Regions in Virgo (v0.9)

Glyn Normington
Steve Powell

March 7, 2011

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

The model informed 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 **1**

1.1 Bundle Find Hook 2

1.2 Bundle Event Hook 2

1.3 Service Find Hook 2

1.4 Service Event Hook 2

1.5 Resolver Hook 3

2 Basic Types **6**

3 Bundle **7**

4 Region **8**

4.1 Proto-Region 8

4.2 Indexed Region 8

4.3 OpenRegion 8

4.4 LinkedRegion 8

4.5 Region 9

4.6 Adding a Bundle to a Region 9

4.7 Combining Regions 10

5 Multiple Regions **12**

5.1 Determining a Bundle's Region 12

5.2 Promoting Region Operations 12

6 Filters **14**

6.1 Filtering Regions 15

7 Connected Regions **16**

7.1 Connecting Regions Together 16

7.2 Adding a Bundle to a Region 17

7.3 Determining a Bundle's Region 17

8 Framework Hooks **18**

9 Z Notation **20**

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 R4.3. **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.

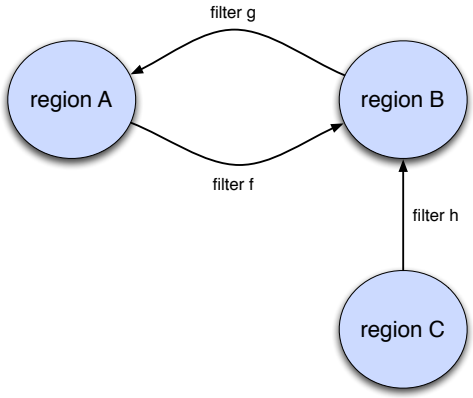

Figure 1: Connected Regions

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 hook 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.

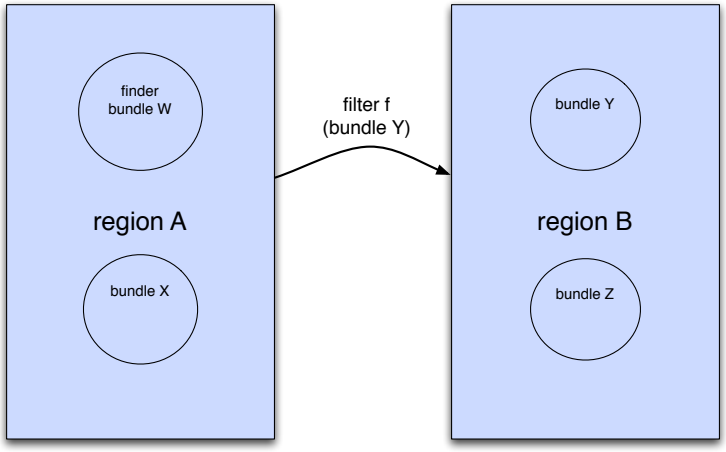

Figure 2: Bundle Find

1.2 Bundle Event Hook

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

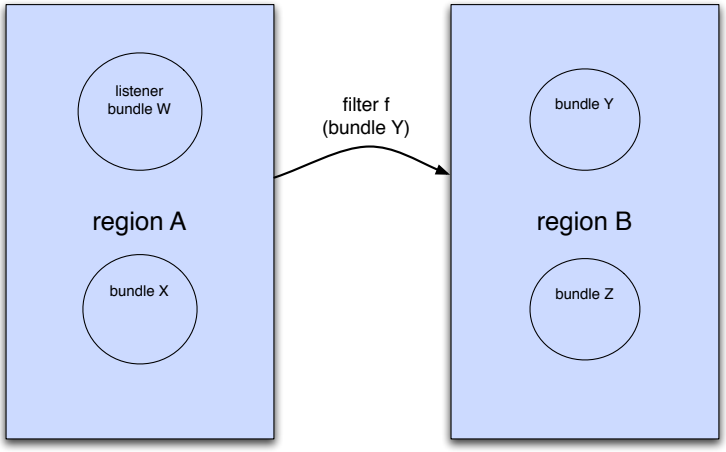

Figure 3: Bundle Event

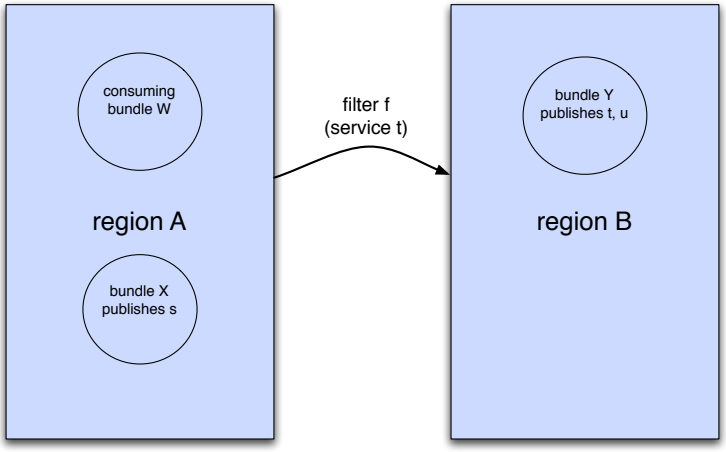

Figure 4: Service Find

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

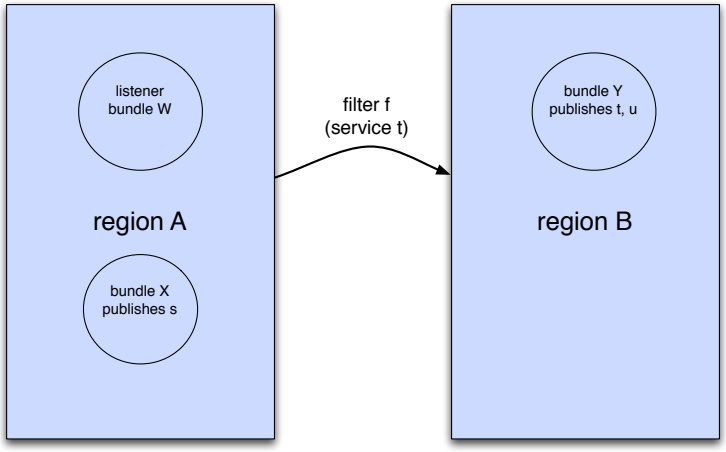

Figure 5: Service Event

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.

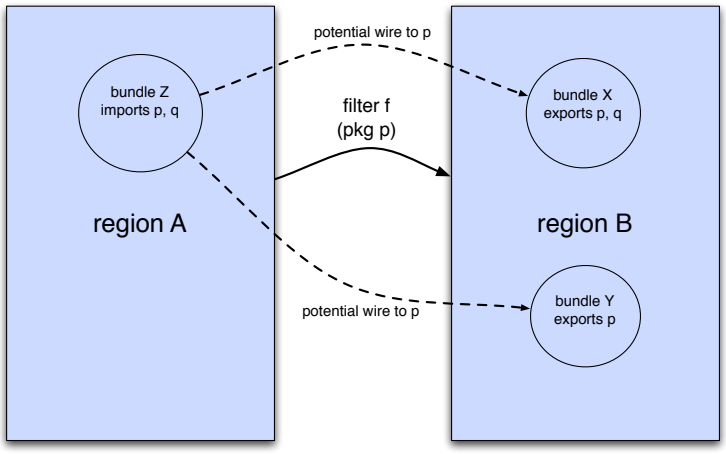

Figure 6: Package Filtering

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-

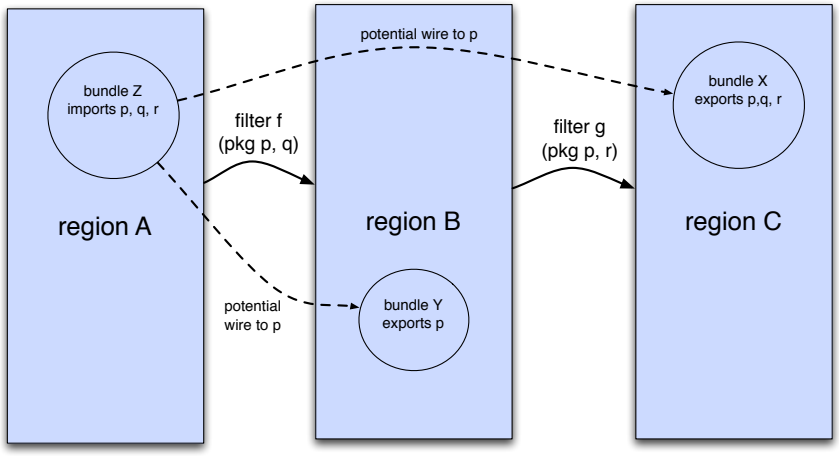

Figure 7: Transitive Package Filtering

gions, filters, and how regions are connected together, via filters, into a directed graph. The model finally defines the behaviour of framework hooks in terms of the directed graph of regions.

2 Basic Types

Some basic types define the primitives used in the model.

Bundles

Bundles are uniquely identified in the OSGi framework by a bundle location, whose details are not important here.

[*BLoc*]

Note that, as of OSGi R4.3, when the framework property `org.osgi.framework.bsnversion` is set to `multiple`, the combination of bundle symbolic name and bundle version is no longer guaranteed to uniquely identify a bundle in the OSGi framework.

Packages

We do not need the notion of package version and the attributes associated with package imports and exports, so we simply declare a set of package objects.

[*Package*]

Services

Neither do we need details of services.

[*Service*]

3 Bundle

A bundle has a location, exports zero or more packages, and publishes zero or more services.

Bundle

$$
\begin{array}{l}
location : BLoc \\
exportedPackages : \mathbb{P}\, Package \\
publishedServices : \mathbb{P}\, Service
\end{array}
$$

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.

4 Region

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

4.1 Proto-Region

A proto-region is a set of bundles. These are intended to represent the bundles installed in the framework and associated with the proto-region.

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

4.2 Indexed Region

An indexed region is a proto-region with an index of a certain sort.

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

$$\begin{array}{|l} \hline \textit{IndexedRegion} \\ \textit{ProtoRegion} \\ l : \textit{BLoc} \rightarrowtail\!\!\!\!\!\shortmid\;\, \textit{Bundle} \\ \hline l = \{ b : \textit{localBundles} \bullet b.\textit{location} \mapsto b \} \\ \hline \end{array}$$

The signature imposes the constraint of injectivity on the index. Thus an indexed region is a set of bundles each of which is uniquely identified by its location. Notice that **all** the bundles in the proto-region are indexed.

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.

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

4.4 LinkedRegion

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

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

The imported bundles, 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).

4.5 Region

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

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

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 with no local or imported bundles, packages, or services.

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

4.6 Adding a Bundle to a Region

Bundles are typically added to regions *as part of* 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 l', $localPkg'$, $localSvc'$, 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.

$$\_ \sqcup \_ : Region \times Region \pfun Region$$

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

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

We now describe a system comprising multiple regions.

Regions are identified by a region identifier, whose details are not important here.

$$[RId]$$

The system of multiple regions has a collection of regions indexed by region identifier. The collection is finite so that certain operations, described later, are well defined.

The system has convenience sets of all the bundles and of all the region identifiers in the system. These sets are derived from the indexed collection.

The system also has a function for determining the region identifier of any bundle in the system from the bundle's location.

Regions

$$\begin{array}{l} reg : RId \pfun Region \\ allBundles : \mathbb{P}\, Bundle \\ rids : \mathbb{P}\, RId \\ breg : BLoc \pfun RId \\ \hline allBundles = \bigcup \{ r : \operatorname{ran} reg \bullet r.bundles \} \\ rids = \operatorname{dom} reg \\ breg = \{ b : Bundle;\ rid : rids \mid b \in (reg\ rid).bundles \bullet (b.location, rid) \} \end{array}$$

Since *breg* is a function, no bundle location can belong to more than one region. Since a bundle's location uniquely identifies the bundle in its region, it follows that a bundle's location uniquely identifies the bundle in the system of multiple regions.

5.1 Determining a Bundle's Region

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

GetRegionOk

$$\begin{array}{l} \Xi Regions \\ b? : Bundle \\ r! : Region \\ \hline b? \in allBundles \\ r! = reg(breg\ b?.location) \end{array}$$

5.2 Promoting Region Operations

We define a fairly standard schema for promoting operations on a single region into operations on the system of multiple regions. The region to be operated on

is singled out using a region identifier and all other regions are left unchanged.

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.

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

This operation has the signature below, which shows that it is now an operation on the system of multiple regions.

RegionsAddBundleOk

$$\Delta Regions$$
$$rid? : RId$$
$$b? : Bundle$$

$$\ldots$$

6 Filters

So far the regions in the system are independent of each other. When we later connect the regions together, we will need to define which bundles, packages, and services are visible across each connection from one region to another. We'll do this using a *filter*.

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

$$
\begin{array}{|l}
\textit{Filter} \\
\hline
bf : \mathbb{P}\, \textit{Bundle} \\
pf : \mathbb{P}\, \textit{Package} \\
sf : \mathbb{P}\, \textit{Service} \\
\hline
\end{array}
$$

We define some helper functions to perform filtering.

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

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

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

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

The *bundleView* function produces a filtered view of a bundle by applying a filter to the bundle's exported packages and published services.

$$
\begin{array}{|l}
\textit{bundleView} : \textit{Bundle} \times \textit{Filter} \rightarrow \textit{Bundle} \\
\hline
\textit{bundleView} = (\lambda\, b : \textit{Bundle};\ f : \textit{Filter} \bullet \\
\quad (\mu\, \textit{Bundle} \mid \\
\qquad \textit{location} = b.\textit{location} \land \\
\qquad \textit{exportedPackages} = \textit{filterPackages}(b.\textit{exportedPackages}, f) \land \\
\qquad \textit{publishedServices} = \textit{filterServices}(b.\textit{publishedServices}, f)))
\end{array}
$$

The *filterBundles* function passes a set of bundles through a filter. The result is the set of filtered views of the bundles allowed by the filter.

$$
\begin{array}{|l}
\textit{filterBundles} : \mathbb{P}\, \textit{Bundle} \times \textit{Filter} \rightarrow \mathbb{P}\, \textit{Bundle} \\
\hline
\textit{filterBundles} = (\lambda\, bs : \mathbb{P}\, \textit{Bundle};\ f : \textit{Filter} \bullet \{b : bs \cap f.bf \bullet \textit{bundleView}(b, f)\})
\end{array}
$$

We also define the most permissive filter.

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

6.1 Filtering Regions

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

$$
\begin{array}{|l}
\_ \downarrow \_ : \mathit{Region} \times \mathit{Filter} \rightarrow \mathit{Region} \\
\hline
(\_ \downarrow \_) = (\lambda\, r : \mathit{Region};\ f : \mathit{Filter} \bullet \\
\qquad\qquad (\mu\, \mathit{Region} \mid \\
\qquad\qquad\quad \mathit{bundles} = \mathit{filterBundles}(r.\mathit{bundles}, f) \land \\
\qquad\qquad\quad \mathit{pkg} = \mathit{filterPackages}(r.\mathit{pkg}, f) \land \\
\qquad\qquad\quad \mathit{svc} = \mathit{filterServices}(r.\mathit{svc}, f)))
\end{array}
$$

$\downarrow$ 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).

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

7 Connected Regions

We now connect up regions to form a directed graph.

Certain pairs of distinct regions are connected by filters. The net effect of connecting a region to other regions is that the region may then be able to see bundles, exported packages, and services in other regions. This net effect is modelled as a collection of textitnet regions.

Every region can be considered to be 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.

RegionDigraph

$$
\begin{array}{l}
Regions \\
filter : RId \times RId \pfun Filter \\
net : RId \pfun Region \\
\hline
\dom filter \subseteq (rids \times rids) \setminus \mathrm{id}\, RId \\
\dom net = rids \\
(\forall\, rid : rids \bullet \\
\quad net\, rid = (reg\, rid) \sqcup \bigsqcup \{ r : RId \mid (rid, r) \in \dom filter \bullet (net\, r) \downarrow filter(rid, r) \})
\end{array}
$$

The *net* 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 *net* 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 *net 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 distinct regions with a filter.

ConnectOk

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

7.2 Adding a Bundle to a Region

Adding a bundle to a region in the directed graph adds the bundle to a region in the system of multiple regions and affects no connections or filters. We add a constraint that the bundle being added must not be present in a filter associated with the region the bundle is being added to. This is to avoid a bundle in a region clashing with a bundle from a connected region.

CRAddBundleOk

$$\Delta RegionDigraph$$
$$RegionsAddBundleOk$$

$$filter' = filter$$
$$(\forall\, s : RId;\ f : Filter \mid (rid?, s) \mapsto f \in filter \bullet$$
$$\quad b? \notin f.bf)$$

7.3 Determining a Bundle's Region

A bundle's region in the directed graph is the same as the bundle's region in the syste, of multiple bundles. This operation does not affect any connections or filters.

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

8 Framework Hooks

We can now describe the behaviour of framework hooks in terms of our directed graph of regions.

A bundle 'find' operation, such as `BundleContext.getBundles`, is returned only the candidates in the net region of the bundle performing the operation.

That is, the bundle performing a bundle 'find' operation is returned the bundles in its own region and bundles in connected regions that are allowed by the corresponding filters.

BundleFindHook

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

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

A bundle listener receives only events for bundles in the net region of the bundle that was used to register the listener.

That is, the bundle listener receives events for bundles in the region used to register the listener and for bundles in connected regions that are allowed by the corresponding filters.

BundleEventHook

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

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

A bundle being resolved may wire only to exported packages and bundles in the net region of the bundle.

That is, the bundle may wire to exported packages and bundles in its own region and to exported packages and bundles in connected regions that are allowed by the corresponding filters.

ResolverHookFilterMatches

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

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

A service 'find' operation is returned only the candidates in the net region of the bundle performing the operation.

That is, the bundle performing a service 'find' operation is returned the services in its own region and services in connected regions that are allowed by the corresponding filters.

ServiceFindHook

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

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

A service listener receives only events for services in the net region of the bundle that was used to register the listener.

That is, the service listener receives events for services in the region used to register the listener and for services in connected regions that are allowed by the corresponding filters.

ServiceEventHook

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

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

9 Z Notation

Numbers:

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

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`
$\mathrm{dom} \ldots$	`Domain`	$\ldots \lhd \ldots$	`Domain restriction`
$\mathrm{ran} \ldots$	`Range`	$\ldots \lhd \ldots$	`Domain subtraction`

Axiomatic descriptions:

Declarations

Predicates

Schema definitions:

SchemaName

Declaration

Predicates