Eclipse Virgo
A Technical Overview

White Paper

Version 1.0
March 2012

Contents

Introduction....3
History....3
What problems does Virgo solve?....4
Benefits....6
- Advanced tooling....8
- Future proof....9

Technology....9
- Artefacts....9
- Kernel....9
- Regions....9
- Scoping....10
- Synthetic Context Generation....11
- Example of Scoping....12
- Deployment Pipeline....12
- Transformers....13
- Quasi Framework....13
- Exception Handling....14
- Repositories....14

Virgo Runtime Deliverables....15
Virgo IDE Tooling....16
Virgo Bundlor....16
Embedded Technology....16
- Tomcat....16
- Jetty....16
- Gemini Web....16
- Gemini Blueprint....17
- Gemini Management....17
- Equinox OSGi Framework....17
- Equinox p2....17
- Equinox Region Bundle....17

Standards....17
- Servlet Specification....17
- Java EE Web Profile....17
- OSGi Core....17
- OSGi Web Applications....17
- OSGi Blueprint....17
- OSGi Declarative Services....17
- OSGi Log Service....17
- OSGi Event Admin Service....18
- OSGi Configuration Admin Service....18
- OSGi Subsystems....18

Further Information....18

Introduction

The Eclipse Virgo project provides modular runtimes and application servers and is part of the EclipseRT umbrella project.

Unlike traditional Java runtimes, Virgo was designed from the beginning as a completely modular collection of OSGi bundles running on a standard OSGi framework implementation (Equinox). Virgo was also one of the first runtimes to provide support for applications written as well-defined collections of OSGi bundles.

As well as providing web servers based on popular servlet containers, Virgo has a separate kernel which provides many of Virgo's technical innovations and recently a lightweight "nano kernel" has been factored out of the kernel and is being used to build a lightweight web server which will support the Java EE Web Profile. The nano-based deliverables of Virgo 3.5 should appear in the Eclipse Juno 2012 release train.

So Virgo is a prime example of the "stackless stack".[1] Users have built their own servers on top of the Virgo kernel while other users are happy to use the servers provided by Virgo.

After recapping the project's history, this paper explains the problems that Virgo set out to solve, the benefits that Virgo provides, and the core technology inside Virgo. It then summarises the runtime deliverables provided by Virgo and the Eclipse-based and stand-alone application development tooling provided by Virgo. The paper concludes with brief surveys of the component technologies embedded in Virgo, the standards supported by Virgo, and some source of further information.

History

The Virgo project started life at the end of 2007 as the SpringSource Application Platform. Later it was renamed to the SpringSource dm Server that many people know today. Virgo shares the basic goals of the first version of dm Server:

- Better OSGi platform
- Migration of Java EE applications
- Modular and extensible

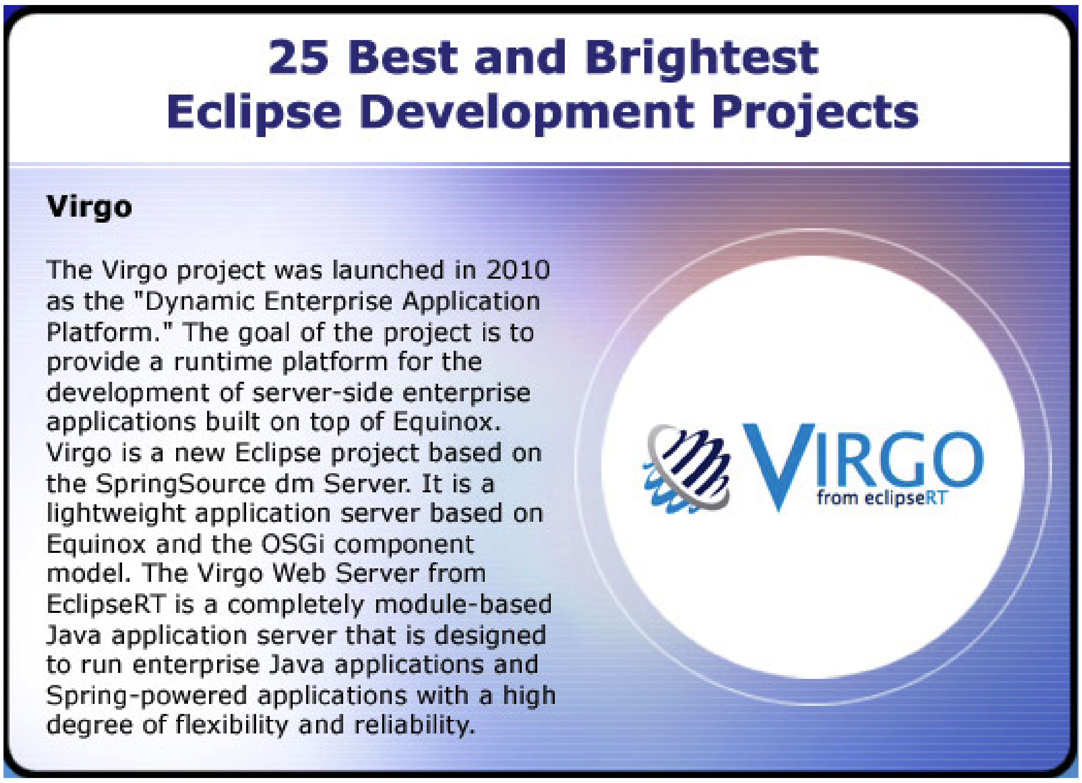
A year later, at the end of 2008, version 1.0 was released to much fanfare. Over the following year more and more people tried it out and a real community of developers grew up around it. Based on feedback from users and the desire to provide a separate kernel, version 2.0 was released at the end of 2009.

By the end of 2009 a lot of the ground work had already been laid for the donation to Eclipse and in January 2010 the announcement was made. The dm Server name went out with a bang by winning the Eclipse award for best RT application.

1 A term coined by RedMonk's James Governor http://www.redmonk.com/jgovernor/2008/02/05/osgi-and-the-rise-of-the-stackless-stack-just-in-time/

eclipse
Eclipse Community Awards 2010 Winner

In June 2010, Virgo was listed in eWeek"s 25 Best and Brightest Eclipse Development Projects.



25 Best and Brightest Eclipse Development Projects

Virgo

The Virgo project was launched in 2010 as the "Dynamic Enterprise Application Platform." The goal of the project is to provide a runtime platform for the development of server-side enterprise applications built on top of Equinox. Virgo is a new Eclipse project based on the SpringSource dm Server. It is a lightweight application server based on Equinox and the OSGi component model. The Virgo Web Server from EclipseRT is a completely module-based Java application server that is designed to run enterprise Java applications and Spring-powered applications with a high degree of flexibility and reliability.

Since then Virgo has shipped 2.1 and 3.0 releases. 3.0 included Jetty support and the snaps framework for modular web applications. At the time of writing in March 2012 3.5 is in development and includes p2 provisioning support and a new pair of deliverables: nano and nano full.

What problems does Virgo solve?

Virgo features:

•admin console - deploy and manage Artefacts, examine diagnostic dumps, and examine bundle wiring, either in the live system or from a resolution failure

•Equinox console extension - manage Virgo and deployed Artefacts

•plans - define the Artefacts that comprise an application, optionally making the application atomic to tie the Artefact lifecycles together, and scoped to isolate the application from other applications

•provisioning - automatically supply the dependencies of an application including bundles, plans, plan archives (PARs), and configurations, from both local and remote repositories

•web container - supports vanilla WAR files, with all their dependencies in `WEB-INF/lib`, and Web Application Bundles, that import their dependencies via OSGi manifest metadata, via the embedded Tomcat-based reference implementation of the OSGi Web Container specification, configured using the standard Tomcat `server.xml`

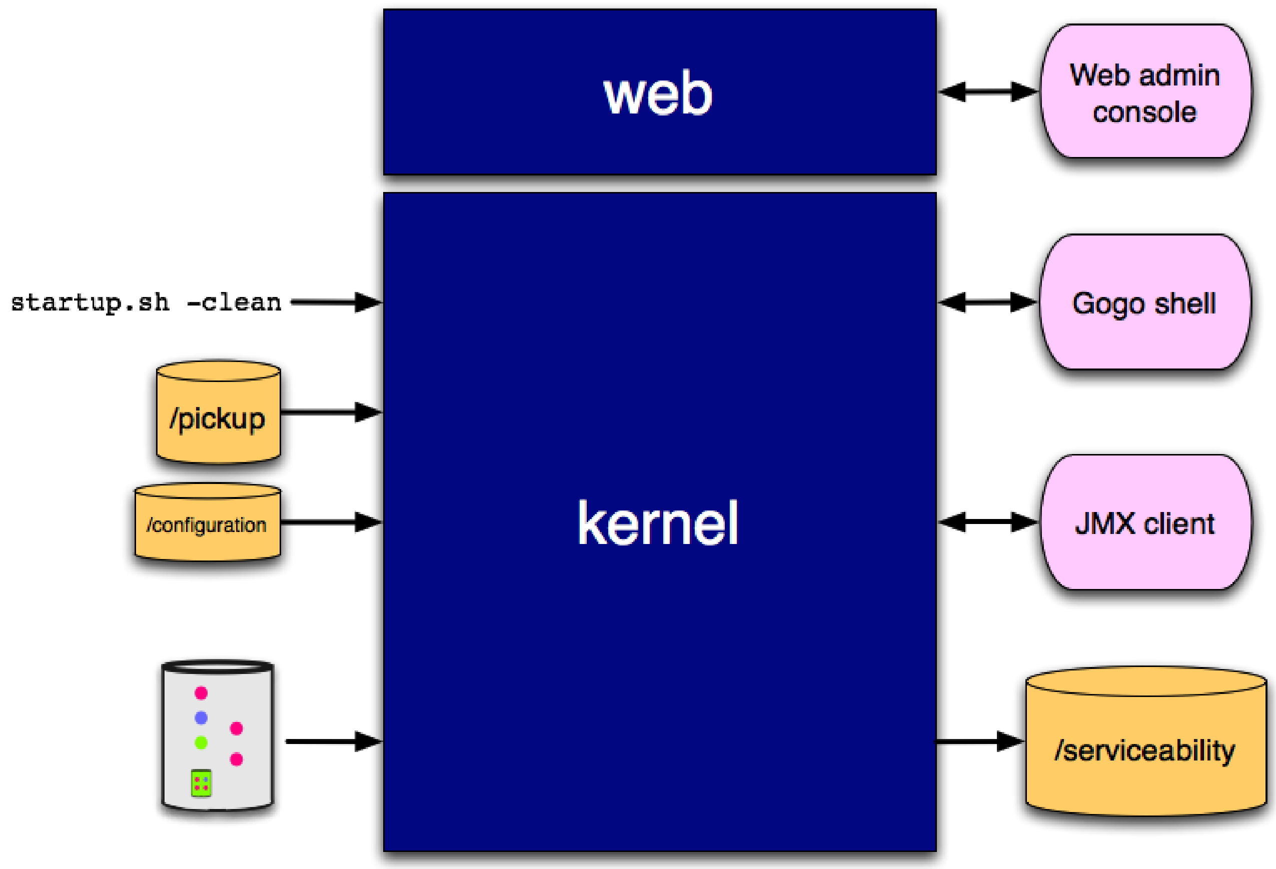
•user region - isolates the kernel from user-installed applications and enables admins to focus on application Artefacts and their dependencies without seeing those of the kernel as well

•Spring 3 - Virgo packages Spring 3.0 final, but can easily be configured to use a different version of Spring

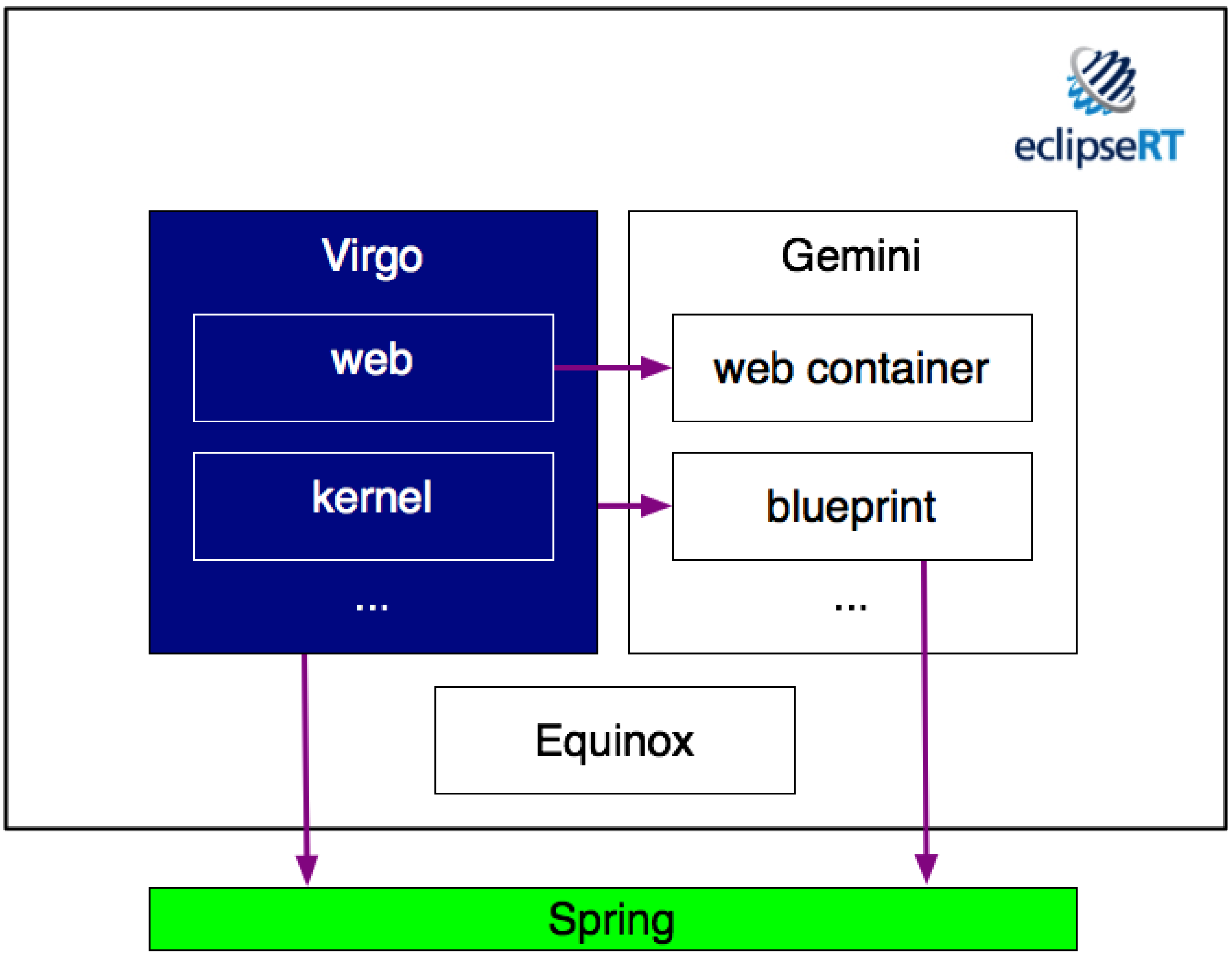
•hot deployment - deploy Artefacts to Virgo by copying them into the `pickup` directory, either in archive or exploded form, as an alternative to deploying via the admin console

•logging - via embedded LogBack, configured in `config/serviceability.xml`, with a rich set of appenders available out of the box

The following diagram shows the key externals of Virgo:



while the diagram below shows the high-level internal structure of Virgo:



Benefits

Virgo is an ideal OSGi server runtime, possibly the ideal OSGi server runtime, from several perspectives.

Standard OSGi

Virgo supports all the standard features of OSGi; encapsulated modules or bundles, a powerful service registry, versioning, class space isolation, sharing of dependencies, dynamic refreshing and updating, and much more.

Bundles all the way down

Virgo was designed from the ground up as OSGi bundles so it's extremely modular and extensible.

For example, the core of the runtime is the Virgo kernel and the Virgo web server is constructed by configuring a web layer, an admin console, and some other utilities on top of the kernel. The kernel protects itself from interference from such additions, and applications, by running them in a nested framework. This nested framework approach also enables applications to use new versions of the Spring framework (regardless of the fact the kernel depends on Spring 3.0.0).

All the major Java EE application servers are now built on top of OSGi, but only Virgo was designedfor OSGi and didn't need OSGi to be retrofitted.

Extra features

Virgo provides a multi-bundle application model to simplify the deployment and management of non-trivial applications.

Virgo also provides a repository which can store dependencies such as OSGi bundles which are "faulted in" when needed. This results in cleaner definitions of applications and a small footprint compared to traditional Java EE servers which pre-load many features just in case an application needs them.

The major Java EE application servers are beginning to follow suit in exposing an application model, so Virgo committers are working with others in the OSGi Alliance to produce a standard multi-bundle application construct.

Existing Java libraries

Virgo enables existing Java libraries to run successfully in an OSGi environment. Ok, you have to convert them to bundles first, but then the Virgo kernel supports thread context class loading, load time weaving, classpath scanning, and a number of other features which are commonly used by persistence providers and other common Java utilities.

Essentially, we observed the commonly-occurring problems when people attempted to migrate to OSGi and implemented general solutions to those problems early on.

Container integration

Virgo integrates OSGi with Spring and a servlet container.

Virgo uses Spring DM to wire application contexts to the OSGi service registry. Spring beans can be published as OSGi services and can consume OSGi services, both with minimal effort.

The embedded form of Tomcat is used as a servlet engine in Virgo's web support and is configured and managed just like standard Tomcat. Jetty support is also on the roadmap for the future.

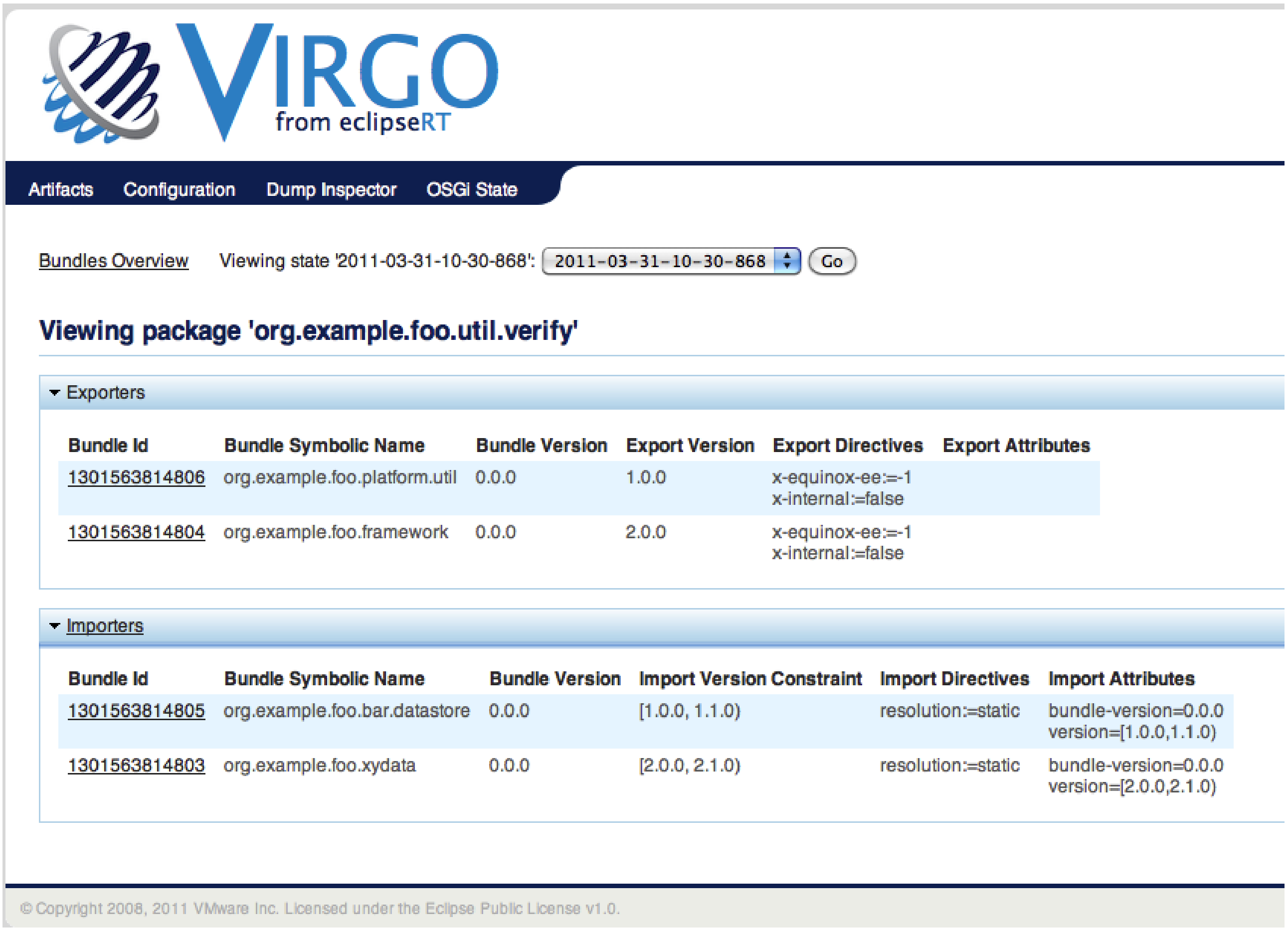
Diagnostics++

Virgo has extensive diagnostics:

- An admin console lets you examine the state of all Artefacts deployed in Virgo as well as explore the state of bundles in the OSGi framework.
- Virgo provides multiple types of diagnostics: event logging aimed at administrators, diagnostics logging aimed at developers, as well as various types of diagnostic dumps.
- Virgo builds on Logback to support highly configurable and efficient logging.
- When an application is deployed, Virgo first resolves the application in a "side state" and, if resolution fails, no changes are committed to the OSGi framework and the side state is dumped to disk so that it can be analysed using the OSGi state inspector in the admin console.
- If a resolution failure occurs, Virgo also analyses the resolver error, including the root causes of any uses constraint violation, and extracts a meaningful message to include in an exception which is then thrown.
- Virgo adds advanced diagnostics to Spring DM to track the starting of bundles and their application contexts and issue error messages on failure and warnings when dependencies are

delayed.

•Virgo automatically detects deadlocks and generates a thread stack dump.



Bundles Overview   Viewing state '2011-03-31-10-30-868': 2011-03-31-10-30-868

Viewing package 'org.example.foo.util.verify'

Exporters

Bundle Id	Bundle Symbolic Name	Bundle Version	Export Version	Export Directives	Export Attributes
1301563814806	org.example.foo.platform.util	0.0.0	1.0.0	x-equinox-ee:=-1 x-internal:=false	
1301563814804	org.example.foo.framework	0.0.0	2.0.0	x-equinox-ee:=-1 x-internal:=false	

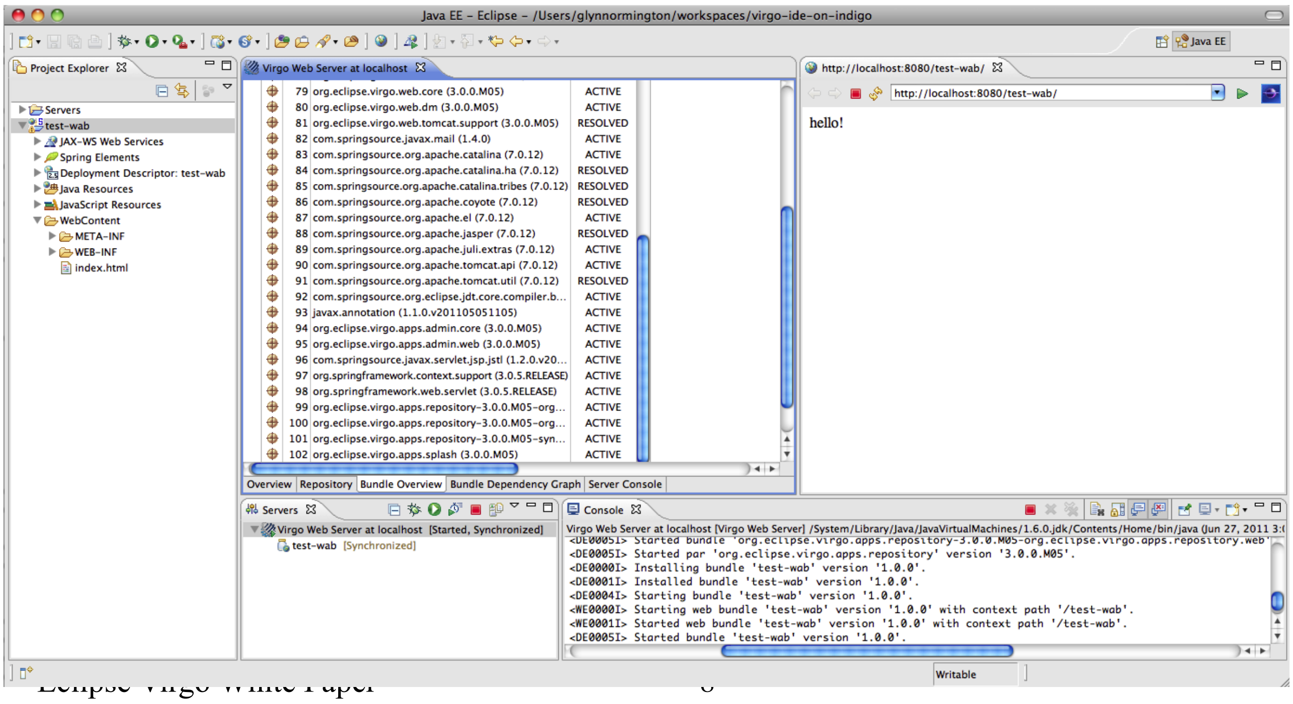
Importers

Bundle Id	Bundle Symbolic Name	Bundle Version	Import Version Constraint	Import Directives	Import Attributes
1301563814805	org.example.foo.bar.datastore	0.0.0	[1.0.0, 1.1.0)	resolution:=static	bundle-version=0.0.0 version=[1.0.0,1.1.0)
1301563814803	org.example.foo.xydata	0.0.0	[2.0.0, 2.1.0)	resolution:=static	bundle-version=0.0.0 version=[2.0.0,2.1.0)

© Copyright 2008, 2011 VMware Inc. Licensed under the Eclipse Public License v1.0.

Advanced tooling

Virgo tooling support is provided in standard Eclipse via the Virgo IDE update site. This enables a Virgo server to run under the control of the tooling, applications to be deployed, debugged, and updated by the tooling, and package and service dependencies between bundles to be analysed.



```
 79 org.eclipse.virgo.web.core (3.0.0.M05)                  ACTIVE
 80 org.eclipse.virgo.web.dm (3.0.0.M05)                    ACTIVE
 81 org.eclipse.virgo.web.tomcat.support (3.0.0.M05)       RESOLVED
 82 com.springsource.javax.mail (1.4.0)                     ACTIVE
 83 com.springsource.org.apache.catalina (7.0.12)           ACTIVE
 84 com.springsource.org.apache.catalina.ha (7.0.12)       RESOLVED
 85 com.springsource.org.apache.catalina.tribes (7.0.12)   RESOLVED
 86 com.springsource.org.apache.coyote (7.0.12)            RESOLVED
 87 com.springsource.org.apache.el (7.0.12)                 ACTIVE
 88 com.springsource.org.apache.jasper (7.0.12)            RESOLVED
 89 com.springsource.org.apache.juli.extras (7.0.12)        ACTIVE
 90 com.springsource.org.apache.tomcat.api (7.0.12)         ACTIVE
 91 com.springsource.org.apache.tomcat.util (7.0.12)       RESOLVED
 92 com.springsource.org.eclipse.jdt.core.compiler.b...     ACTIVE
 93 javax.annotation (1.1.0.v201105051105)                  ACTIVE
 94 org.eclipse.virgo.apps.admin.core (3.0.0.M05)           ACTIVE
 95 org.eclipse.virgo.apps.admin.web (3.0.0.M05)            ACTIVE
 96 com.springsource.javax.servlet.jsp.jstl (1.2.0.v20...   ACTIVE
 97 org.springframework.context.support (3.0.5.RELEASE)     ACTIVE
 98 org.springframework.web.servlet (3.0.5.RELEASE)         ACTIVE
 99 org.eclipse.virgo.apps.repository-3.0.0.M05-org...      ACTIVE
100 org.eclipse.virgo.apps.repository-3.0.0.M05-org...      ACTIVE
101 org.eclipse.virgo.apps.repository-3.0.0.M05-syn...      ACTIVE
102 org.eclipse.virgo.apps.splash (3.0.0.M05)               ACTIVE
```

hello!

Virgo Web Server at localhost [Started, Synchronized]
test-wab [Synchronized]

```
Virgo Web Server at localhost [Virgo Web Server] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Jun 27, 2011 3:(
<DE0005I> Started bundle 'org.eclipse.virgo.apps.repository-3.0.0.M05-org.eclipse.virgo.apps.repository.web'
<DE0005I> Started par 'org.eclipse.virgo.apps.repository' version '3.0.0.M05'.
<DE0000I> Installing bundle 'test-wab' version '1.0.0'.
<DE0001I> Installed bundle 'test-wab' version '1.0.0'.
<DE0004I> Starting bundle 'test-wab' version '1.0.0'.
<WE0000I> Starting web bundle 'test-wab' version '1.0.0' with context path '/test-wab'.
<WE0001I> Started web bundle 'test-wab' version '1.0.0' with context path '/test-wab'.
<DE0005I> Started bundle 'test-wab' version '1.0.0'.
```

Future proof

Virgo is an open source Eclipse project with a liberal license and active participation from multiple vendors, which positions it ideally for the future.

Technology

Artefacts

Virgo supports the deployment of various types of Artefact. Standard Artefact types understood by the Virgo deployer are:

1. bundle - an OSGi bundle;
2. configuration - a properties file with name of the form `<pid>.properties` which when deployed produces a Configuration Admin dictionary with PID `<pid>`;
3. plan - an XML file which refers to arbitrary other Artefacts in the Virgo repository -- plans may be scoped or unscoped, as described below, and atomic or non-atomic in terms of whether the lifecycle of the plan is tied to the lifecycle of the plan's Artefacts; and
4. PAR – a Plan ARchive which contains a collection of Artefacts and is equivalent to a scoped, atomic plan referencing those Artefacts (except the Artefacts need not be present in the Virgo repository).

In addition, it is possible to add user-defined Artefact types.

Nano

Virgo nano is a small core of function which was factored out of the original Virgo kernel. It provides most of Virgo's diagnostic features as well as p2 provisioning support. Nano can be run on its own to provide a single region framework.

Kernel

The Virgo kernel is the core runtime which may be used on its own or to deploy one or more server types and applications for those server types. The kernel houses the deployment pipeline, described below, as well as support for common Artefact types (bundle, configuration, plan, and PAR), regions, scoping, and other core Virgo features.

Currently, for example, the Virgo web server packaging build uses the kernel to deploy a web server plan which includes the Gemini web container and the Virgo web bundle (which integrates Gemini web into Virgo).

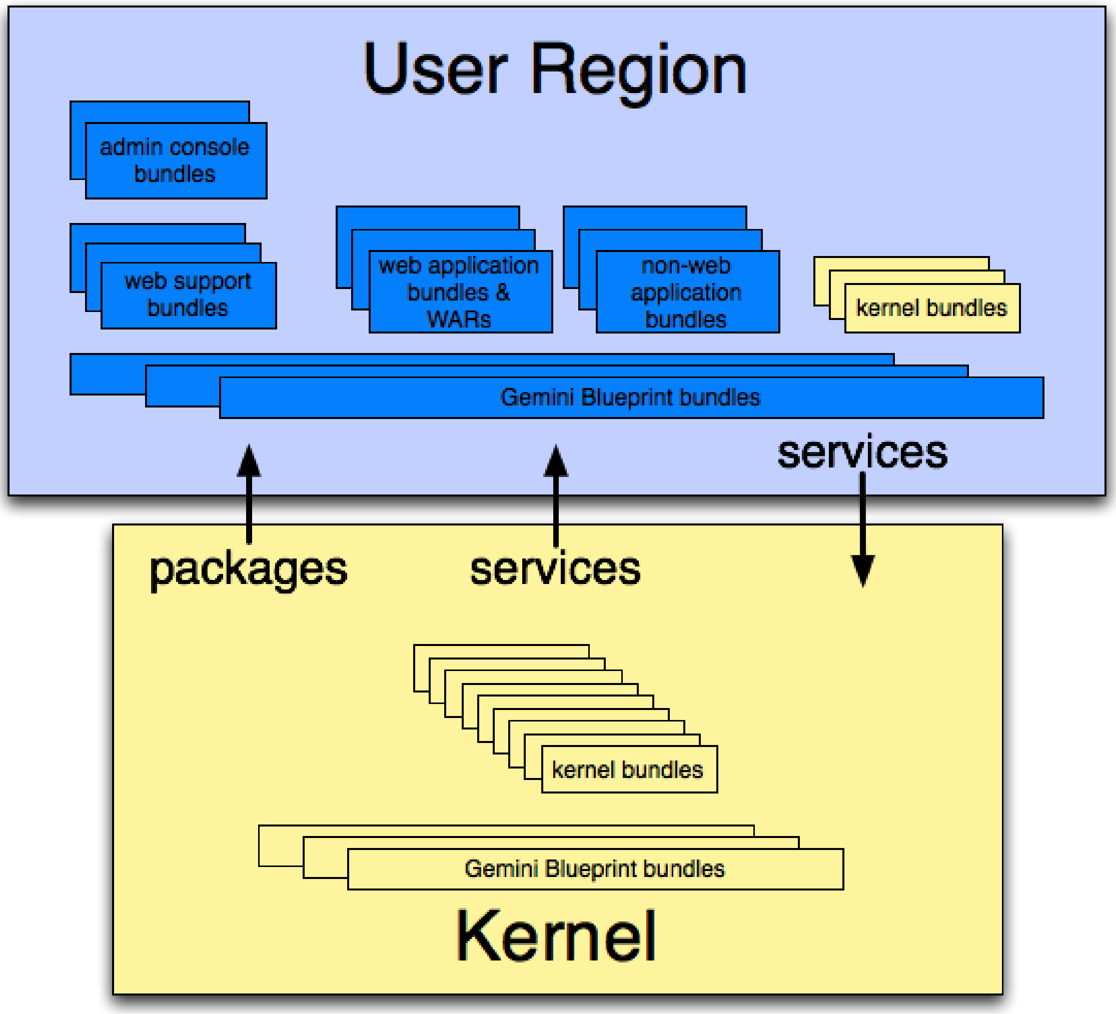
Regions

Region support was added to enable applications to run with a different version of Spring than that used by the kernel. A minimal set of Spring bundles is installed into the kernel region with very few optional dependencies which keeps the kernel footprint and startup time low. In principle, Spring could be entirely removed from the kernel region if the kernel was modified not to depend on Spring (most of these dependencies are because the kernel uses Spring DM to publish and find kernel services).

Certain packages and services are imported from the kernel region into the user region and certain services, but no packages, are exported from the user region to the kernel region. This isolates the kernel region from interference due to types and package wirings in the user region. The configuration file config/org.eclipse.virgo.kernel.userregion.properties controls the importing of packages and services into and the exporting of services out of the user region.

The content of the kernel region is controlled by the configuration file lib/org.eclipse.virgo.kernel.launch.properties.

So, apart from the basic principle that no packages are exported from the user region to the kernel



region, there is a lot of flexibility for changing the contents of both kernel and user regions and for specifying which packages and services are shared between the regions.

In future, Virgo could be extended to support multiple user regions in order to isolate applications from each other.

Scoping

Virgo adds the concept of scoping to OSGi. The main use case for scoping is where a group of bundles form an application which needs to avoid clashing with other applications and which needs reliable behaviour when it calls third party bundles which use thread context class loading. Clashes can occur because of bundles, packages, or services conflicting in some way.

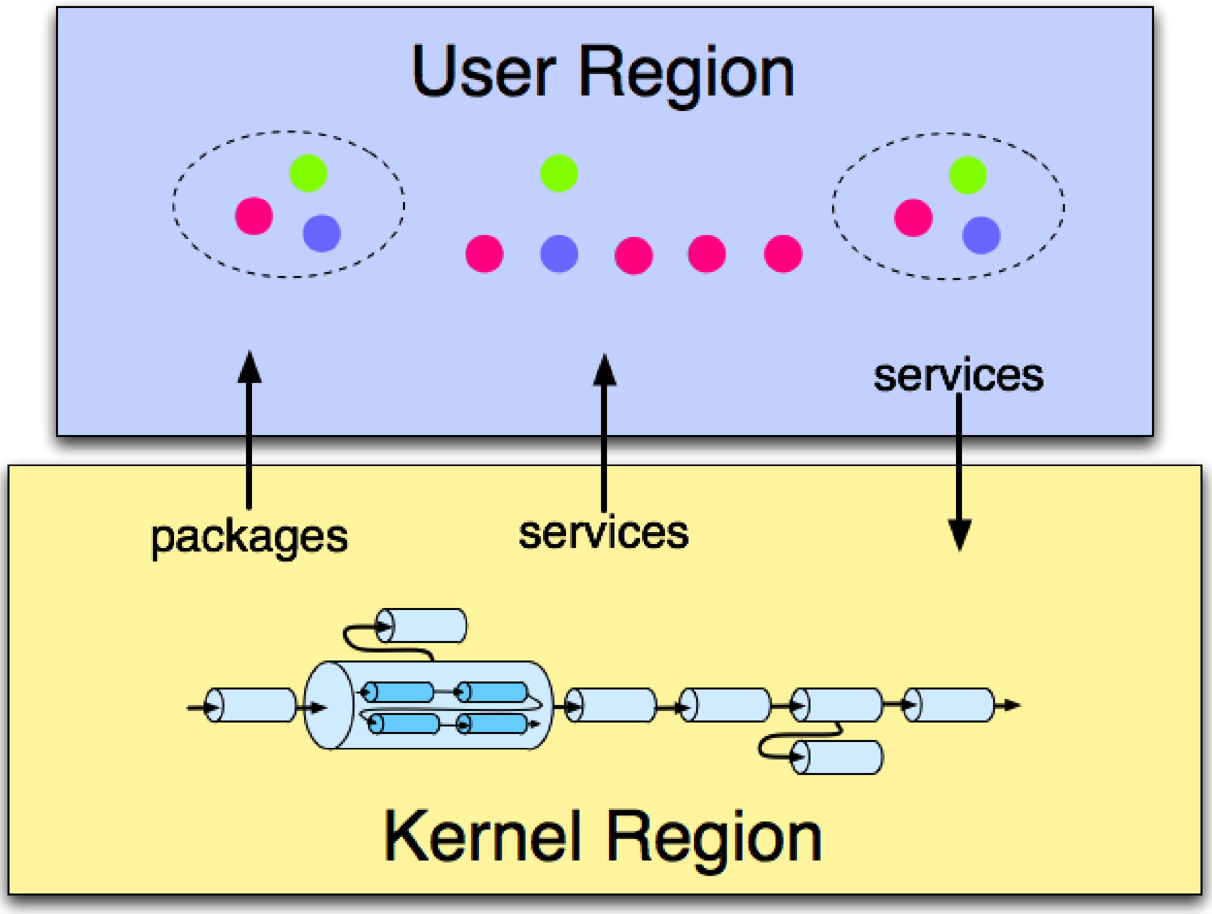
Metadata Rewriting

Virgo rewrites the metadata of bundles in a scope to prefix the bundle symbolic names with a

scope-specific prefix and to add a mandatory matching attribute, with a scope-specific value, to packages exported by bundles in the scope.
Virgo also uses the standard OSGi service registry hooks to limit the visibility of services published by bundles in a scope.
However, a bundle in a scope may access bundles, packages, and services not provided in the scope but which are available outside the scope, that is from unscoped bundles. So a scope acts similarly to a programming language scope such as Java's curly braces:

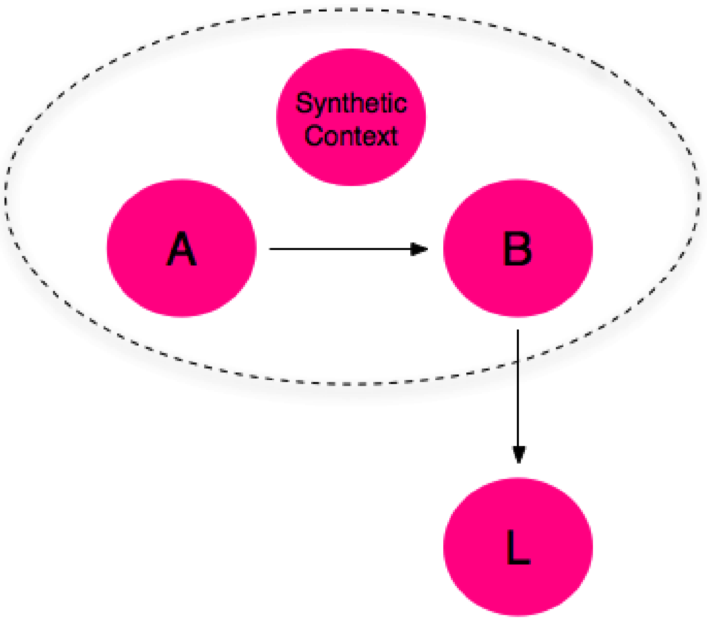
```
int x;
// b is not visible here
{
    int b;
    // both b and x are visible here
}
```

Metadata rewriting is performed using a sophisticated deployment pipeline, described later. This pipeline runs in the kernel region with limited exposure to the User Region, as shown below.



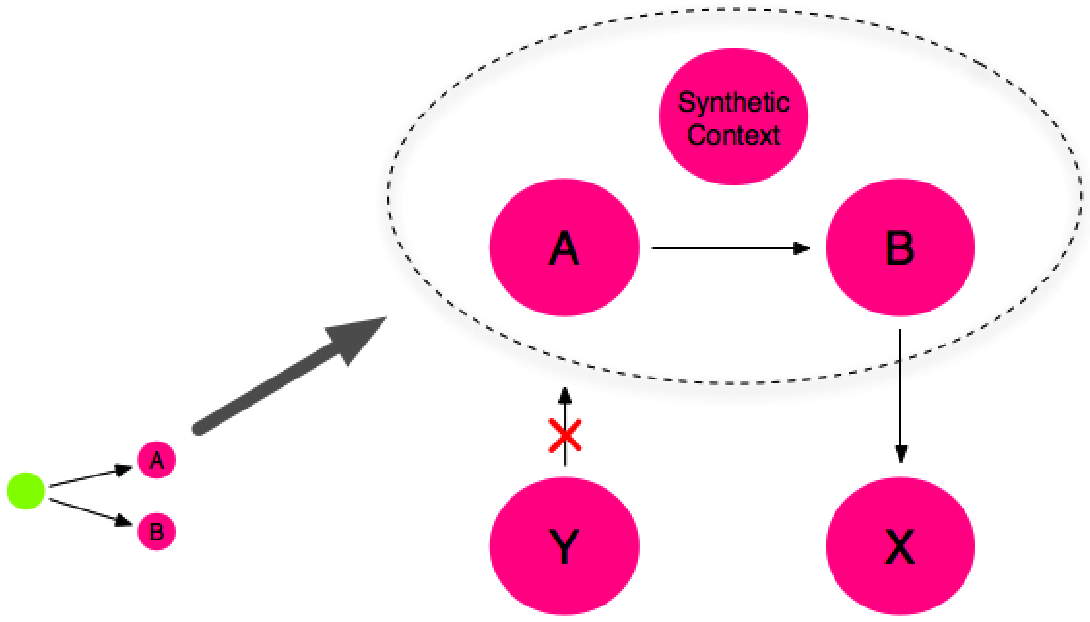
Synthetic Context Generation

To ensure reliable thread context class loading when third party bundles are called from a scope, Virgo generates a synthetic context bundle in the scope. The class loader of the synthetic context bundle is used as the thread context class loader when bundles in the scope make calls outside the scope. The synthetic bundle imports each of the other bundles in the scope using the Virgo import-bundle header. This is semantically equivalent to importing all the exported packages of the other bundles in the scope. So to make a package of a scoped application available for thread context class loading, it is simply necessary to export the package.



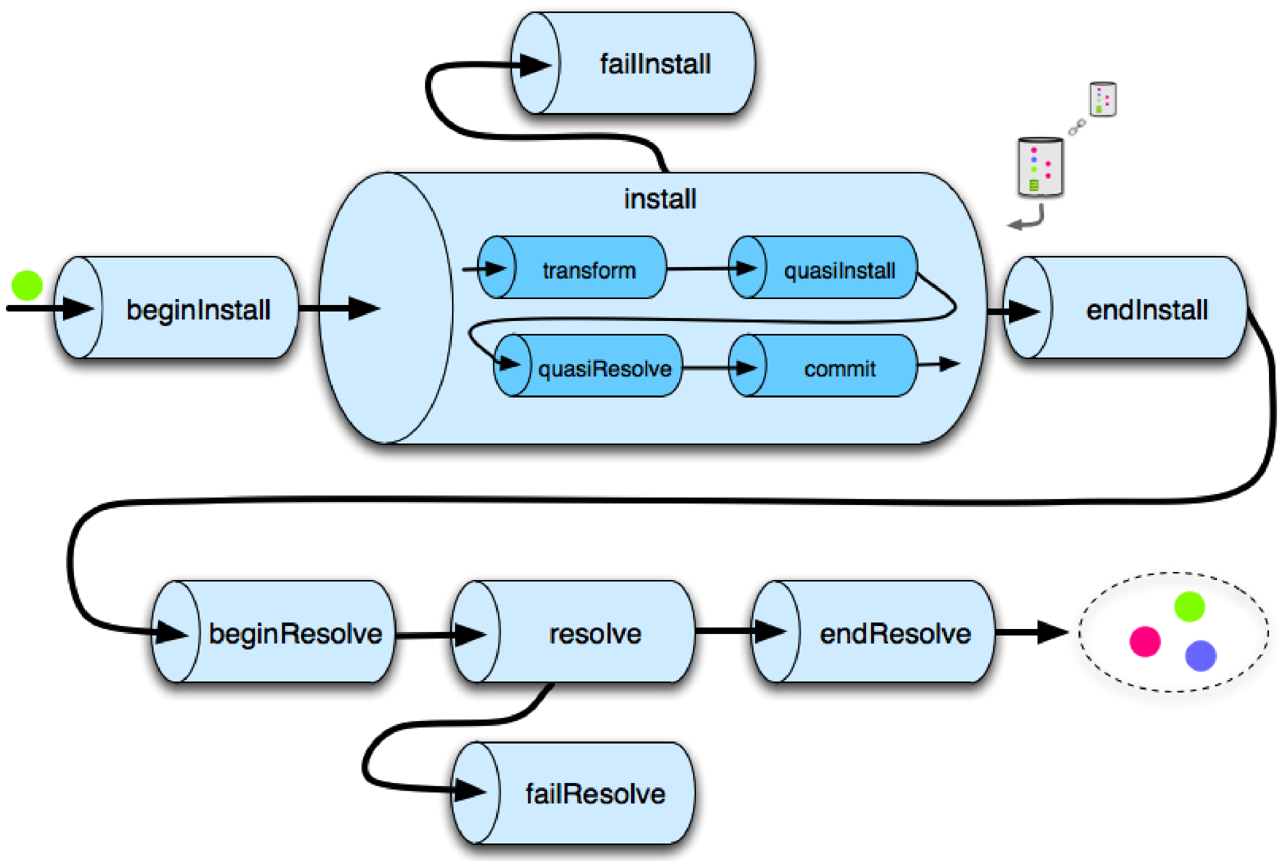
Example of Scoping

The figure below shows a scoped plan referring to two bundles A and B being deployed. The result is a scope containing the bundles A and B as well as the synthetic context bundle. Note that bundles inside the scope can access bundles, such as X, outside the scope. Also, bundles outside the scope, such as Y, cannot access bundles inside the scope.



Deployment Pipeline

Artefacts are deployed into Virgo using a deployment pipeline consisting of several pipeline stages some of which have pipelines nested inside them as shown in the figure below.



The pipeline, and each pipeline stage, accepts a tree of install Artefacts as input and outputs a possibly modified tree. The deployment pipeline is constructed by the `Plumber` class.

Transformers

Many of the interesting modifications to the tree are performed by the transform stage which uses the whiteboard pattern to drive all services of a Transformer type in order of service ranking. A number of standard Transformer services are defined in the Spring context file `deployer-context.xml` in the kernel's deployer bundle. Some interesting examples of standard Transformers are:

- `PlanResolver` which takes as input a tree consisting of a single plan node and adds a subtree representing the content of the plan, including any nested plans and their subtrees,
- `ScopingTransformer` which rewrites the metadata of a subtree rooted in a scoped plan and gathers service scoping information for the subtree,
- `SyntheticContextBundleCreatingTransformer` which adds a synthetic context bundle as a child node of a scoped plan, and
- `ImportExpandingTransformer` which converts Virgo-specific headers such as `import-bundle` into standard OSGi `import-package` statements.

Quasi Framework

The quasi framework is an abstraction of the Equinox State and is used in auto-provisioning missing dependencies during deployment. The quasiInstall stage installs the bundles in the input tree into an instance of the quasi framework. The quasiResolve stage attempts to resolve these bundles and auto-provision any missing dependencies from the Virgo repository by installing them in the quasi framework instance. The commit stage attempts to install the bundles in the input tree, along with any auto-provisioned bundles, into the OSGi framework.

Exception Handling

There are two approaches to handling exceptions thrown by a pipeline stage. In general, unexpected exceptions are allowed to percolate upward and result in diagnostics and a failed deployment. However, certain expected exceptions, such as failure to resolve the dependencies of the install Artefact tree, need to be handled more gracefully. In these cases, acompensating pipeline stage is defined which drives a compensation stage if an exception is thrown. failInstall and failResolve in the figure above are examples of compensation stages.

Repositories

Virgo repositories contain Artefact URLs and metadata and are indexed by the Cartesian product of Artefact type, name, and version. There are three kinds of repository: external, watched, and remote. Repositories are passive in the sense that changes to repository content do not cause Artefacts to be deployed into Virgo, refreshed, or undeployed. Repositories support queries which allow sets of Artefacts satisfying certain criteria, for example with a version in a given version range, to be determined.

External Repositories

External repositories are created by scanning a directory which contains Artefacts, possibly in nested directories. The repository configuration specifies a pattern which says which files should be treated as Artefacts. After the repository is created, changes to the directory do not affect the repository content.

The Virgo kernel's default repository configuration, in `config/org.eclipse.virgo.repository.properties`, specifies an external repository created from the `repository/ext` directory.

Watched Repositories

Watched repositories are created by scanning a directory which contains Artefacts but no nested directories. All files in the directory are treated as Artefacts. The directory is re-scanned periodically and the interval between re-scans is specified in the repository configuration. Changes detected by re-scanning are reflected in the repository content. Note that changing the content of a watched repository does not cause Artefacts to be deployed into Virgo, refreshed, or undeployed.

The Virgo kernel's default repository configuration specifies a watched repository based on the contents of the `repository/usr` directory.

Remote Repositories

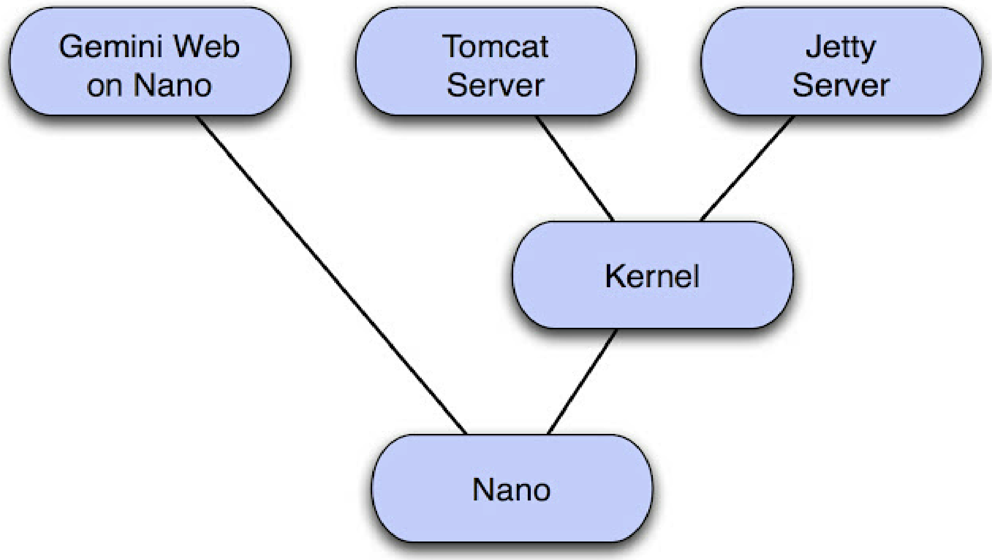
A remote repository refers to a repository hosted by a Virgo instance sometimes known as a repository server. The hosted repository is configured using the file`config/org.eclipse.virgo.apps.repository.properties` and may be either an external or a watched repository.

The remote repository is accessed by a Virgo instance sometimes known as a repository client. The repository client is normally a different instance of Virgo to the instance hosting the repository, but it can be the same instance which is handy for testing. The remote repository periodically downloads its content from the hosted repository. The period between downloads may be configured in the repository configuration. The remote repository also caches Artefacts which have secure hashes associated with them in the hosted repository. Only bundles currently have secure hashes associated with them. The secure hash is used to determine when a cached Artefact is stale and needs to be freshly downloaded.

Repository Chains

The Virgo repository is configured as a chain of external, watched, and remote repositories. The chain is a list which is searched in the configured order. The effect of this search order is that an Artefact with a given type, name, and version which appears in more than one repository in the chain is only accessed from the first repository in the chain in which it appears. Abstractly, the repository chain behaves as a single repository, but its content may mutate in quite a different way to the content of an individual external, watched, or remote repository.

Virgo Runtime Deliverables



Virgo Tomcat Server is a completely module-based Java application server that is designed to run enterprise Java applications, Spring-powered applications, and OSGi applications with a high degree of flexibility and reliability. It offers a simple yet comprehensive platform to develop, deploy, and service such applications.

Virgo Jetty Server is similar to Virgo Tomcat Server but embeds the Jetty servlet container instead of the embedded Tomcat servlet container.

Virgo kernel supports the core concepts of Virgo and is not biased towards the web server, thus enabling other types of server to be created. The kernel can also be used stand-alone as a rich OSGi application platform. A server runtime can easily be constructed by deploying suitable bundles on top of the kernel.

Virgo Nano is a minimal core of the Virgo kernel which is designed to run relatively simple OSGi applications in a single region. Bundles can be provisioned at runtime using p2.

Virgo Nano Full builds on Virgo Nano and embeds Gemini Web to provide Tomcat-based servlet support. Bundles, including Web Application Bundles, can be provisioned at runtime using p2.

Virgo IDE Tooling

Explain relationship to Libra.

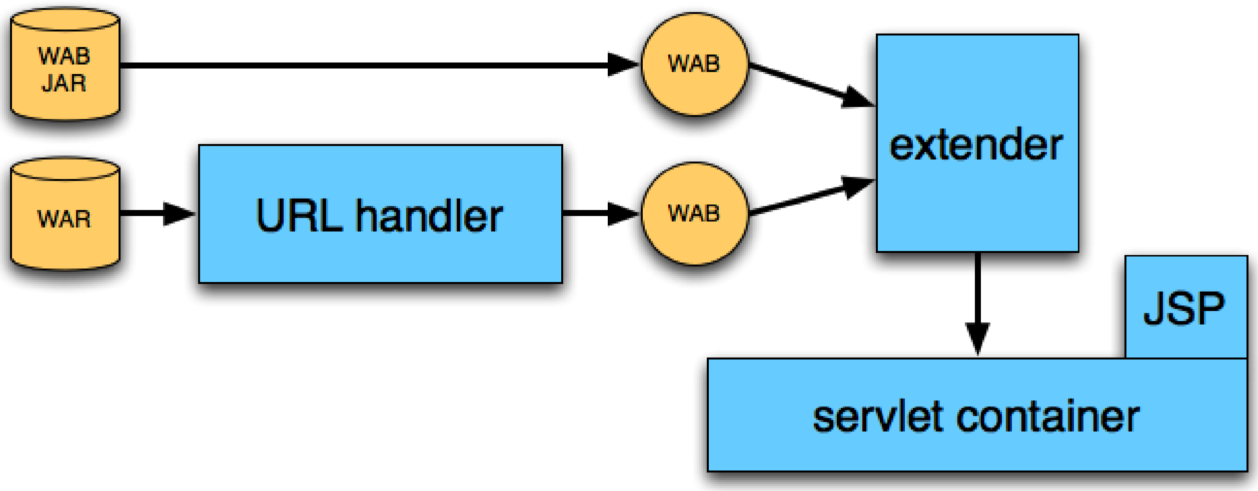
Virgo Bundlor

Embedded Technology

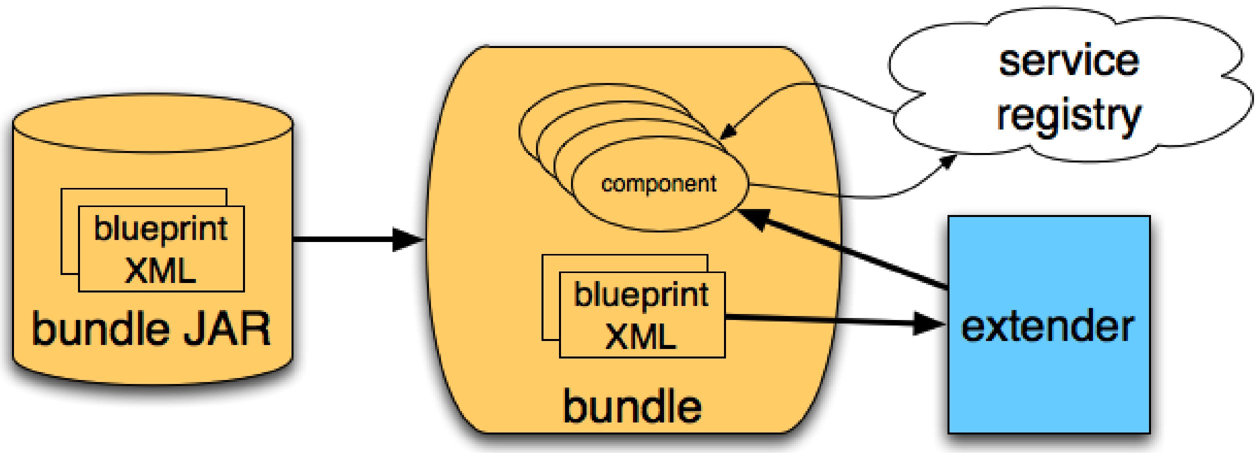
Tomcat

Jetty

Gemini Web



Gemini Blueprint



Gemini Management

Equinox OSGi Framework

Equinox p2

Equinox Region Bundle

Standards

Servlet Specification

Java EE Web Profile

OSGi Core

OSGi Web Applications

OSGi Blueprint

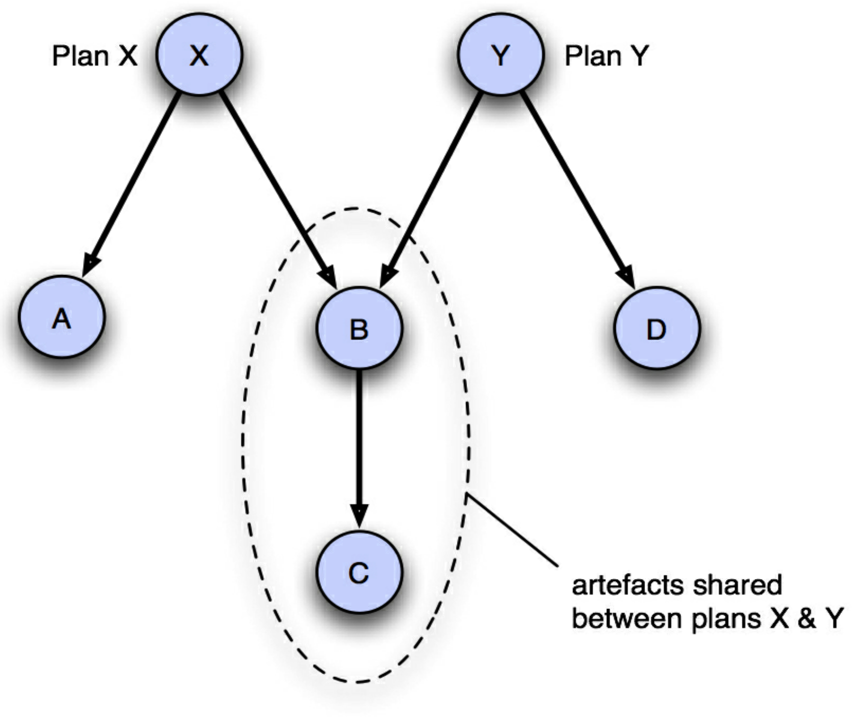
OSGi Declarative Services

OSGi Log Service

OSGi Event Admin Service

OSGi Configuration Admin Service

OSGi Subsystems



Further Information

- Virgo home page at http://www.eclipse.org/virgo/
- Virgo documentation at http://www.eclipse.org/virgo/documentation/
  - User Guide
  - Programmer Guide
  - Snaps Guide
- Greenpages sample application at http://www.eclipse.org/virgo/samples/
  - Sample ready to build with Maven
  - Greenpages Guide\
- Virgo Wiki at http://wiki.eclipse.org/Virgo
- Virgo FAQ at http://wiki.eclipse.org/Virgo/FAQ