

Draft

Regions in Virgo (v0.3)

Glyn Normington
Steve Powell

January 12, 2011

The aim is to model how Virgo divides the OSGi framework into a connected graph of *regions*.

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

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.

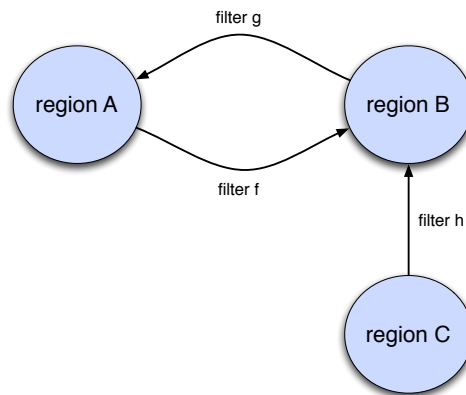


Figure 1: Connected Regions

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.

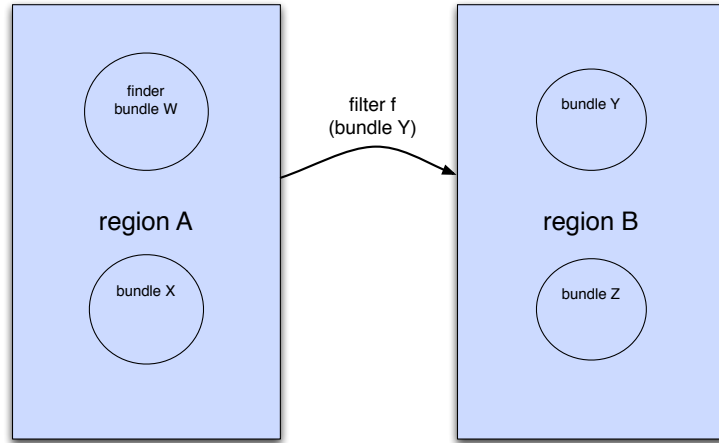


Figure 2: Bundle Find

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.

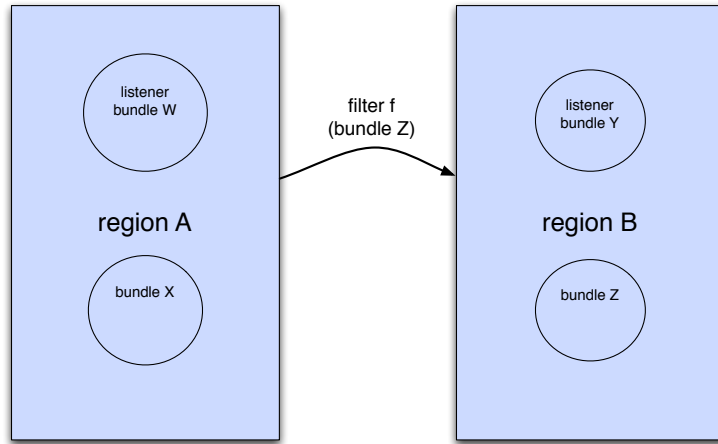


Figure 3: Bundle Event

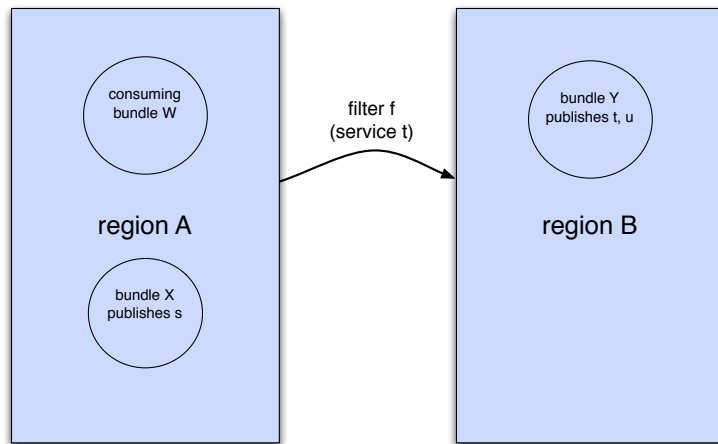


Figure 4: Service Find

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

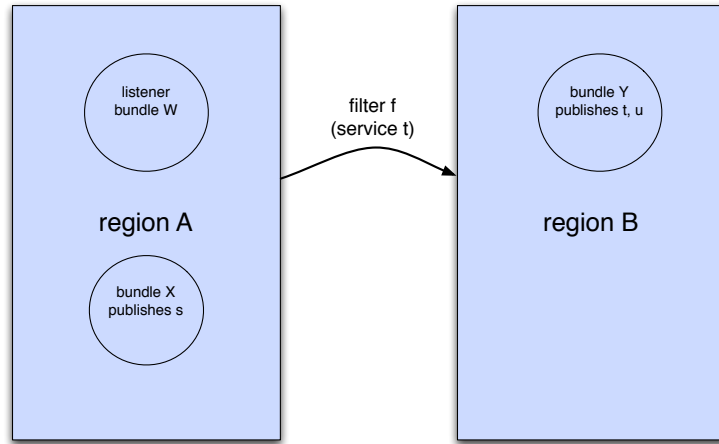


Figure 5: Service Event

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.

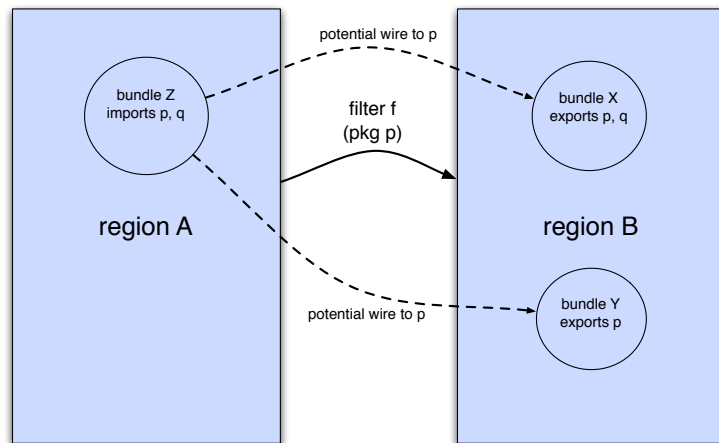


Figure 6: Package Filtering

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.

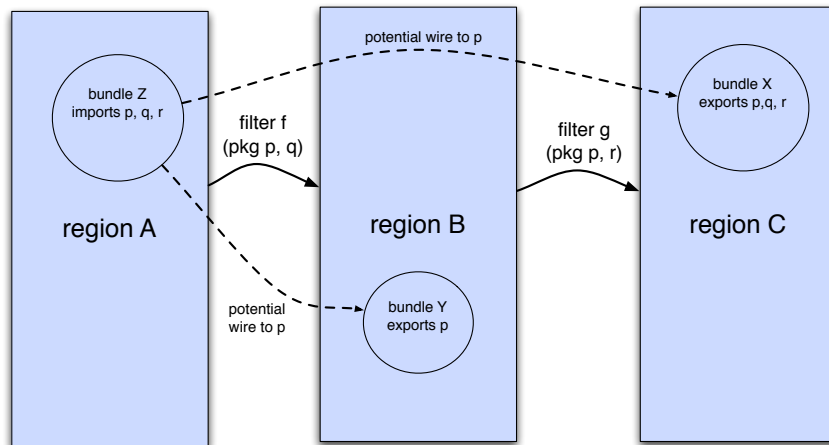


Figure 7: Transitive Package Filtering

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 identifier.

$$[RId]$$

3 Bundle

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

| | |
|--|--|
| <i>Bundle</i> | |
| <i>name</i> : <i>BSN</i> | |
| <i>version</i> : <i>BVer</i> | |
| <i>location</i> : <i>BLoc</i> | |
| <i>exportedPackages</i> : \mathbb{P} <i>Package</i> | |
| <i>publishedServices</i> : \mathbb{P} <i>Service</i> | |

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.

| | |
|--|--|
| $bid : Bundle \rightarrow (BSN \times BVer)$ | |
| $bid = (\lambda Bundle \bullet (name, version))$ | |

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.

| |
|---|
| $\text{ProtoRegion} \text{ ---}$ $\text{bundles} : \mathbb{P} \text{ Bundle}$ |
|---|

4.2 Indexed Region

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

An indexed region contains a set of bundles each of which is uniquely identified *within the region* by its bundle symbolic name and bundle version and by its location.

The indices are a function identifying the bundles in the region by symbolic name and version and a function identifying the bundles in the region by location.

| |
|--|
| $\text{IndexedRegion} \text{ ---}$ ProtoRegion $nv : \text{BSN} \times \text{BVer} \rightarrow \text{Bundle}$ $l : \text{BLoc} \rightarrow \text{Bundle}$ |
| $nv = \{b : \text{bundles} \bullet (bid\ b) \mapsto b\}$ $l = \{b : \text{bundles} \bullet b.location \mapsto b\}$ |

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.

| |
|--|
| $\text{OpenRegion} \text{ ---}$ IndexedRegion $\text{localPkg} : \mathbb{P} \text{ Package}$ $\text{localSvc} : \mathbb{P} \text{ Service}$ |
| $\text{localPkg} = \bigcup \{b : \text{bundles} \bullet b.exportedPackages\}$ $\text{localSvc} = \bigcup \{b : \text{bundles} \bullet b.publishedServices\}$ |

4.4 LinkedRegion

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

| |
|--|
| $LinkedRegion$ $OpenRegion$ $importedPkg : \mathbb{P} Package$ $importedSvc : \mathbb{P} Service$ |
|--|

4.5 Region

A region is a linked region with derived sets of all packages and all services.

| |
|--|
| $Region$ $LinkedRegion$ $pkg : \mathbb{P} Package$ $svc : \mathbb{P} Service$ |
| $pkg = localPkg \cup importedPkg$ $svc = localSvc \cup importedSvc$ |

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

| |
|---|
| $RegionAddBundleOk$ $\Delta Region$ $b? : Bundle$ |
| $(bid\ b?) \notin \text{dom } nv$ $b?.location \notin \text{dom } l$ $bundles' = bundles \cup \{b?\}$ $importedPkg' = importedPkg$ $importedSvc' = importedSvc$ |

We define a nil region.

| |
|---|
| $NIL : Region$ |
| $NIL.bundles = \emptyset$ $NIL.importedPkg = \emptyset$ $NIL.importedSvc = \emptyset$ |

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

| |
|---|
| $_ \sqcup _ : Region \times Region \leftrightarrow Region$ |
| $(_ \sqcup _) = (\lambda r1, r2 : Region \bullet$ $\quad (\mu Region \mid$ $\quad \quad bundles = r1.bundles \cup r2.bundles \wedge$ $\quad \quad pkg = r1.pkg \cup r2.pkg \wedge$ $\quad \quad svc = r1.svc \cup r2.svc))$ |

The precondition of \sqcup is that the input regions have consistent bundle spaces, that is no bundle in one region has the same symbolic name and version as a bundle in the other region. Also, no bundle in one region has the same location as a bundle in the other region.

\sqcup is idempotent, commutative, and associative and NIL acts as a zero.

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

We define the set of all pairwise consistent regions

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

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 and associative, so we define a distributed form.

$$\left| \begin{array}{l} \sqcup : \mathbb{F} Region \leftrightarrow Region \\ \hline \sqcup \emptyset = NIL \wedge \\ (\forall r : Region; f : ConsistentRegionSets \mid \{r\} \cup f \in ConsistentRegionSets \bullet \\ \quad \sqcup(\{r\} \cup f) = r \sqcup \sqcup f) \end{array} \right.$$

5 Multiple Regions

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

| | |
|--|--|
| <i>Regions</i> | |
| $reg : RId \mapsto Region$ | |
| $allBundles : \mathbb{P} Bundle$ | |
| $rids : \mathbb{P} RId$ | |
| $breg : Bundle \mapsto RId$ | |
| $allBundles = \bigcup \{r : \text{ran } reg \bullet r.bundles\}$ | |
| $rids = \text{dom } reg$ | |
| $breg = \{b : Bundle; rid : rids \mid b \in (reg \text{ } rid).bundles \bullet (b, rid)\}$ | |

Since $breg$ is a function, no bundle can belong to more than one region.

We expose the convenience function as an operation for determining a bundle's region. This function is well defined thanks to the preceding theorem.

| | |
|------------------------------|--|
| <i>GetRegionOk</i> | |
| $\Xi Regions$ | |
| $b? : Bundle$ | |
| $r! : Region$ | |
| $b? \in allBundles$ | |
| $r! = reg(breg \text{ } b?)$ | |

We define a promotion schema.

| | |
|---|--|
| <i>PromoteRegion</i> | |
| $\Delta Regions$ | |
| $\Delta Region$ | |
| $rid? : RId$ | |
| $rid? \in \text{dom } reg$ | |
| $\theta Region = reg \text{ } rid?$ | |
| $reg' = reg \oplus \{rid? \mapsto \theta Region'\}$ | |

Then we promote the operation to add a bundle.

$$RegionsAddBundleOk \hat{=} (\exists \Delta Region \bullet PromoteRegion \wedge RegionAddBundleOk)$$

6 Filters

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

| | |
|------------------------------------|--|
| <i>Filter</i> | |
| $bf : \mathbb{P}(BSN \times BVer)$ | |
| $pf : \mathbb{P} Package$ | |
| $sf : \mathbb{P} Service$ | |

We define some helper functions to perform filtering.

| | |
|---|--|
| $xb : Bundle \times Filter \rightarrow Bundle$ | |
| $xb = (\lambda b : Bundle; f : Filter \bullet$ $(\mu Bundle \mid$ $name = b.name \wedge$ $version = b.version \wedge$ $location = b.location \wedge$ $exportedPackages = b.exportedPackages \cap f.pf \wedge$ $publishedServices = b.publishedServices \cap f.sf))$ | |
| $fb : \mathbb{P} Bundle \times Filter \rightarrow \mathbb{P} Bundle$ | |
| $fb = (\lambda bs : \mathbb{P} Bundle; f : Filter \bullet \{b : bs \mid (bid\ b) \in f.bf \bullet xb(b, f)\})$ | |
| $fp : \mathbb{P} Package \times Filter \rightarrow \mathbb{P} Package$ | |
| $fp = \{ps : \mathbb{P} Package; f : Filter \bullet ((ps, f), ps \cap f.pf)\}$ | |
| $fs : \mathbb{P} Service \times Filter \rightarrow \mathbb{P} Service$ | |
| $fs = \{ss : \mathbb{P} Service; f : Filter \bullet ((ss, f), ss \cap f.sf)\}$ | |

We also define the most permissive filter.

| | |
|----------------------------|--|
| $TOP : Filter$ | |
| $TOP.bf = BSN \times BVer$ | |
| $TOP.pf = Package$ | |
| $TOP.sf = Service$ | |

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

| | |
|---|--|
| $-\wr - : Region \times Filter \rightarrow Region$ | |
| $(-\wr -) = (\lambda r : Region; f : Filter \bullet$ $(\mu Region \mid$ $bundles = \{b : r.bundles \mid (bid\ b) \in f.bf \bullet xb(b, f)\} \wedge$ $nv = f.bf \triangleleft r.nv \wedge$ $l = nv \circ bid \circ r.l \wedge$ $pkg = r.pkg \cap f.pf \wedge$ $svc = r.svc \cap f.sf))$ | |

\wr 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.

$$\vdash \forall r : Region \bullet r \wr TOP = r$$

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

$$\vdash \forall r : Region; f, g : Filter \bullet (r \wr f) \wr g = (r \wr g) \wr f$$

7 Connected Regions

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.

| |
|--|
| <i>ConnectedRegions</i> |
| <i>Regions</i> |
| $filter : RId \times RId \leftrightarrow Filter$ |
| $squash : RId \leftrightarrow Region$ |
| $first \models \text{dom } filter = rids$ |
| $second \models \text{dom } filter = rids$ |
| $(\forall rid : rids \bullet filter(rid, rid) = TOP)$ |
| $\text{dom } squash = rids$ |
| $(\forall rid : rids \bullet$ |
| $(\text{let } r == reg\ rid \bullet$ |
| $squash(rid) = r \sqcup \bigsqcup \{r2 : RId \mid (rid, r2) \in \text{dom } filter \wedge r2 \neq rid \bullet$ |
| $squash(r2) \wr filter(rid, r2)\})$ |

This schema places a constraint on the system that a bundle in any given region, can never see two distinct bundles with identical bundle symbolic names and versions. We go further than this when adding connections and bundles and make sure that the filters never clash over a bundle symbolic name and version and that the bundles in a given region never clash with the filters associated with the connections from the given region.

We define a condition on an output system of connected regions which states that for a given input region identifier, any filters connecting the corresponding region to distinct regions do not overlap on any bundle symbolic name and bundle version.

| |
|--|
| <i>FiltersConsistent</i> |
| <i>ConnectedRegions'</i> |
| $r? : RId$ |
| $(\forall s, t : RId; f, g : Filter \mid (r?, s) \mapsto f \in filter' \wedge (r?, t) \mapsto g \in filter' \bullet$ |
| $f.bf \cap g.bf = \emptyset$) |

We define an operation to connect two regions with a filter which must be consistent with the other filters associated with the first of the regions.

| |
|---|
| <i>ConnectOk</i> |
| $\Delta ConnectedRegions$ |
| $r?, s? : RId$ |
| $f? : Filter$ |
| <i>FiltersConsistent</i> |
| $(r?, s?) \notin \text{dom } filter$ |
| $\theta Regions' = \theta Regions$ |
| $filter' = filter \cup \{(r?, s?) \mapsto f?\}$ |

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.

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

We also promote the get region operation for completeness.

$$CRGetRegionOk \triangleq \Xi ConnectedRegions \wedge GetRegionOk$$

8 Hooks

We now describe the behaviour of the hooks in terms of the system of connected regions.

| |
|--|
| <i>BundleFindHook</i> $\exists \text{ConnectedRegions}$ $\text{finder?} : \text{Bundle}$ $\text{candidates?} : \mathbb{P} \text{Bundle}$ $\text{found!} : \mathbb{P} \text{Bundle}$ |
| $\text{finder?} \in \text{allBundles}$ $\text{found!} = \text{candidates?} \cap (\text{squash}(\text{breg finder?})).\text{bundles}$ |
| <i>BundleEventHook</i> $\exists \text{ConnectedRegions}$ $\text{listeners?} : \mathbb{P} \text{Bundle}$ $\text{eb?} : \text{Bundle}$ $\text{fl!} : \mathbb{P} \text{Bundle}$ |
| $\text{listeners?} \subseteq \text{allBundles}$ $\text{fl!} = \{l : \text{listeners?} \mid \text{eb?} \in (\text{squash}(\text{breg } l)).\text{bundles}\}$ |
| <i>ResolveHookFilterMatches</i> $\exists \text{ConnectedRegions}$ $\text{requirer?} : \text{Bundle}$ $\text{candidates?} : \mathbb{P} \text{Package}$ $\text{filtered!} : \mathbb{P} \text{Package}$ |
| $\text{requirer?} \in \text{allBundles}$ $\text{filtered!} = \text{candidates?} \cap (\text{squash}(\text{breg requirer?})).\text{pkg}$ |
| <i>ServiceFindHook</i> $\exists \text{ConnectedRegions}$ $\text{finder?} : \text{Bundle}$ $\text{candidates?} : \mathbb{P} \text{Service}$ $\text{found!} : \mathbb{P} \text{Service}$ |
| $\text{finder?} \in \text{allBundles}$ $\text{found!} = \text{candidates?} \cap (\text{squash}(\text{breg finder?})).\text{svc}$ |
| <i>ServiceEventHook</i> $\exists \text{ConnectedRegions}$ $\text{listeners?} : \mathbb{P} \text{Bundle}$ $\text{es?} : \text{Service}$ $\text{fl!} : \mathbb{P} \text{Bundle}$ |
| $\text{listeners?} \subseteq \text{allBundles}$ $\text{fl!} = \{l : \text{listeners?} \mid \text{es?} \in (\text{squash}(\text{breg } l)).\text{svc}\}$ |

9 Z Notation

Numbers:

\mathbb{N} Natural numbers $\{0, 1, \dots\}$

Propositional logic and the schema calculus:

| | | | |
|------------------------------|----------------------|---------------------------------------|------------------------|
| $\dots \wedge \dots$ | And | $\langle\langle \dots \rangle\rangle$ | Free type injection |
| $\dots \vee \dots$ | Or | $[\dots]$ | Given sets |
| $\dots \Rightarrow \dots$ | Implies | $', ?, !, 0 \dots 9$ | Schema decorations |
| $\forall \bullet ..$ | For all | $\dots \vdash \dots$ | theorem |
| $\exists \bullet ..$ | There exists | $\theta \dots$ | Binding formation |
| $\dots \backslash \dots$ | Hiding | $\lambda \dots$ | Function definition |
| $\dots \triangleq \dots$ | Schema definition | $\mu \dots$ | Mu-expression |
| $\dots == \dots$ | Abbreviation | $\Delta \dots$ | State change |
| $\dots ::= \dots \dots$ | Free type definition | $\Xi \dots$ | Invariant state change |

Sets and sequences:

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

Functions and relations:

| | | | |
|----------------------------------|-------------------|-------------------------------|------------------------------|
| $\dots \leftrightarrow \dots$ | Relation | $\dots \mapsto \dots$ | maplet |
| $\dots \rightarrow \dots$ | Partial function | $\dots \sim$ | Relational inverse |
| $\dots \twoheadrightarrow \dots$ | Total function | \dots^* | Reflexive-transitive closure |
| $\dots \rightsquigarrow \dots$ | Partial injection | $\dots \langle \dots \rangle$ | Relational image |
| $\dots \hookrightarrow \dots$ | Injection | $\dots \oplus \dots$ | Functional overriding |
| dom... | Domain | $\dots \triangleleft \dots$ | Domain restriction |
| ran... | Range | $\dots \trianglelefteq \dots$ | Domain subtraction |

Axiomatic descriptions:

| |
|---------------------|
| <i>Declarations</i> |
| <i>Predicates</i> |

Schema definitions:

| |
|--------------------|
| <i>SchemaName</i> |
| <i>Declaration</i> |
| <i>Predicates</i> |