 5 shows a bundle W which has a service listener. Bundle W's service listener receives events for s and t. It does not receive events for service u which is filtered out.

1.5 Resolver Hook

The resolver hook limits the exported packages that the bundles in a given region may wire to, depending on the region containing the bundle that exports each candidate exported package and the filters between the regions.

For example, Figure 6 shows a bundle Z being resolved which imports packages p and q. Bundle X in region B exports both p and q while bundle Y in region

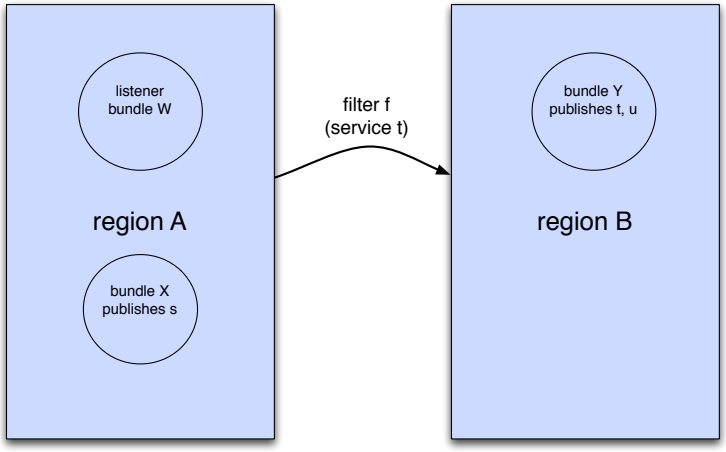

Figure 5: Service Event

B exports only p.

Region A is connected to region B with a filter that allows only package p to be seen by region A. The net effect is that the import of p may be satisfied by either bundle X or Y but the import of package q may not be satisfied by bundle X since q is filtered out.

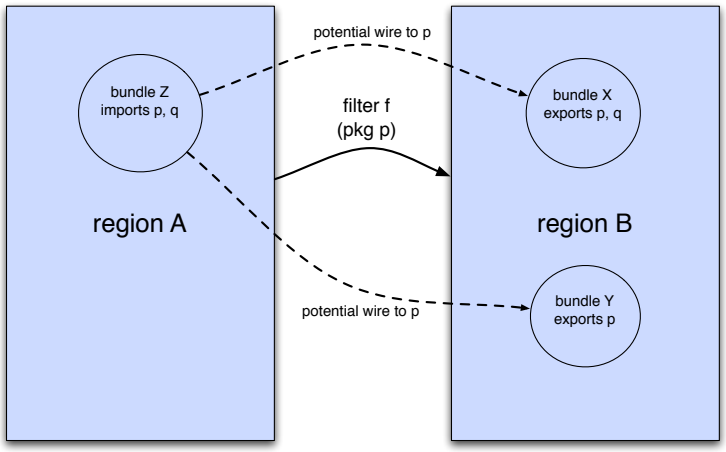

Figure 6: Package Filtering

Another example in Figure 7 shows a package p transitively visible through two filters via an intermediate region.

Bundle Z may wire to bundle X or bundle Y for package p, but not for packages q and r which are both filtered out on the way from C to A.

The remaining chapters provide a formal, albeit partial, model of bundles, re-

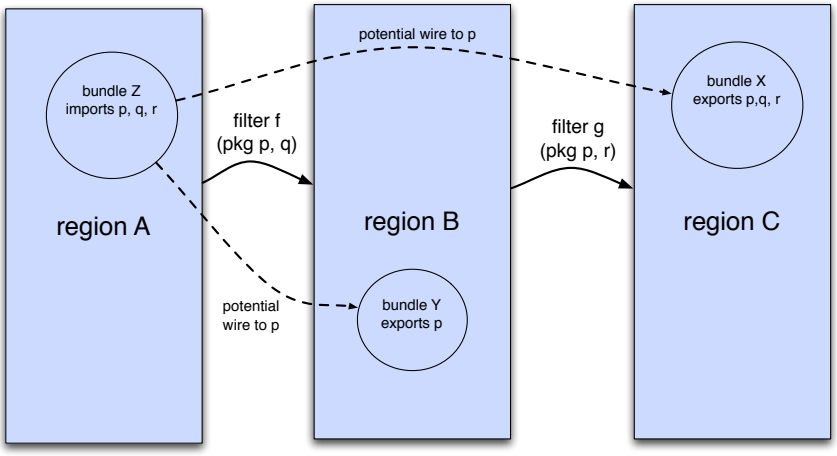

Figure 7: Transitive Package Filtering

gions, filters, and how regions are connected together, via filters, into a directed graph. The model finally indicates how it may be implemented using framework hooks.

2 Basic Types

Some basic types need defining.

Bundles

Bundles are identified by a bundle symbolic name and bundle version, but uniquely identified in the OSGi framework by a bundle location.

$[BSN, BVer, BLoc]$

Packages

We abstract the notion of package version and the attributes associated with package imports and exports.

$[Package]$

Services

We abstract all the details of services.

$[Service]$

Regions

Regions are identified by a region identifer.

$[RId]$

3 Bundle

A bundle has a bundle symbolic name and bundle version, exports zero or more packages, and publishes zero or more services.

$$
\begin{array}{|l}
\textit{Bundle} \\
\hline
name : BSN \\
version : BVer \\
location : BLoc \\
exportedPackages : \mathbb{P}\, Package \\
publishedServices : \mathbb{P}\, Service \\
\hline
\end{array}
$$

It is unusual, and usually bad practice, to include the primary key (in this case the location) of an entity in the entity's schema. We do it here to ensure that all bundles are distinct and use this property later when determining which region a bundle belongs to.

We define a helper function for extracting a pair consisting of the bundle symbolic name and bundle version from a bundle.

$$
\begin{array}{|l}
bid : Bundle \rightarrow (BSN \times BVer) \\
\hline
bid = (\lambda\, Bundle \bullet (name, version)) \\
\end{array}
$$

4 Region

We build up to a definition of a region in several small steps.

4.1 Proto-Region

A proto-region contains a set of bundles which are installed in the framework and associated with the proto-region either by being added to the proto-region or by being imported into the proto-region.

$$
\begin{array}{|l}
\hline
\textit{ProtoRegion} \\
\textit{bundles} : \mathbb{P}\, \textit{Bundle} \\
\hline
\end{array}
$$

4.2 Indexed Region

An indexed region adds an index, and thereby some constraints, to a proto-region.

The index is a function identifying the bundles in the region by location.

$$
\begin{array}{|l}
\hline
\textit{IndexedRegion} \\
\textit{ProtoRegion} \\
l : \textit{BLoc} \rightarrowtail \textit{Bundle} \\
\hline
l = \{b : \textit{bundles} \bullet b.\textit{location} \mapsto b\} \\
\hline
\end{array}
$$

The signature imposes the constraint of injectivity on the index. Thus an indexed region contains a set of bundles each of which is uniquely identified by its location.

4.3 OpenRegion

An open region is an indexed region with some derived sets of packages exported by bundles in the region and services published by bundles in the region.

$$
\begin{array}{|l}
\hline
\textit{OpenRegion} \\
\textit{IndexedRegion} \\
\textit{localPkg} : \mathbb{P}\, \textit{Package} \\
\textit{localSvc} : \mathbb{P}\, \textit{Service} \\
\hline
\textit{localPkg} = \bigcup \{b : \textit{bundles} \bullet b.\textit{exportedPackages}\} \\
\textit{localSvc} = \bigcup \{b : \textit{bundles} \bullet b.\textit{publishedServices}\} \\
\hline
\end{array}
$$

We allow a given exported package to be associated with more than one bundle, which can happen if a bundle uses `require-bundle` with the reexport option. We also allow a given service to be published by more than one bundle, which can never happen, but which causes no difficulty for this model and would complicate the model if we constrained it.

4.4 LinkedRegion

A linked region is an open region with additional sets of imported packages and services.

$$
\begin{array}{|l}
\textit{LinkedRegion} \\
\hline
\textit{OpenRegion} \\
\textit{importedPkg} : \mathbb{P}\, \textit{Package} \\
\textit{importedSvc} : \mathbb{P}\, \textit{Service} \\
\hline
\end{array}
$$

The imported packages and services come into play later when regions are filtered and combined because it will be possible to filter out a bundle but not its exported packages or published services. We want to represent the result as a region for simplicity, so the region needs to have a way of recording packages and services which are 'detached' from their bundle(s).

We do not need a corresponding set of imported bundles because these can be stored in *bundles*.

4.5 Region

A region is a linked region in which we derive sets of all packages and all services.

$$
\begin{array}{|l}
\textit{Region} \\
\hline
\textit{LinkedRegion} \\
\textit{pkg} : \mathbb{P}\, \textit{Package} \\
\textit{svc} : \mathbb{P}\, \textit{Service} \\
\hline
\textit{pkg} = \textit{localPkg} \cup \textit{importedPkg} \\
\textit{svc} = \textit{localSvc} \cup \textit{importedSvc} \\
\hline
\end{array}
$$

The local and imported packages are not necessarily disjoint and neither are the local and imported services. This is important as packages exported by and services published by imported bundles are classified as both local and imported.

We define a nil region.

$$
\begin{array}{|l}
\textit{NIL} : \textit{Region} \\
\hline
\textit{NIL.bundles} = \varnothing \\
\textit{NIL.importedPkg} = \varnothing \\
\textit{NIL.importedSvc} = \varnothing \\
\end{array}
$$

4.6 Adding a Bundle to a Region

Bundles are typically added to regions *under* the bundle install operation so that the bundle is associated with its region by the time the install completes.

A bundle may also be added to a region after the bundle is installed, although this opens up a window in which the bundle does not belong to any region.

A bundle can be added to a region provided the region does not already contain a bundle with the given bundle's location.

RegionAddBundleOk

$$\Delta Region$$
$$b? : Bundle$$

$$b?.location \notin \operatorname{dom} l$$
$$bundles' = bundles \cup \{b?\}$$
$$importedPkg' = importedPkg$$
$$importedSvc' = importedSvc$$

Note that $localPkg'$, $localSvc'$, pkg', and svc' are determined by the constraints of $Region'$.

4.7 Combining Regions

Although we will not support operations to combine regions, it is necessary to define how to combine regions so we can discuss the behaviour of directed graphs of regions later.

Regions with consistent indexing may be combined to produce a composite region using the infix $\sqcup$ operator.

$$\_ \sqcup \_ : Region \times Region \nrightarrow Region$$

$$(\_ \sqcup \_) = (\lambda\, r1, r2 : Region \bullet (\mu\, Region \mid bundles = r1.bundles \cup r2.bundles \land pkg = r1.pkg \cup r2.pkg \land svc = r1.svc \cup r2.svc))$$

The precondition of $\sqcup$ is that no bundle in one region has the same location as a bundle in the other region.

$\sqcup$ is idempotent, commutative, and associative (where defined) and *NIL* acts as a zero.

$$\begin{aligned}\vdash \forall\, r, s, t : Region \mid \{(r,s),(s,t),(t,r)\} \subseteq \operatorname{dom}(\_ \sqcup \_) \bullet\\ (r,r) \in \operatorname{dom}(\_ \sqcup \_) \land r \sqcup r = r \land\\ (s,r) \in \operatorname{dom}(\_ \sqcup \_) \land r \sqcup s = s \sqcup r \land\\ (r, s \sqcup t) \in \operatorname{dom}(\_ \sqcup \_) \land (r \sqcup s, t) \in \operatorname{dom}(\_ \sqcup \_) \land r \sqcup (s \sqcup t) = (r \sqcup s) \sqcup t \land\\ (r, NIL) \in \operatorname{dom}(\_ \sqcup \_) \land r \sqcup NIL = NIL\end{aligned}$$

We define the set of all pairwise consistent regions

$$ConsistentRegionPairs == \operatorname{dom}(\_ \sqcup \_)$$

and the set of all pairwise consistent finite sets of regions.

$$ConsistentRegionSets == \{f : \mathbb{F}\, Region \mid \forall\, r, s : f \bullet (r,s) \in ConsistentRegionPairs\}$$

Since $\sqcup$ is commutative, associative, idempotent, and has a zero, we define a distributed form.

$$\bigsqcup : \mathbb{F}\, Region \pfun Region$$

$$\bigsqcup \varnothing = NIL \land$$
$$(\forall\, r : Region;\ f : \mathbb{F}\, Region \mid \{r\} \cup f \in ConsistentRegionSets \bullet$$
$$\quad \bigsqcup(\{r\} \cup f) = r \sqcup \bigsqcup f)$$

5 Multiple Regions

A system of multiple regions has a finite, indexed collection of regions, a convenience function of all the region identifiers in the system, and a function for determining the region identifier of any bundle in the system.

Regions

$$reg : RId \pfun Region$$
$$allBundles : \mathbb{P}\, Bundle$$
$$rids : \mathbb{P}\, RId$$
$$breg : Bundle \pfun RId$$

$$allBundles = \bigcup \{r : \operatorname{ran} reg \bullet r.bundles\}$$
$$rids = \operatorname{dom} reg$$
$$breg = \{b : Bundle;\ rid : rids \mid b \in (reg\ rid).bundles \bullet (b, rid)\}$$

Since *breg* is a function, no bundle can belong to more than one region. This is a constraint on the regions in the collection expressed by the functionality of *breg* rather than by a quantification over the bundles in the regions.

5.1 Determining a Bundle's Region

We expose the convenience function as an operation for determining a bundle's region.

GetRegionOk

$$\Xi Regions$$
$$b? : Bundle$$
$$r! : Region$$

$$b? \in allBundles$$
$$r! = reg(breg\ b?)$$

5.2 Promoting Region Operations

We define a fairly standard promotion schema.

PromoteRegion

$$\Delta Regions$$
$$\Delta Region$$
$$rid? : RId$$

$$rid? \in \operatorname{dom} reg$$
$$\theta Region = reg\ rid?$$
$$reg' = reg \oplus \{rid? \mapsto \theta Region'\}$$

Then we promote the operation to add a bundle.

$$RegionsAddBundleOk \mathrel{\widehat=} (\exists\, \Delta Region \bullet PromoteRegion \land RegionAddBundleOk)$$

6 Filters

A filter specifies sets of bundles, exported packages, and services.

$$
\begin{array}{|l}
\textit{Filter} \\
\hline
bf : \mathbb{P}(BSN \times BVer) \\
pf : \mathbb{P}\, Package \\
sf : \mathbb{P}\, Service \\
\hline
\end{array}
$$

We define some helper functions to perform filtering.

The *xb* function applies a filter to a bundle by filtering its exported packages and published services.

$$
\begin{array}{|l}
xb : Bundle \times Filter \rightarrow Bundle \\
\hline
xb = (\lambda\, b : Bundle;\ f : Filter \bullet \\
\quad (\mu\, Bundle \mid \\
\qquad name = b.name \land \\
\qquad version = b.version \land \\
\qquad location = b.location \land \\
\qquad exportedPackages = b.exportedPackages \cap f.pf \land \\
\qquad publishedServices = b.publishedServices \cap f.sf))
\end{array}
$$

The *fb* function passes a set of bundles through a filter and then applies the filter to the remaining bundles.

$$
\begin{array}{|l}
fb : \mathbb{P}\, Bundle \times Filter \rightarrow \mathbb{P}\, Bundle \\
\hline
fb = (\lambda\, bs : \mathbb{P}\, Bundle;\ f : Filter \bullet \{b : bs \mid (bid\ b) \in f.bf \bullet xb(b, f)\})
\end{array}
$$

The *fp* function passes a set of exported packages through a filter.

$$
\begin{array}{|l}
fp : \mathbb{P}\, Package \times Filter \rightarrow \mathbb{P}\, Package \\
\hline
fp = \{ps : \mathbb{P}\, Package;\ f : Filter \bullet ((ps, f), ps \cap f.pf)\}
\end{array}
$$

The *fs* function passes a set of published services through a filter.

$$
\begin{array}{|l}
fs : \mathbb{P}\, Service \times Filter \rightarrow \mathbb{P}\, Service \\
\hline
fs = \{ss : \mathbb{P}\, Service;\ f : Filter \bullet ((ss, f), ss \cap f.sf)\}
\end{array}
$$

We also define the most permissive filter.

$$
\begin{array}{|l}
TOP : Filter \\
\hline
TOP.bf = BSN \times BVer \\
TOP.pf = Package \\
TOP.sf = Service
\end{array}
$$

6.1 Filtering Regions

We define an infix $\downarrow$ operator to apply a filter to a region and produce another region.

$$
\begin{array}{|l}
\_ \downarrow \_ : \mathit{Region} \times \mathit{Filter} \rightarrow \mathit{Region} \\
\hline
(\_ \downarrow \_) = (\lambda\, r : \mathit{Region};\ f : \mathit{Filter} \bullet \\
\qquad (\mu\, \mathit{Region} \mid \\
\qquad\quad \mathit{bundles} = \{b : r.\mathit{bundles} \mid (\mathit{bid}\ b) \in f.\mathit{bf} \bullet \mathit{xb}(b, f)\} \land \\
\qquad\quad \mathit{pkg} = r.\mathit{pkg} \cap f.\mathit{pf} \land \\
\qquad\quad \mathit{svc} = r.\mathit{svc} \cap f.\mathit{sf}))
\end{array}
$$

$\downarrow$ is total since packages and services not filtered out which are exported or published by a bundle which *is* filtered out end up in the resultant region's imported packages and imported services sets, respectively.

The most permissive filter can be applied to any region without effect.

$$\vdash \forall\, r : \mathit{Region} \bullet r \downarrow \mathit{TOP} = r$$

Filters can be applied in any order with the same effect.

$$\vdash \forall\, r : \mathit{Region};\ f, g : \mathit{Filter} \bullet (r \downarrow f) \downarrow g = (r \downarrow g) \downarrow f$$

$\downarrow$ distributes over $\sqcup$ (where defined).

$$
\begin{array}{l}
\vdash \forall\, f : \mathit{Filter};\ r, s : \mathit{Region} \bullet \\
\qquad (r \sqcup s) \downarrow f = (r \downarrow f) \sqcup (s \downarrow f)
\end{array}
$$

7 Connected Regions

We now connect up regions to form a directed graph.

Regions are connected by filters. Every region is connected to itself by the most permissive filter. So the bundles in a region can see all the bundles, exported packages, and services in that region.

ConnectedRegions

$$
\begin{array}{l}
Regions \\
filter : RId \times RId \pfun Filter \\
squash : RId \pfun Region \\
\hline
first(\!| \operatorname{dom} filter |\!) = rids \\
second(\!| \operatorname{dom} filter |\!) = rids \\
(\forall rid : rids \bullet filter(rid, rid) = TOP) \\
\operatorname{dom} squash = rids \\
(\forall rid : rids \bullet \\
\quad (\textbf{let } r == reg\ rid \bullet \\
\quad\quad squash(rid) = r \sqcup \bigsqcup \{r2 : RId \mid (rid, r2) \in \operatorname{dom} filter \land r2 \neq rid \bullet \\
\quad\quad\quad squash(r2) \downarrow filter(rid, r2)\}))
\end{array}
$$

The *squash* function is well defined but this isn't immediately obvious. Since there are finitely many regions, the $\bigsqcup$ expression in the constraint is well defined. Also, we need to be sure that there is a *squash* function which satisfies the constraint. This turns out to be the case because if we start at a given region r and then form all the non-looping paths to other regions, then we can apply all the filters along each non-looping path and then use $\bigsqcup$ to combine the results to form $squash\, r$ observing that visiting the same region via a longer, looping path does not affect the result since more filters will be applied compared to the non-looping path to that region.

7.1 Connecting Regions Together

We define an operation to connect two regions with a filter.

ConnectOk

$$
\begin{array}{l}
\Delta ConnectedRegions \\
r?, s? : RId \\
f? : Filter \\
\hline
(r?, s?) \notin \operatorname{dom} filter \\
\theta Regions' = \theta Regions \\
filter' = filter \cup \{(r?, s?) \mapsto f?\}
\end{array}
$$

7.2 Promoting Multiple Region Operations

We promote the add bundle operation and add a constraint that the bundle being added must not have a bundle symbolic name and version which is present

in a filter associated with the region the bundle is being added to.

$$
\begin{array}{|l}
\hline
CRAddBundleOk \\
\Delta ConnectedRegions \\
RegionsAddBundleOk \\
\hline
filter' = filter \\
(\forall\, s : RId;\ f : Filter \mid (rid?, s) \mapsto f \in filter \bullet \\
\quad (bid\ b?) \notin f.bf) \\
\hline
\end{array}
$$

We also promote the get region operation for completeness.

$$CRGetRegionOk \mathrel{\widehat=} \Xi ConnectedRegions \land GetRegionOk$$

8 Framework Hooks

We now describe the behaviour of framework hooks sufficient to implement our model of connected regions.

BundleFindHook

$\Xi ConnectedRegions$
$finder? : Bundle$
$candidates? : \mathbb{P}\, Bundle$
$found! : \mathbb{P}\, Bundle$

$finder? \in allBundles$
$found! = candidates? \cap (squash(breg\ finder?)).bundles$

BundleEventHook

$\Xi ConnectedRegions$
$listeners? : \mathbb{P}\, Bundle$
$eb? : Bundle$
$fl! : \mathbb{P}\, Bundle$

$listeners? \subseteq allBundles$
$fl! = \{l : listeners? \mid eb? \in (squash(breg\ l)).bundles\}$

ResolverHookFilterMatches

$\Xi ConnectedRegions$
$requirer? : Bundle$
$candidates? : \mathbb{P}\, Package$
$filtered! : \mathbb{P}\, Package$

$requirer? \in allBundles$
$filtered! = candidates? \cap (squash(breg\ requirer?)).pkg$

ServiceFindHook

$\Xi ConnectedRegions$
$finder? : Bundle$
$candidates? : \mathbb{P}\, Service$
$found! : \mathbb{P}\, Service$

$finder? \in allBundles$
$found! = candidates? \cap (squash(breg\ finder?)).svc$

ServiceEventHook

$\Xi ConnectedRegions$
$listeners? : \mathbb{P}\, Bundle$
$es? : Service$
$fl! : \mathbb{P}\, Bundle$

$listeners? \subseteq allBundles$
$fl! = \{l : listeners? \mid es? \in (squash(breg\ l)).svc\}$

9 Z Notation

Numbers:

$\mathbb{N}$ `Natural numbers` {0,1,...}

Propositional logic and the schema calculus:

$\ldots \land \ldots$	And	$\langle\!\langle \ldots \rangle\!\rangle$	Free type injection
$\ldots \lor \ldots$	Or	$[\ldots]$	Given sets
$\ldots \Rightarrow \ldots$	Implies	$', ?, !_{0 \ldots 9}$	Schema decorations
$\forall .. \mid .. \bullet ..$	For all	$\ldots \vdash \ldots$	theorem
$\exists .. \mid .. \bullet ..$	There exists	$\theta \ldots$	Binding formation
$\ldots \setminus \ldots$	Hiding	$\lambda \ldots$	Function definition
$\ldots \mathrel{\widehat=} \ldots$	Schema definition	$\mu \ldots$	Mu-expression
$\ldots == \ldots$	Abbreviation	$\Delta \ldots$	State change
$\ldots ::= \ldots \mid \ldots$	Free type definition	$\Xi \ldots$	Invariant state change

Sets and sequences:

$\{\ldots\}$	Set	$\ldots \setminus \ldots$	Set difference
$\{.. \mid .. \bullet ..\}$	Set comprehension	$\bigcup \ldots$	Distributed union
$\mathbb{P} \ldots$	Set of subsets of	$\# \ldots$	Cardinality
$\varnothing$	Empty set	$\ldots \subseteq \ldots$	Subset
$\ldots \times \ldots$	Cartesian product	$\ldots \subset \ldots$	Proper subset
$\ldots \in \ldots$	Set membership	$\ldots$ partition $\ldots$	Set partition
$\ldots \notin \ldots$	Set non-membership	seq	Sequences
$\ldots \cup \ldots$	Union	$\langle \ldots \rangle$	Sequence
$\ldots \cap \ldots$	Intersection	disjoint $\ldots$	Disjoint sequence of sets

Functions and relations:

$\ldots \leftrightarrow \ldots$	Relation	$\ldots \mapsto \ldots$	maplet
$\ldots \nrightarrow \ldots$	Partial function	$\ldots^{\sim}$	Relational inverse
$\ldots \rightarrow \ldots$	Total function	$\ldots^{*}$	Reflexive-transitive closure
$\ldots \rightarrowtail \ldots$	Partial injection	$\ldots(\!\mid \ldots \mid\!)$	Relational image
$\ldots \rightarrowtail \ldots$	Injection	$\ldots \oplus \ldots$	Functional overriding
dom $\ldots$	Domain	$\ldots \lhd \ldots$	Domain restriction
ran $\ldots$	Range	$\ldots \lhd \ldots$	Domain subtraction

Axiomatic descriptions:

Declarations

Predicates

Schema definitions:

SchemaName

Declaration

Predicates