Universidad Nacional de Quilmes

Departamento de Ciencia y Tecnología

Tecnicatura en Programación Informática

Trabajo de Inserción Profesional

Fronttier

Manejador de dependencias de la *capa de presentación* para plataformas basadas en lenguajes para la Máquina Virtual de Java

---

Alumno
Alan Rodas Bonjour
alanrodas@gmail.com

Directora
Dra. Gabriela B. Arévalo
garevalo@unq.edu.ar

septiembre 2014

Agradecimientos

Quiero agradecer a todos los profesores de la carrera de Tecnicatura en Programación Informática y a mis compañeros de clase quienes me acompañaron durante todo mi proceso de formación, me brindaron su apoyo, su compañía y cariño.

Quiero agradecer también a mi familia, por el aliento que me han brindado desde el principio de mis estudios, en especial a mis padres, quienes con mucho esfuerzo apostaron por mi educación.

Índice

Resumen

Al desarrollar sistemas web utilizando lenguajes que corren en la *JVM*, es generalmente necesario descargar *dependencias* para la *capa de presentación* y agregarlas al código del sistema de forma manual. Esta tarea lleva tiempo, es tediosa y propensa a errores. Actualmente no hay herramientas que corran enteramente sobre la *JVM* y que automaticen la tarea. Este trabajo propone una solución al problema mediante el desarrollo de un *manejador de dependencias* hecho completamente en *Scala*, capaz de descargar automáticamente estos archivos, organizarlos en el código y dejarlos listos para su posterior uso.

1. Introducción

En la presente sección se explicará cual es el contexto actual del desarrollo de software empresarial. Se analizará también como se organiza un sistema de este tipo y la técnica de reutilización de código mediante el uso de *dependencias*.

1.1. Desarrollo de sistemas web en la *JVM*

El presente trabajo se encuadra dentro del desarrollo de una herramienta para sistemas web desarrollados sobre la *JVM*. La elección no es casual y responde a la creciente popularidad de los lenguajes que corren sobre esta plataforma. También influye en la decisión los cambios por los que ha transitado el desarrollo de soluciones empresariales a lo largo de los años.
La plataforma de *Oracle*, la *Java Virtual Machine* (*JVM*), permite correr en ella cualquier lenguaje que compile a *bytecode Java*. En particular, el lenguaje *Java* fue el primero en compilar a esta plataforma. Con el tiempo, muchos otros lenguajes capaces de correr en esta plataforma han aparecido. *Scala*, *Clojure* y *Groovy* son algunos de los más populares y han dotado a la plataforma de muchas nuevas herramientas y posibilidades.
El apéndice A muestra como ha crecido esta plataforma y las posibilidades que ofrece.
Por otro lado, en la industria del software para desarrollo empresarial, las tecnologías web se han posicionado como la elección obvia. Esto se ha debido

a su facilidad de desarrollo, su bajo costo, y la facilidad de crecimiento que presentan los sistemas desarrollados de esta forma. Todo esto se combina con la creación del *HTML* 5 para el desarrollo de aplicaciones web, que permite un alto grado de usabilidad, así como la aparición de plataformas que brindan servicios de hosting de alto rendimiento y escalabilidad a bajo costo.
Estos factores, junto con una breve reseña de la evolución de la industria del software empresarial se puede encontrar en el apéndice B.

1.2. Estructura común de sistemas web

La mayoría de los sistemas web suelen dividirse en capas (También llamadas *layers* o *tiers*). Estas capas se dividen de forma tal que cada una esté encargada de partes muy puntuales de la lógica de la aplicación.
La mayoría de los autores suelen identificar tres capas, las cuales, dependiendo del autor, suelen recibir distintos nombres, aunque la estructura es siempre la misma. Estas capas son: la de interfaz o *presentación*, la de *lógica de negocios*, modelo o aplicación y la de *datos* o persistencia.
La capa de datos es la encargada de manejar los accesos a la base de datos, persistir la información, y todo lo relacionado a elementos de los que se deba guardar registro. La capa de lógica de negocios es la que contiene el código de las competencias de la aplicación, es decir, las cosas que la aplicación puede hacer, su funcionalidad. Finalmente, la capa de interfaz es la que permite acceder a la funcionalidad y visualizar los datos a los usuarios o clientes [Baarish,2002, pag 29-30]. La figura 1 muestra la arquitectura básica de una aplicación web según este esquema.

El presente trabajo se enfocará en la *capa de presentación*, aunque utilizará conceptos desarrollados para la *capa de lógica de negocios* para su desarrollo.

1.3. Reutilización de código mediante *dependencias*

Existen numerosos *frameworks*, *toolkits*, bibliotecas y porciones de código que los desarrolladores pueden utilizar en el código de su proyecto, de forma de obtener funcionalidades genéricas previamente desarrolladas. Estas porciones de código de terceros que utilizan los desarrolladores en sus proyectos reciben el nombre de *dependencias*.
En este trabajo se identificaran dos tipos de dependencias. las de la *capa de lógica de negocios*, dadas por *paquetes* de código *Java* compilado (archivos .jar o .war), y las de *capa de presentación* que consisten en (pero no se limitan a) archivos *CSS* y *JavaScript*. A su vez, estas dependencias pueden ser

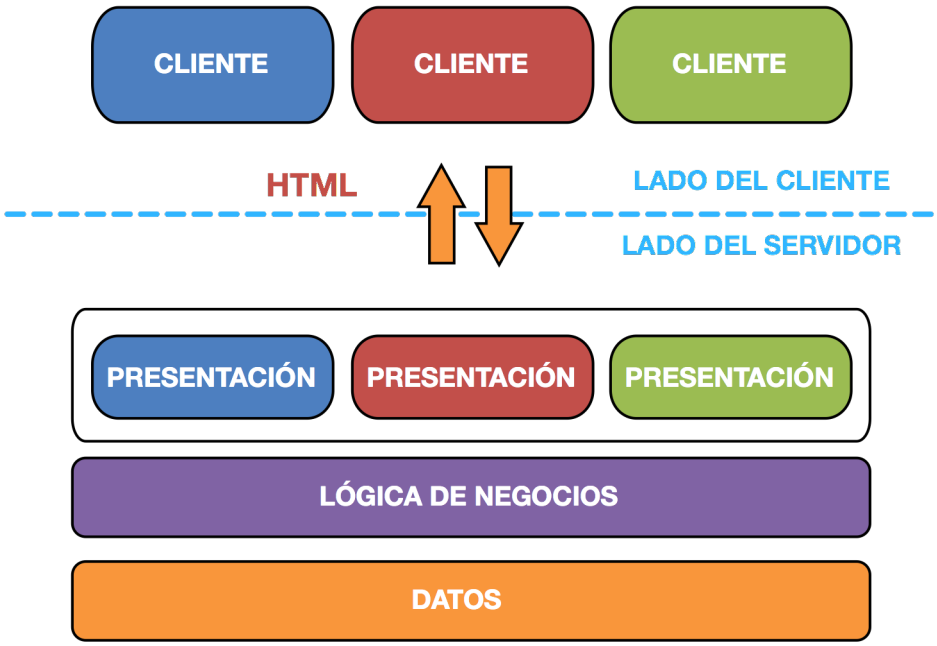
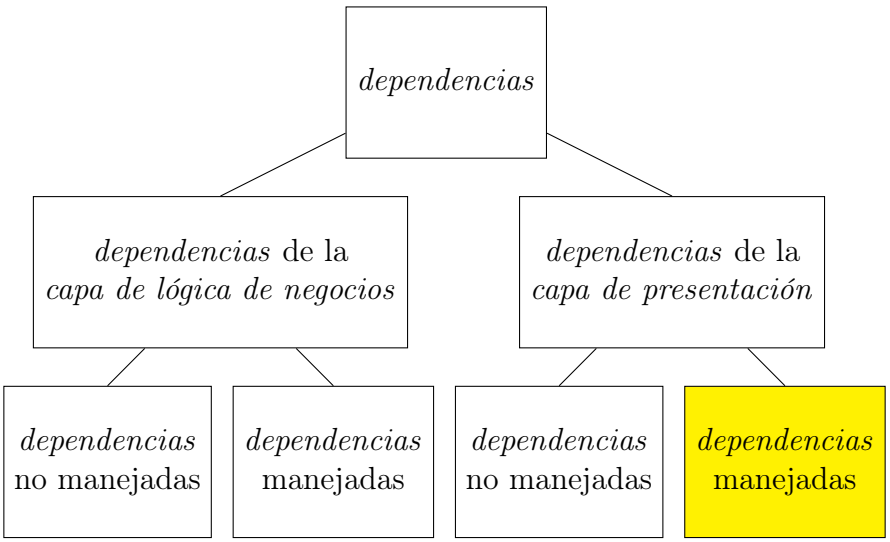


Figura 1: Arquitectura em capas básica de un sistema web.

manejadas o no manejadas.
Las *dependencias* no manejadas son código que se agrega manualmente al proyecto mediante la clásica técnica de *"copiar y pegar"* o agregando los archivos correspondientes a esa *dependencia* en alguna carpeta del proyecto (Proceso al que se denominará *"instalar"*). Por su parte, las *dependencias* manejadas consisten en algún tipo de descripción en un *archivo de configuración* o símil que declarará el código de terceros requerido. Un programa externo evaluara el *archivo de configuración* al momento de la compilación y se encargará de descargar e *instalar* las *dependencias* declaradas. Estos programas son conocidos como *manejadores de dependencias*.
La figura siguiente muestra de forma gráfica los tipos de *dependencias* para

una mejor comprensión de lo previamente expresado.



Las *dependencias* no manejadas implican una serie de pasos repetitivos y propensos a errores que el usuario debe realizar con el objetivo de poder correr su código. Por su parte, las *dependencias* manejadas son preferentes, ya que eliminan errores comunes y ahorran tiempo y trabajo a los desarrolladores. Además, los *manejadores de dependencias* suelen simplifican la complejidad asociada a instalar las *dependencias* anidadas[1] [Larman y Vodde,2010].
Las *dependencias* de la *capa de presentación* en la plataforma *Java*, suelen ser no manejadas. El presente trabajo se enfoca en transformar esas dependencias, en dependencias manejadas.
El anexo C muestra como funcionan los distintos *manejadores de dependencias* en las diferentes capas de aplicación.

2. Solución Propuesta

Es por ende de amplio interés para los desarrolladores contar con una herramienta que corra sobre la *JVM* y que permita manejar las *dependencias* de

[1] Asuma el lector un proyecto *A* con una *dependencia B*. El paquete *B* es a su vez un proyecto con *dependencias*, por ejemplo *C*. Para que *A* funcione correctamente, requiere *B* y este a su vez requiere *C*, haciendo que de forma transitiva *A* requiera *C*.

la *capa de presentación*.
El presente trabajo propone entonces el desarrollo de un *manejador de dependencias* hecho en *Scala*. El mismo utilizará *archivos de configuración* con distintos formatos para configurar las dependencias requeridos. Adicionalmente, se integrará con herramientas externas para facilitar su uso con los flujos de trabajo ya establecidos.

2.1. Características deseadas

En primer lugar, hay que tener en cuenta que lo que se desea es generar una herramienta que se posicione como el estándar de los lenguajes de la *JVM* en cuanto a lo que manejo de *dependencias* de la *capa de presentación* se refiere. Las siguientes son todas aquellas características deseables en la herramienta a desarrollar.

- Hecho enteramente sobre la *JVM*
- Posibilidad de ser usado desde cualquier lenguaje de la *JVM*
- Archivos de configuración personalizados, con varios formatos por defecto
- Múltiples repositorios
- Seguridad mediante autenticación en los repositorios
- Integración con estándares existentes de *manejadores de dependencias*
- Integración con tecnologías web existentes en la *JVM*

En primer lugar deberá correr enteramente sobre la *JVM*, evitando depender de la instalación de software adicional en el equipo y haciendolo independiente del sistema operativo. Por esto, es necesario que se encuentre desarrollada integramente en uno varios lenguajes que compilen a *bytecode Java*. Adicionalmente deberá poder ser utilizada desde cualquier otro lenguaje de la *JVM* de forma natural. En caso de características incompatibles de los lenguajes, debería proveer una capa de abstracción de forma tal que el uso sea transparente y completo. Incluso es deseable que pueda ser ejecutada en forma de programa independiente desde la linea de comandos. Esto último posibilitaría que pueda ser utilizado prácticamente desde cualquier lenguaje, aunque implica contar con acceso al sistema de archivos del sistema operativo.
La herramienta deberá ser capaz de descargar y acomodar los paquetes que el usuario determine como *dependencias* en un *archivo de configuración*. En lugar de forzar un formato especifico para el *archivo de configuración*, el *manejador de dependencias* desarrollado deberá brindar al usuario la posibilidad

de usar un formato que se adecue a sus necesidades[2].
Al existir soluciones como *Bower* o *Component* la herramienta debería complementarlas, permitiendo por ejemplo utilizar sus archivos de configuración y sus repositorios.
Finalmente, sería ideal contar con un repositorio central, en donde desarrolladores puedan subir sus paquetes, o al menos registrar los mismos, permitiendo así consultar la existencia de dependencias. Además, los usuarios podrían querer generar sus propios repositorios y hacer que la herramienta consulte en ellos, evitando la centralización del repositorio. Esta dualidad entre contar con un repositorio centralizado, y múltiples repositorios descentralizados, permite a organizaciones de distinto tamaño contar con posibilidades acorde a sus requerimientos, y se ha mostrado exitosa en proyectos como *Maven*.
Algunos de estos repositorios podrían ser privados, y por tanto, se requeriría acceso mediante un usuario y contraseña que la herramienta debe enviar de forma correcta y segura.
Finalmente, al existir numerosos *frameworks* para distintos lenguajes de la *JVM* que apuntan al desarrollo de aplicaciones web, sería deseable la integración con los mismos[3].
Podemos agregar además que la mayoría de los *manejadores de dependencias* actuales se basan en archivos, es decir, un archivo por dependencia. En las *dependencias* de la *capa de presentación* esto no es necesariamente cierto, y a veces una dependencia consiste en múltiples archivos. La herramienta tiene que proveer una forma de contemplar estos casos para permitir bajar todos los archivos que sean necesarios.
Otros requerimientos podrían surgir a futuro, pero no son tenidos en cuenta al momento de desarrollar esta solución.

2.2. Elección de tecnologías a utilizar

Se utilizará *Scala* para desarrollar la aplicación. La elección se basa en que la combinación de *Programación Orientada a Objetos* y *Programación Funcional* que presenta el lenguaje permiten desarrollar complejas soluciones en poco tiempo. Además, el código desarrollado es de fácil mantenimiento y

[2] Por ejemplo, los usuarios de *Apache Maven* podrían optar por un archivo de configuración en *XML* similar al que acostumbran y conocen, mientras que los usuarios de *SBT* podrían elegir un archivo de configuración cuya sintaxis asemeje los usados en esa herramienta. Debería ser lo bastante flexible para permitir la generación de formatos de archivo de configuración por los mismos usuarios de la herramienta de forma sencilla.

[3] Por ejemplo, múltiples *frameworks* utilizan sistemas de templates para generar el código *HTML* que finalmente será utilizado. Sería ideal poder brindar algún componente que integre los elementos descargados a los templates del usuario de forma automática.

gran expresibilidad, dado a la capacidad de *Scala* de crear sencillos DSLs.
Adicionalmente, este lenguaje presenta una gran cantidad de bibliotecas estándar que permiten realizar rapidamente tareas complejas. Por ejemplo, su biblioteca de análisis sintáctico (*parsing*) resultarán muy útiles cuando se requiera leer archivos de configuración.
Scala será utilizado en combinación con *SBT* para manejar las dependencias del proyecto.
Al momento de desarrollar herramientas de integración con otros lenguajes o tecnologías existentes es posible que sea necesario el uso de otros lenguajes, como *Java*, *Clojure* o *Groovy*.

2.3. Alcance del presente trabajo

El alcance del presente trabajo no abarca la totalidad de las caracteristicas deseadas expresadas en 2.1. El mismo solamente se enfocará en las funcionalidades más importantes y dejará asentadas las bases para el desarrollo de las partes no desarrolladas en trabajos futuros. Así, el enfoque será puesto en la estructura del sistema, orientada siempre a la extensibilidad del mismo.
Se presenta como objetivo el desarrollo de al menos las siguientes funcionalidades:

- Desarrollo del núcleo del sistema
- Descargar dependencias desde la web vía HTTP
- Descargar desde GitHub mediante *Git*
- Descargar desde un repositorio *SVN*
- Lectura de *archivos de configuración* en *XML* estilo *Apache Maven*
- Lectura de *archivos de configuración* en *XML* estilo *Apache Ivy*
- Lectura de *archivos de configuración* en estilo *SBT*
- Agregado de múltiples repositorios
- Usar la herramienta desde la linea de comandos
- Usar la herramienta desde *Scala*
- Usar la herramienta desde *Java*
- Integración de la herramienta con *Apache Maven*
- Integración de la herramienta con *SBT*
- Integración de la herramienta con Play! Framework

La herramienta será liberada como *Software Libre* con licencia *Apache* versión 2. Esto permitirá que la comunidad de desarrolladores expanda la herramienta para funcionar con todos los lenguajes de la *JVM*, en muchos más *frameworks*, y que se integre con una mayor cantidad de procesos.
Para lograr una buena recepción por parte de la comunidad de *Software Libre*,

la documentación del proyecto es sumamente importante. El desarrollo debe contar necesariamente con documentación que apunte a explicar la estructura del sistema, así como el código utilizado. También resulta interesante contar con un centro de referencia para futuros desarrolladores, por lo que colocar esta documentación a disposición de la comunidad a través de *Internet* es esencial.

2.4. Elección del nombre

La herramienta recibirá el nombre de *Fronttier*. El nombre deviene de un juego de palabras entre las palabras *front* (nombre que se le suele dar a la *capa de presentación*) y *tier* (El nombre que recibe cada una de las capas de la aplicación).

3. Guía de Uso

La presente sección muestra el uso de la herramienta desarrollada. Inicialmente se verá su uso mediante la *línea de comandos*. Luego, se mostrará la forma en la que la herramienta puede ser usada y extendida desde otros lenguajes.

3.1. Uso desde la *línea de comandos*

Para correr la aplicación desde la *línea de comandos* es necesario ejecutar el archivo JAR de la herramienta, a través de la *JVM*. El siguiente código muestra como ejecutar *Fronttier* desde la interfaz de *línea de comandos*:

```
$ java -jar fronttier.jar
```

Adicionalmente, se puede pasar una serie de argumentos a la aplicación para configurarla a gusto y necesidad del usuario, tal como se muestra a continuación:

```
$ java -jar fronttier.jar <argumentos>
```

Con intención de simplificar el trabajo del usuario al momento de ejecutar la aplicación, *Fronttier* incluye un simple programa que puede ser colocado en

cualquier lugar accesible para el usuario[4], permitiendo ejecutar la aplicación de forma directa, tal como se muestra en el siguiente ejemplo:

```
$ fronttier <argumentos>
```

En los próximas apartados se verán los argumentos posibles en más detalle, así como la forma de configuración de la herramienta.

3.1.1. Uso con los valores por defecto

Es posible llamar al programa sin ningún argumento y obtener funcionalidad del mismo mediante *"convenciones"* que *Fronttier* utiliza.
Al ser ejecutada sin ningún tipo de argumentos, *Fronttier* utilizará los valores por defecto de la aplicación. Estos incluyen: utilizar la *cache global*, buscar dependencias en un archivo de configuración con el formato estándar de *Fronttier* con el nombre *fronttier.ftt* e instalar las dependencias descargadas en el directorio local.
En las próximas secciones se verá con mayor profundidad cómo funcionan cada uno de los argumentos y a qué refieren los valores por defecto con mayor precisión.

3.1.2. Selección del archivo de configuración y su formato

Por defecto, *Fronttier* intentá localizar en el directorio actual un archivo con el nombre de *fronttier.ftt*. El mismo es donde se declaran las dependencias y repositorios del *proyecto*.
A su vez, el usuario puede especificar un archivo distinto en donde buscar dependencias. Esto se logra mediante la opción *--filename* o *-f* pasada como argumento a la aplicación e incluyendo luego el nombre del archivo que se desea utilizar para declaración de dependencias y repositorios. A continuación se puede observar un ejemplo de como llamar a la aplicación para que lea un archivo llamado *dependencies.ftt*:

```
$ fronttier --filename dependencies.ftt
```

Estos archivos pueden tener internamente distintos formatos, es decir, distintas formas de representar la información. Cada formato posee un nombre único por el cual el usuario puede identificarlo y, así, indicarle a la herramienta qué debería esperar de los contenidos del archivo de configuración. Esto

[4] Un lugar accesible es, por ejemplo, cualquier carpeta que se encuentre dentro de la variable de entorno *"PATH"* en el sistema operativo del usuario.

se logra mediante el argumento *--config* o *-c* . El siguiente código muestra como elegir un formato con nombre *"xml"*:

```
$ fronttier --config xml
```

Los formatos registrados por defecto en *Fronttier* son:

- fronttier
- xml
- attrxml
- scale

Sin embargo, el usuario puede registrar nuevos formatos mediante extensiones a la herramienta. En caso de no especificar el formato del archivo, el formato *fronttier* es el utilizado.
El formato elegido del archivo repercute sobre el nombre por defecto del mismo. Por ejemplo, el nombre de archivo por defecto para el formato *xml* es *fronttier.xml*, al igual que para el formato *attrxml*. Por su parte el formato *fronttier* buscará un archivo por el nombre de *fornttier.ftt*.
Los argumentos *--config* y *--filename* se pueden usar al mismo tiempo dando lugar a cualquier combinación de nombre de archivo y formato. Por ejemplo, el siguiente código muestra como utilizar *Fronttier* para que utilice un archivo *dependencies.xml* que posee formato *xml*.

```
$ fronttier --config xml --filename dependencies.xml
```

La estructura interna de cada uno de los formatos, así como la forma de declarar dependencias y repositorios es especificada en la sección 3.4.

3.1.3. Elección de carpeta de destino

Por defecto *Fronttier* trabaja sobre la carpeta actual, y es allí en donde se *instalan* las dependencias descargadas.
No obstante, la carpeta sobre la que trabajará la herramienta puede ser especificada, de forma tal que las *dependencias* se instalen en alguna carpeta especifica o una subcarpeta de la carpeta local. Para ello se utiliza el comando *--destination* o *-d* seguido de la ubicación en donde se desean descargar las dependencias.

```
$ fronttier --destination <folder>
```

El valor por defecto es ., es decir, la carpeta actual. Por tanto, los siguientes

dos comandos son equivalentes

```
$ fronttier
```

```
$ fronttier --destination .
```

El elegir la carpeta de destino resulta útil en proyectos en donde el código compilado termina en una carpeta especifica que no forma parte del proyecto y que se elimina cada vez que se recompila el mismo. Así, o más común es elegir un subdirectorio de la carpeta de trabajo en donde se deben colocar las dependencias previo a ejecutar el sistema. El siguiente ejemplo muestra como indicarle a *Fronttier* que utilice como carpeta de descarga un subdirectorio con el nombre de *"WEB-INF/html"*:

```
$ fronttier --destination ./WEB-INF/html
```

Si bien no es algo común, se pueden especificar también direcciones absolutas en lugar de relativas. En caso de que las direcciones contengan espacios se pueden utilizar comillas simples o dobles para escapar la dirección como se muestra en el siguiente ejemplo.

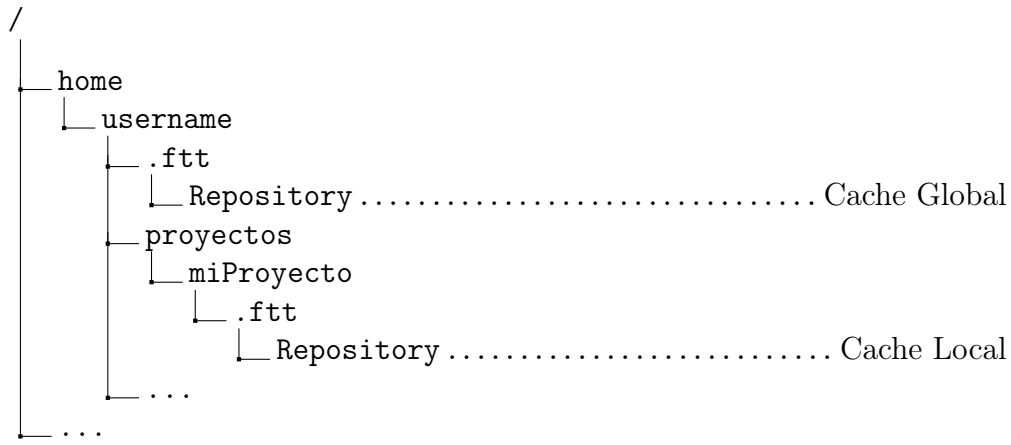
```
$ fronttier --destination "C:\Web Dependencies"
```

Se debe tener en cuenta que *Fronttier* no realiza ninguna acción para limpiar la carpeta de destino, con lo que el usuario deberá garantizar su limpieza previo a futuras ejecuciones.

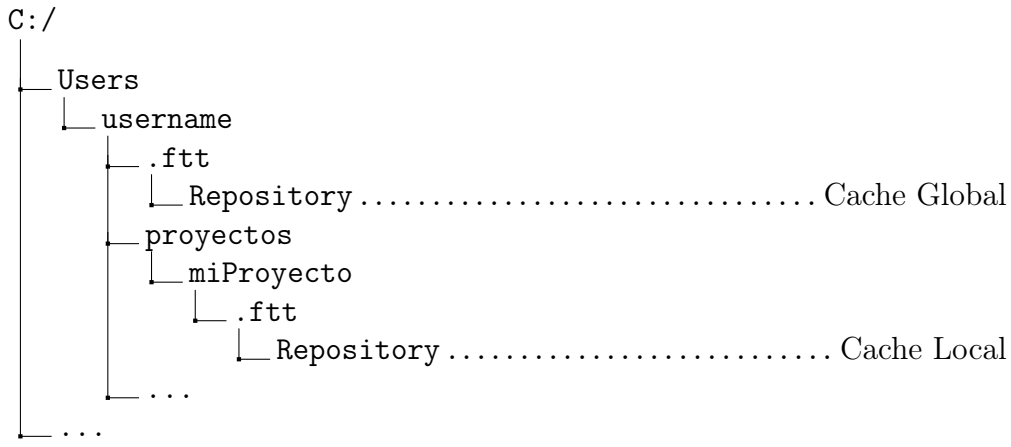
3.1.4. Cache local y cache global

Fronttier mantiene en la maquina del usuario una cache con todos los archivos que va descargando a lo largo de su uso. Esto permite acelerar el proceso de *instalación* de las dependencias en un proyecto particular. Así, *Fronttier* mantiene dos lugares que utiliza de cache, la *cache local* y la *cache global*.
La *cache local* tiene alcance solo sobre el proyecto de código sobre el cual se está trabajando (Es decir, el *archivo de configuración* que se está leyendo y el directorio sobre el cual se están instalando las dependencias). Su uso se limita a evitar tener que descargar las dependencias en subsequentes ejecuciones del programa en ese proyecto.
Por su parte, la *cache global* contiene las dependencias descargadas por todos los proyectos. Esta cache se utiliza no solo evitar la descarga de dependen-

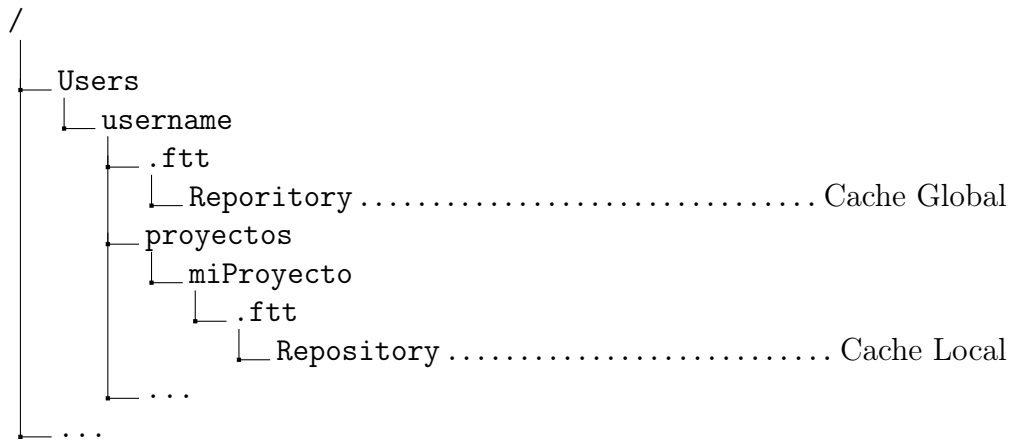
cias en futuras ejecuciones del programa sobre un proyecto determinado, sino además evitar la descarga en caso de que dos proyectos independientes requieran eventualmente la misma dependencia.
Las caches permiten adicionalmente trabajar sin la necesidad de una conexión a Internet, ya que no se deben descargar las dependencias, pues ya se encuentran descargadas.
La *cache global* se encuentra en una carpeta creada por *Fronttier* en el directorio personal del usuario. Por su parte, la *cache local* se encuentra en una carpeta similar, pero localizada en el directorio del proyecto sobre el que aplica dicha cache. Las estructuras de carpetas expresadas en 1, 2 y 3 muestran la ubicación de las carpetas de cache en distintos sistemas operativos.



Estructura de Carpetas 1: Caches en un sistema Linux



Estructura de Carpetas 2: Caches en un sistema Windows



Estructura de Carpetas 3: Caches en un sistema Mac OS X

Para comprender un poco mejor el sistema de caches es necesario comprender como *Fronttier* descarga dependencias. En primer lugar, *Fronttier* leerá el *archivo de configuración* y analizará las dependencias a instalar. Para cada *dependencia*, se evaluará si existe en la *cache local*, si se encuentra, se copiara desde la misma a la carpeta en donde se requiera instalar la *dependencia*. En caso de no encontrarse, se recurrirá a buscarla en la *cache global* para copiarla desde allí. Finalmente, si no se encuentra en las caches, se descargará de alguno de los *repositorios en línea* disponibles. *Fronttier* indicará un error al usuario si no encuentra en estos últimos la *dependencia*.
Fronttier copiará por defecto las *dependencias* que descargue desde *repositorios en línea* a la *cache global* para futuros usos. Este comportamiento esta dado mediante el *flag --global* o *-g* . Así, los siguientes dos códigos son equivalentes:

```
$ fronttier
```

```
$ fronttier -g
```

Mediante el *flag --local* o *-l* se puede indicar que no se desea copiar las *dependencias* a la *cache global* sino a la local. Esto puede resultar útil en algunos proyectos. El *flag --nocache* evita que se guarden las *dependencias* descargadas en cualquiera de las caches.
Por ultimo, se puede utilizar el *flag -f* o *--force* para forzar la descarga de una *dependencia* desde los *repositorios en línea* incluso si se encuentran en alguna cache. Esta acción reemplazará la versión actual en la cache con la

nueva versión.
Todos estos *flags* pueden combinarse para dar lugar a acciones determinadas. Por ejemplo, el siguiente código muestra como descargar dependencias sin guardarlas en la cache, y utilizando siempre la versión disponible en los *repositorios en línea*.

```
$ fronttier --nocache -f
```

También es posible descargar las dependencias en la *cache local*, independientemente de si se encuentran en alguna de las caches, como muestra el siguiente código.

```
$ fronttier -lf
```

Note que los *flags* en su formato corto pueden agruparse utilizando un solo signo - y pasando todos los *flags* juntos. Así *-lf* es equivalente a *-l -f* .

3.1.5. Modo verbose

Por defecto *Fronttier* brinda muy poca información al usuario con respecto a que está haciendo en cada momento, indicando solamente los errores que se puedan llegar a producir. Es posible, sin embargo, cambiar dicho comportamiento activando el modo *"verbose"*.
Precisamente el *flag --verbose* o *-v* activa este modo permitiendo ver información de todos los pasos que la herramienta va realizando. Esto resulta útil en caso de errores inesperados para poder descubrir el origen de los mismos.

3.1.6. Agregar archivos de expansión

La herramienta *Fronttier* puede ser extendida por código del usuario. Esto permite, entre otras cosas, agregar nuevos formatos de *archivo de configuración*.
Para extender el programa, basta cargar los archivos JAR (código compilado a *bytecode Java*) que contienen las extensiones. Esto se realiza mediante el argumento *--load* o *-l* el cual recibe el o los nombres de archivo a cargar, al que se denomina *plugin*. Por ejemplo, para cargar los archivos *"first.jar"* y

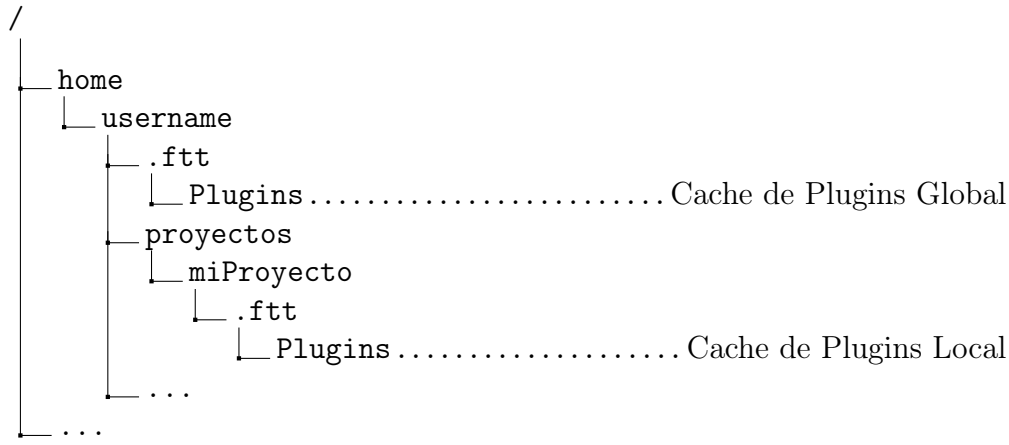
second.jar se debería ejecutar la aplicación de la siguiente forma:

```
$ fronttier -l first second
```

La extensión del archivo no es necesaria pues se asume que siempre será .jar. Los archivos de extensión, en este caso, se encuentran en la carpeta actual desde donde se ejecuta la aplicación. También se pueden utilizar rutas absolutas o relativas para llamar a los archivos como se muestra a continuación:

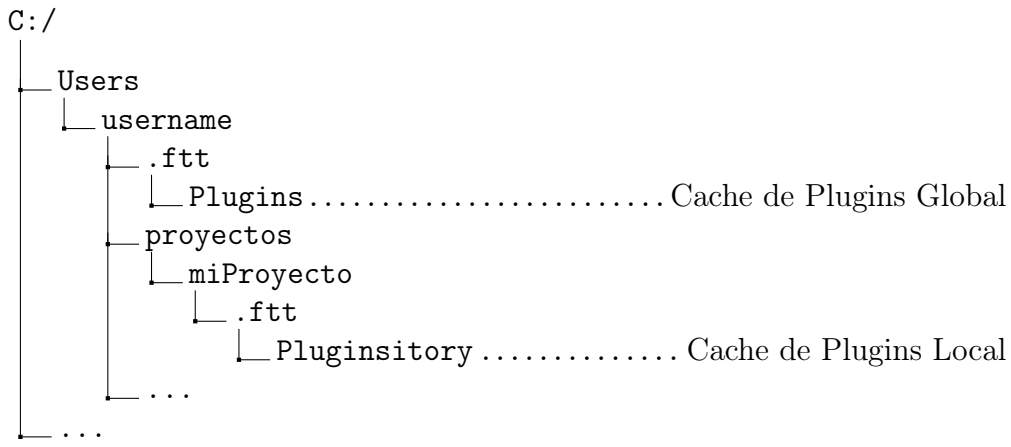
```
$ fronttier --load "C:\Fronttier Extensions\first "
```

Finalmente, cabe destacar la posibilidad de *instalar* extensiones en una carpeta global o local de forma tal que puedan ser utilizados en cualquier momento. Esta carpeta actúa de forma similar a la cache para los archivos de dependencia. De hecho, su ubicación es precisamente en un directorio hermano a la carpeta del repositorio de cache. En las estructuras de carpetas 4, 5 y 6 se muestra la ubicación de las carpetas de extensiones de *Fronttier*.

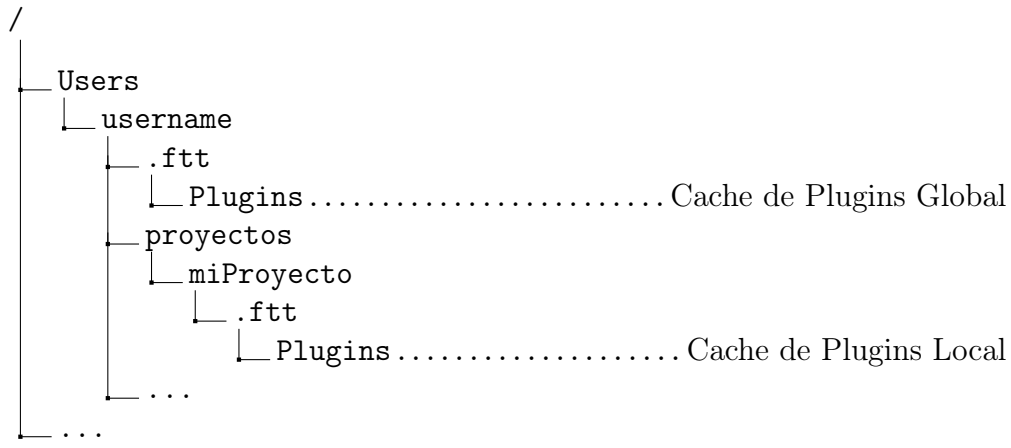


Estructura de Carpetas 4: Cache de Plugins en un sistema Linux

Los archivos JAR que se encuentren en una carpeta de *cache de plugins* pueden ser utilizados como si se encontraran en la carpeta actual, es decir, mediante su nombre.
Cuando *Fronttier* recibe un *plugin* a cargar que no contiene ruta, primero evaluará su existencia en la carpeta actual, luego en la *cache de plugins local*, y finalmente en la *cache de plugins global*. Utilizará entonces aquel que encuentre primero, indicando un error en caso de no encontrar coincidencias.



Estructura de Carpetas 5: Cache de Plugins en un sistema Windows



Estructura de Carpetas 6: Cache de Plugins en un sistema Mac OS X

3.2. Comandos de manejo de la aplicación

Fronttier incluye una serie de comandos adicionales para facilitar el manejo de la herramienta y realizar tareas básicas sin necesidad de utilizar un archivo de configuración o realizar dichas tareas manualmente. Esto incluye cosas como instalar dependencias, copiar dependencias a la cache, borrar dependencias de la cache, instalar o desinstalar *plugins*, etc.

3.2.1. Instalar dependencias

Es posible instalar dependencias sin la necesidad de declararlas en un archivo de configuración mediante el uso del comando *install*. Este comando tomará como argumento el nombre de la dependencia a instalar. El siguiente código

muestra como funciona el comando:

```
$ fronttier install {company}:{package}:[{version}]
```

Donde *company* y *package* son cadenas de texto que hacen referencia inequívoca a la dependencia a instalar. Opcionalmente *version* puede ser especificado para indicar una versión especifica del mismo (Si no se indica, se descargará la versión más reciente). El siguiente código muestra un ejemplo puntual para descargar la biblioteca *jQuery* en su versión 1.9.0:

```
$ fronttier install org.jquery:jquery:1.9.0
```

La mayoría de los argumentos especificados en la sección 3.1 pueden ser utilizados, quedando exceptuados aquellos que refieren a los *archivos de configuración*. Por ejemplo, el argumento *--destination* o *-d* puede ser utilizado como se muestra a continuación:

```
$ fronttier install org.jquery:jquery:2.0 -l -d ./WEB-INF/html
```

La búsqueda del archivo en las caches es exactamente igual a cuando se trabaja con un *archivo de configuración*.

3.2.2. Cachear dependencias

El comando *cache* y el comando *delete* permiten guardar y borrar dependencias en la cache de dependencias de *Fronttier*. Se puede entonces pedir que un determinado archivo se descargue a la cache de la siguiente forma:

```
$ fronttier cache org.jquery:jquery:1.9.0
```

Los argumentos *--global* o *-g* y *--local* o *-l* pueden ser utilizados para indicar exactamente sobre que cache se va a trabajar. Por ejemplo, el siguiente código remueve un archivo de la *cache local*:

```
$ fronttier remove org.jquery:jquery:1.9.0 -l
```

3.2.3. Instalar *plugins*

Los *plugins* pueden ser instalados a las *caches de plugins* mediante los comandos *plug* y desinstalados con el comando *unplug*. Los *flags* *--global* o *-g* y *--local* o *-l* pueden ser utilizados. A continuación se muestra como instalar un *plugin* desde la carpeta local a la *cache de plugins global*:

```
$ fronttier plug myCoolPlugin
```

Y aquí como desinstalar un *plugin* desde la *cache de plugins local*:

```
$ fronttier unplug myCoolPlugin -l
```

3.3. Archivos de defaults de la herramienta

Fronttier trabaja con varios valores predeterminados que incluyen cosas como el formato del *archivo de configuración* a utilizar o que cache usar. Estos valores pueden ser modificados mediante los llamados archivos de defaults de *Fronttier*.
Así, pueden existir una serie de archivos que deben tener por nombre *.fttrc* y que modifican los valores por defecto de la herramienta cuando esta es utilizada desde la linea de comandos. Los archivos deben encontrarse en alguna de las siguientes 3 ubicaciones:

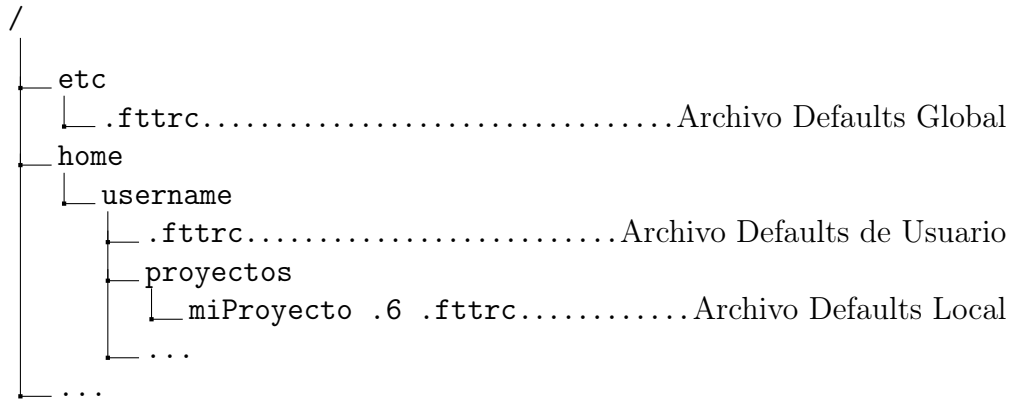
- La carpeta de trabajo actual, o carpeta de proyecto
- La carpeta personal del usuario
- La carpeta de archivos de configuración, comúnmente denominada /etc

Los ejemplos de estructuras de carpetas 7, 8 y 9 muestran la ubicación de cada uno de estos archivos en los sistemas operativos más comunes.

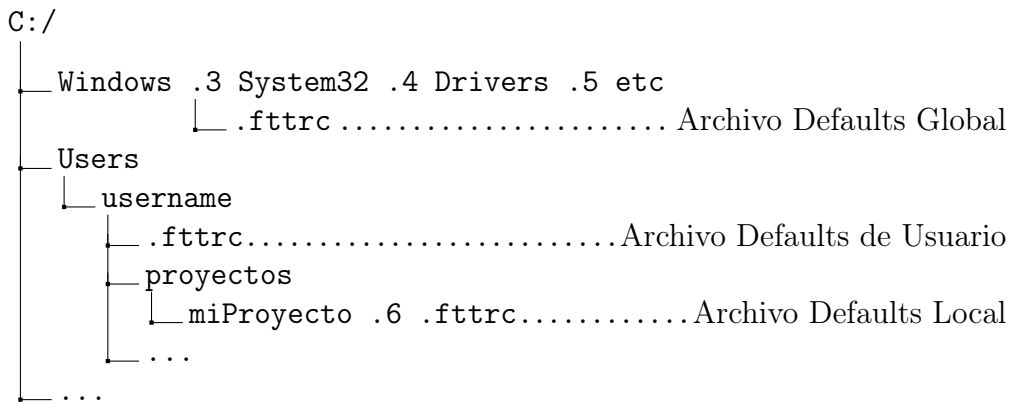
Cada archivo posee configuraciones para la ejecución por defecto de *Fronttier*.
El manual de la aplicación provee las indicaciones necesarias para crear y modificar estos archivos, así como los contenidos esperados en los mismos.

3.4. Distintos formatos del archivo configuración

Como ya se ha mencionado en el presente, existen múltiples formatos de archivos de configuración posibles. El formato elegido usando el método expresado en la sección 3.1.2 afecta así a la forma en la que deben estar declaradas



Estructura de Carpetas 7: Archivos de defaults en un sistema Linux

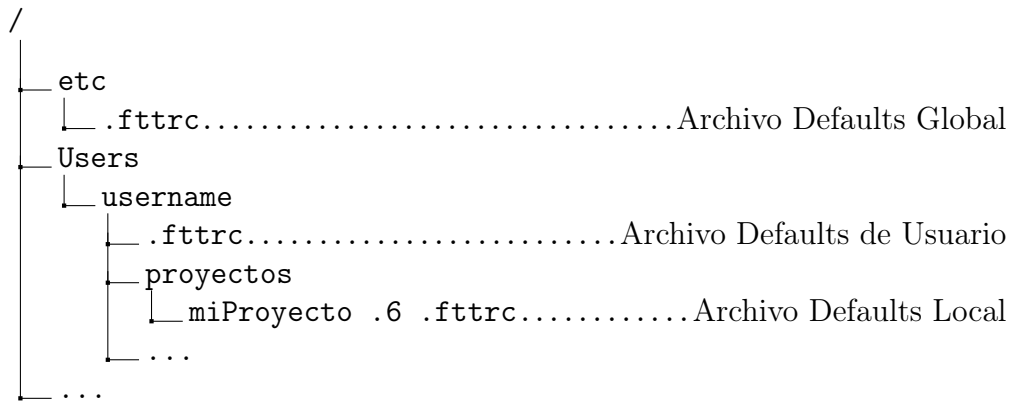


Estructura de Carpetas 8: Archivos de defaults en un sistema Windows

las dependencias en el archivo. En las secciones siguientes se muestran los formatos incluidos por defecto con la herramienta.

3.4.1. Formato de configuración estándar

El formato de configuración estándar o formato *Fronttier* es el método por defecto para declarar las dependencias y repositorios.
En este formato se deben declarar dos secciones. La sección *repositories* la cual es opcional, indica los repositorios adicionales que se desea utilizar, uno por linea, o separados por punto y coma. La sección *dependencies* es obligatoria e indica las dependencias requeridas por el proyecto. Las secciones se encuentran delimitadas por llaves, y solo puede haber una sección *dependencies* y una sección*repositories*.
En la sección *dependencies* cada *dependencia* es declarada en una nueva lí-



Estructura de Carpetas 9: Archivos de defaults en un sistema Mac OS X

nea (o separandolas por punto y coma) indicando la *company* seguida de dos puntos y el nombre. De forma opcional se puede agregar dos puntos luego del nombre e indicar la versión puntual que desea instalar.
En este formato de archivo, los espacios en blanco no son tenidos en cuenta y se pueden agregar comentarios de linea mediante el caracter #.
El código 1 muestra un ejemplo de una declaración de este formato de archivo.

```
repositories {
  http://myrepository.com
  https://some-other-repo.org/repository
}

dependencies {
  org.jquery:jquery:2.1.1; org.jquery:jquery-ui:1.11.0
  # Esto es un comentario
  com.twitter:bootstrap:3.2.0
}
```

Código 1: Ejemplo del formato *fronttier*

El formato Backus-Naur Extendido para este archivo se muestra de forma simplificada en el código 2.

Este formato se encuentra registrado bajo el identificador *fronttier*.

```
delimiter = ";"|"\n"
string = "\s", {"\s"}
repository^^I= string, delimiter
repositories = "repositories {" {repository} "}"
dependency^^I= string, ":", string, [":", string]
dependencies = "dependencies {" {dependency} "}"
depsfirst = dependencies, [repositories]
depslast = [repositories], dependencies
grammar = depsfirst | depslast
```

Código 2: EBNF de *fronttier*

3.4.2. Formato de configuración estilo *Maven*

El formato *XML* permite configurar las dependencias y repositorios utilizando *XML*, algo que puede ser más cómodo para los usuarios de *Maven*. En este formato los repositorios, *dependencias* y todas las propiedades de cada una de las dependencias se expresan como texto entre tags *XML* anidados. Los tags posibles y sus anidaciones están dadas por un archivo DTD, el cual se puede apreciar en el código 3.

```
<!ELEMENT fronttier (repositories?, dependencies)>
<!ELEMENT repositories (repository*)>
<!ELEMENT repository (#CDATA)>
<!ELEMENT dependencies (dependency*)>
<!ELEMENT dependency (group, name, version?)>
<!ELEMENT group (#CDATA)>
<!ELEMENT name (#CDATA)>
<!ELEMENT version (#CDATA)>
```

Código 3: DTD del formato *xml*

Los nombres y las convenciones utilizadas en el formato se corresponden completamente con aquellos usados por *Apache Maven* para una más fácil adopción por parte de los usuarios de esta herramienta. En el código 4 se muestra un ejemplo del formato *xml*.

3.4.3. Formato de configuración estilo *Ivy*

Además del formato *xml*, el formato *attrxml* también utiliza *XML* para de-

```
<?xml version="1.0"?>
<!DOCTYPE fronttier-xml>
<fronttier>
  <repositories>
    <repository>http://myrepository.com</repository>
    <repository>https://some-other-repo.org/repository</repository>
  </repositories>
  <dependencies>
    <dependency>
      <group>org.jquery</group>
      <name>jquery</name>
      <version>2.11.0</version>
    </dependency>
    <dependency>
      <group>org.jquery</group>
      <name>jquery-ui</name>
    </dependency>
  </dependencies>
</fronttier>
```

Código 4: Ejemplo del formato *xml*

clarar *dependencias*. A diferencia del anterior, este formato es más compacto, ya que los valores se declaran como atributos de los tags *XML*.
Este formato respeta las convenciones impuestas por *Apache Ivy*, pudiendo ser de interés para los usuarios de dicha herramienta.
La estructura de estos archivos está dada por el DTD que se muestra en el código 5

El código en 6 muestra un ejemplo de este formato.

3.5. Integración en sistemas

Fronttier no es solamente una herramienta que corre desde la *línea de comandos*, sino que, por ser un archivo JAR, puede ser utilizado como una biblioteca de la plataforma *Java*. Esto permite que se utilice desde otros programas escritos para cualquiera de los lenguajes que corren sobre la *JVM*.
Para cada acción que se puede ejecutar desde la *línea de comandos Fronttier* provee objetos y métodos que pueden ser utilizados desde el código del usuario. Esta es la forma en que se pueden crear extensiones para la herramienta,

```
<!ELEMENT fronttier (repositories?, dependencies)>
<!ELEMENT repositories (repository*)>
<!ELEMENT repository EMPTY>
<!ATTLIST repository url CDATA #REQUIRED>
<!ELEMENT dependencies (dependency*)>
<!ELEMENT dependency EMPTY>
<!ATTLIST dependency group CDATA #REQUIRED>
<!ATTLIST dependency name CDATA #REQUIRED>
<!ATTLIST dependency ver CDATA #IMPLIED>
```

Código 5: DTD del formato *attrxml*

```
<?xml version="1.0"?>
<!DOCTYPE fronttier-attrxml>
<fronttier>
  <repositories>
    <repository url="http://myrepository.com"/>
    <repository url="https://some-other-repo.org/repository"/>
  </repositories>
  <dependencies>
    <dependency group="org.jquery" name="jquery" ver="2.11.0"/>
    <dependency group="org.jquery" name="jquery-ui"/>
  </dependencies>
</fronttier>
```

Código 6: Ejemplo del formato *attrxml*

como por ejemplo nuevos formatos de *archivos de configuración*.
El código de *Fronttier* es compatible por defecto con *Scala*, por estar hecho en este lenguaje. También se proveen bindings para *Java* que permiten utilizar todas las funcionalidades.
La mayoría de los lenguajes que corren desde la *JVM* permiten importar código *Java* y utilizarlo. Sin embargo, el código *Java* no necesariamente se adapta a las técnicas de programación que se utilizan en ese lenguaje. Por eso, nuevos bindings que permitan usar la herramienta de forma transparente en estos lenguajes pueden ser creados a futuro.
En particular, el paquete `com.alanrodas.fronttier` provee una serie de traits, objetos y clases para extender la aplicación o usar sus características. A su vez el paquete `com.alanrodas.fronttier.java` provee los bindings correspondientes para realizar las mismas acciones desde *Java*. La documen-

tación completa se distribuirá oportunamente junto con la aplicación.

4. Desarrollo

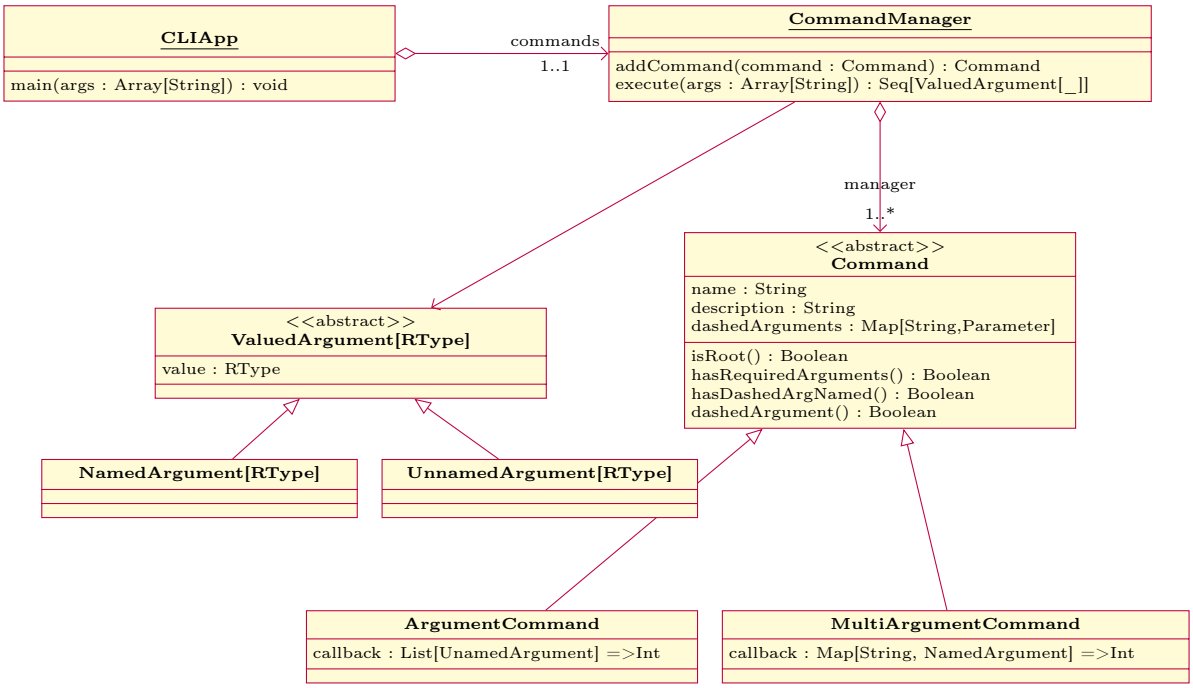
El sistema se desarrolló realizando módulos independientes entre sí, y finalmente conectando todo en el sistema *Fronttier*.

4.1. El modelo general

4.2. Modelo del parser de *línea de comandos*

El módulo *sCaLIapp* es una biblioteca que funciona de forma independiente para crear aplicaciones de *línea de comandos*.
A través de un simple DSL y mediante extender la clase `CLIApp` se pueden crear aplicaciones que toman comando de forma rápida. Cada comando tiene una función asociada la cual se activa al recibir la aplicación dicho comando. Además, se pueden definir de forma rápida argumentos, en su modo largo y corto, *flags* y argumentos que reciben parámetros.
El código principal se encuentra en el paquete `com.alanrodas.scaliapp`, el cual, al importarlo, provee todas las clases y funciones necesarias para utilizar el DSL.
El UML en 1 da una vista simplificada de sCaLIapp. `CLIApp` no es más que una clase que provee las mismas funcionalidades que `App` en *Scala*, pero evaluando todos los comandos declarados y delegando en `CommandManager` su procesamiento.
Para cada `Command` existe un `callback`, una función que dado una lista de `ValuedArgument` (los valores pasados a la aplicación para ese comando) devuelven un entero. El entero devuelto representa la salida del programa (cero para programas que terminaron correctamente, o el número de error correspondiente en caso contrario). A su vez, se espera que la ejecución de dicho `callback` genere algún efecto, por ejemplo, mostrando un mensaje en la terminal del usuario, o ejecutando una porción de código determinado.

`sCaLIapp` es una biblioteca completamente independiente del proyecto *Fronttier*, y por tanto, tiene sus propios objetivos, trabajos a futuro, etc.
La idea de `sCaLIapp` es la de poder construir aplicaciones que corran en la



UML 1: Diagrama general de sCaLIapp.

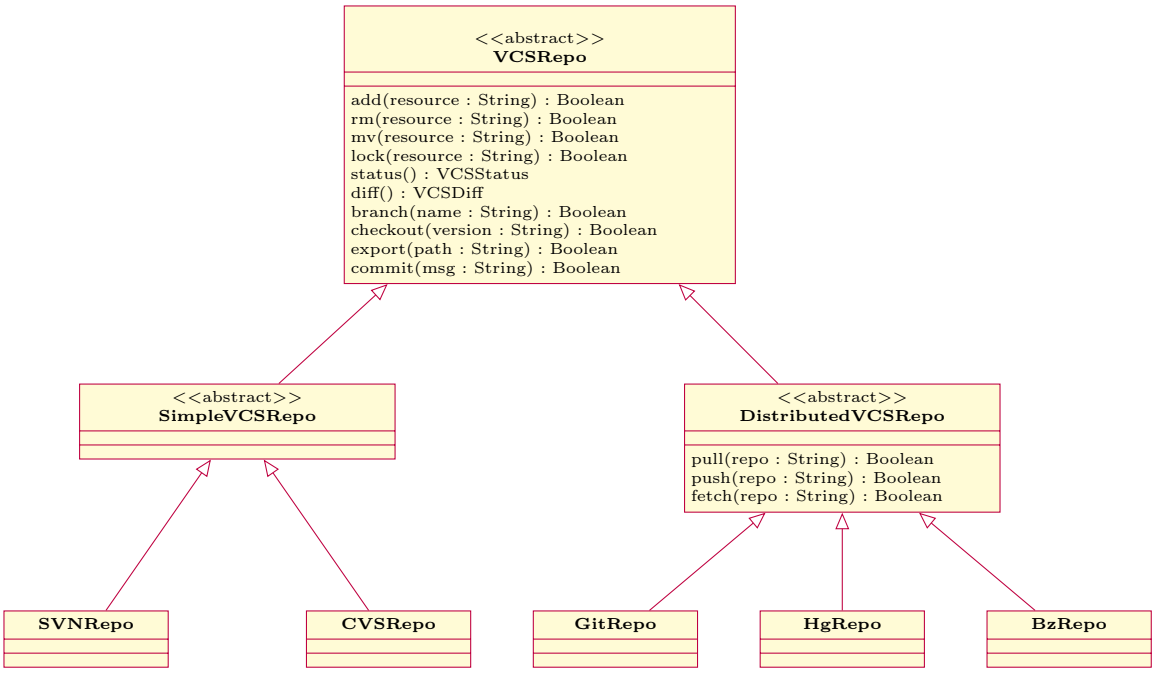
máquina del cliente de forma muy rápida y sencilla. No solo eso, sino que la idea general es que la herramienta sea lo suficientemente flexible como para soportar cualquier aplicación que corra desde la línea de comandos, incluyendo aplicaciones interactivas. Es por esto que el trabajo realizado en el presente en cuanto a *sCaLIapp* no se da por concluido pues solamente se limitó al desarrollo de la parte que fuera imprescindible para *Fronttier*.

4.3. Modelo de *archivos de configuración*

4.4. Modelo de sistema de descargas

El modelo del sistema de descargas cuenta con dos partes. La parte de descargas mediante HTTP y que se encuentra dentro del modulo de *Fronttier*, y la parte de descargas desde sistemas de control de versiones, el cual hace uso del modulo *ScalaVCS*. Hablaremos primero de este último.
ScalaVCS es una biblioteca que intenta generar una capa de abstracción sobre el funcionamiento y de los distintos sistemas de control de versiones

(VCSs), como *Git*, *SVN*, *CVS*, *Bazaar* y *Mercurial*.
Este modulo parte de la base de que existen una serie de acciones que el usuario puede desear realizar sobre un VCS, independientemente de cual sea este. Tales operaciones incluyen añadir un archivo al sistema de control, remover un archivo, listar los archivos agregados, salvar la versión actual, volver a una versión anterior, ver las versiones salvadas o comparar dos versiones salvadas, entre otras.
A su vez, *ScalaVCS* distingue dos tipos de sistemas de control de versiones, aquellos que funcionan con un solo servidor (*CVS*, *SVN*) y los distribuidos (*Git*, *Mercurial*, *Bazaar*). Los sistemas distribuidos tienen acciones adicionales, como sincronizar de forma distribuida, o bajar una versión especifica.
El diagrama UML 2 muestra la estructura básica de los repositorios de *ScalaVCS*. Existe a su vez un objeto para cada tipo de sistema de control de versiones, que actúa a modo de constructor. La sintaxis de *Scala* junto con este patrón permite obtener un DSL muy claro y conciso.



UML 2: Diagrama general de los repositorios de ScalaVCS.

ScalaVCS utiliza parámetros implícitos en todos los objetos que actúan de constructores. Estos parámetros pueden ser importados desde el paquete `com.alanrodas.scalavcs.connectors` y sus subpaquetes, pasándose auto-

maticamente a los objetos constructores. Tienen por objetivo proveer el conector que se utilizará como cliente para un determinado sistema de VCS. Así, es posible usar un conector que utilice la aplicación *Git* mediante llamadas a la misma a través del sistema operativo, o utilizar una implementación realizada en completamente en *Java*, como *SVNKit*.
Fronttier utiliza entonces a *ScalaVCS* para descargar todo lo referente a sistemas de control de versiones. Por otro lado, utiliza la biblioteca *RaptureIO* para descargar contenidos desde *Internet*.
RaptureIO provee una base de abstracción sobre el proceso de entrada y salida en *Scala*.

5. Conclusiones

Se ha logrado construir una herramienta que es altamente extensible, y que pone al desarrollador en una posición de pocas restricciones con respecto a las extensiones posibles a la misma.
Si bien la herramienta puede no lograr convertirse en un estándar prontamente, no es difícil pensar que futuras modificaciones, mayor cantidad de interacción con otras herramientas y la creación de repositorios libres y comunes puedan llevar a *Fronttier* a convertirse en un referente en cuanto a manejo de dependencias de la *capa de presentación* se refiere.
Durante el proceso de programación de la herramienta, se ha podido observar como el desarrollo de módulos pensados como una API son fundamentales para una fácil y rápida extensión del sistema.

6. Trabajo a futuro

Como se refleja en la sección 2.3, este trabajo tiene un alcance limitado con respecto a los deseos expresados en la sección 2.1. En particular quedan expresadas pero pendientes de desarrollo las siguientes tareas:

- Soportar más formas de descarga a través de otros métodos sistemas de control de versiones, como *Mercurial*, *Bazaar*, *CVS*, etc.
- Mayor cantidad de formatos de *archivos de configuración*
- Seguridad en los repositorios mediante autenticación
- Funcionar desde otros lenguajes de la *JVM* como *Clojure* y *Groovy*

- Integración con más cantidad de procesos, como *Grape*, *Gradle*, etc.
- Integración con mayor cantidad de *frameworks* web
- Integración con interfaces de desarrollo integradas (IDEs)

También queda pendiente la generación de un sitio capaz de funcionar como repositorio de *dependencias* centralizado para que se constituya como la referencia en cuanto a paquetes para usar en la *capa de presentación*. El sitio debería permitir a los desarrolladores subir y manejar ellos mismos los paquetes que desean distribuir en el repositorio. Este es realmente uno de los trabajos pendientes más importantes para lograr una buena aceptación de la herramienta y posicionarla como un estándar en la industria.
El hecho de que exista una enorme cantidad de lenguajes para la *JVM*, y una enorme cantidad de *frameworks* especializados en el desarrollo de aplicaciones web que corren en los mismos, hace que sea extremadamente sencillo encontrar constantemente nuevas posibilidades a realizar. La adaptación a más lenguajes de forma *"nativa"*, y la integración con más procesos y *frameworks* es una de las formas de expansión de este trabajo más deseables.
Finalmente podemos comentar que si bien la plataforma elegida fue la *JVM*, existen numerosos lenguajes que no corren sobre esta y que se enfocan también en el desarrollo de sistemas web, como la plataforma *.NET* o *PHP*. Se podrían entonces crear adaptaciones (*ports*) a estos otros lenguajes y plataformas como una opción de trabajo a futuro.
Por último, es de suma importancia remarcar que la herramienta es liberada como *Software Libre* haciendo que el trabajo pendiente expresado en esta sección pueda ser realizado por la comunidad. Incluso es posible que existan desarrollos que expandan el sistema en formas que no han sido planteadas en esta sección.

Siglas

CDN Red de Distribución de Contenidos (Content Delivery Network)

CSS Hoja de Estilo en Cascada (Cascade Style Sheet)

DARPA Agencia de Proyectos de Investigación Avanzados de Defensa (Defense Advanced Research Projects Agency)

HTML Lenguaje de Marcas de Hipertexto (HyperText Markup Language)

HTML5 Lenguaje de Marcas de Hipertexto versión 5 (HyperText Markup Language version 5)

HTTP Lenguaje de Transferencia de Hipertexto (HyperText Transfer Language)

JVM Máquina Virtual de Java (Java Virtual Machine)

PC Computadora Personal (Personal Computer)

RIA Aplicacion de Internet Enriquecida (Rich Internet Application)

XML Lenguaje de Marcado Extensible (eXtensible Markup Language)

Glosario

CDN
Una red de distribución de contenido consiste en una serie de servidores en *Internet* distribuidos en distintos puntos geográficos. Los usuarios solicitan contenido a estos servidores y es siempre el servidor más próximo a la ubicación del usuario el que entrega el mismo, disminuyendo los tiempos de descarga.

CSS
CSS es un lenguaje utilizado para describir el aspecto y el formato de un documento escrito en lenguaje HTML o similar. Actualmente es un estandar web y es soportado por prácticamente todos los navegadores.

Fuentes web (Webfonts)
Las fuentes web consisten en tipografías capas de ser utilizadas en el navegador sin necesidad de ser instaladas en el equipo del usuario. Previo a la creación de esta tecnología, solamente era posible utilizar un numero limitado de fuentes comunes a todos+los sistemas operativos.

Git
Git es un software de control de versiones distribuido, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

GitHub
GitHub es un popular sitio web que permite almacenar código a sus usuarios. Los usuarios pueden entonces mantener su código actualizado y compartirlo online.

HTML
HTML, siglas de *HyperText Markup Language* (lenguaje de marcas de hipertexto), es un lenguaje de marcado para la elaboración de páginas web. Fue desarrollado por Tim Berners-Lee en 1991 y rápidamente se convirtió en un estándar.

HTML5
Se conoce como HTML5 a la quinta revisión importante del lenguaje básico de marcado de documentos para *Internet*. Su nombre suele hacer referencia no solo al nuevo estándar, todavía experimental de este lenguaje, sino también a las tecnologías que lo acompañan, CSS en su versión 3, y JavaScript.

HTTP
HTTP es el protocolo usado en cada transacción de Internet. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.

JAR
El JAR es una extensión de archivo utilizada por aplicaciones de la plataforma *Java*. Es un archivo comprimido que contiene en su interior código *bytecode Java* junto con metadata. Son, básicamente, programas que corren en la *JVM*..

JavaScript
JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado, orientado a objetos y basado en prototipos, utilizado principalmente del lado del cliente. Este lenguaje es un estándar web ya que es implementado como parte de prácticamente todos los navegadores web, permitiendo interactividad del usuario con la interfaz sin necesidad de comunicarse constantemente con el servidor.

JVM
La JVM es una parte fundamental de la plataforma del lenguaje Java, creada originalmente por *Sun Microsystems*. La maquina virtual crea una capa de abstracción entre el sistema operativo y el código Java compilado (*Bytecode*). De esta forma, el código Java puede ser compilado a *bytecode* una sola vez, y correrse en cualquier sistema que cuente con una JVM.

MVNRepo
MVNRepo o Maven Repository, es un sitio web alojado en la dirección *http://mvnrepository.com/* que permite buscar una amplia cantidad de paquetes *jar* o *war* que podemos importar en nuestro proyecto y nos provee del código necesario para agregarlo al mismo a través de distintos *manejador de dependencias*.

Node.js
Node.js es un entorno de programación en la capa del servidor basado en el lenguaje de programación *JavaScript*, con I/O de datos en una arquitectura orientada a eventos y basado en el motor *JavaScript* V8. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web.

Open Source
Código abierto es la expresión con la que se conoce al software distribuido y desarrollado libremente. Se focaliza más en los beneficios prácticos (acceso al código fuente) que en cuestiones éticas o de libertad que tanto se destacan en el software libre.

Parser
Un analizador sintáctico (o parser) es una de las partes de un compilador que transforma su entrada en un árbol de derivación.

RIA
Una aplicación de Internet enriquecida es una aplicación que corre en el navegador del usuario y que permite emular complejos comportamientos y funcionalidades que antes solo eran disponible en sistemas de escritorio, tales como arrastras y soltar, ordenar elementos o crear complejos gráficos. Para esto hacen uso de distintas tecnologías, que pueden ser, programas que corren sobre el navegador en forma de complementos, o aplicaciones creadas enteramente en HTML, CSS y JavaScript.

Software Libre
Software Libre es la denominación del software que respeta la libertad de todos los usuarios que adquirieron el producto y, por tanto, una vez obtenido el mismo puede ser usado, copiado, estudiado, modificado, y redistribuido libremente de varias formas.

StackOverflow
StackOverflow es un popular sitio sobre programación, en donde usuarios suelen realizar preguntas a problemas puntuales y otros usuarios las responden, transformando al sitio en una suerte de foro de intercambio de conocimientos sobre el tópico.

SVN
Subversion (SVN) es una herramienta de control de versiones open source basada en un repositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros.

XML
XML es un lenguaje de marcas utilizado para almacenar datos en forma legible. XML se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

Referencias

Alexander, Alvin (2013). *Scala Cookbook*. Sebastopol, CA, United States of America: O'Reilly Media (vid. pág. 50).

Allan, Roy A. (2001). *A history of the personal computer: the people and the technology*. London, Ontario, Canada: Allan Publishing. URL: http://www.retrocomputing.net/info/allan/ (vid. pág. 47).

Baarish, Greg (2002). *Building Scalable and High-performance Java Web Applications using J2EE Technology*. Indianapolis, Indiana: Pearson Education (vid. pág. 8).

Carbonelle, Pierre (2014). *PYPL PopularitY of Programming Language index*. URL: https://sites.google.com/site/pydatalog/pypl/PyPL-PopularitY-of-Programming-Language (vid. pág. 45).

CDN.js. *CDN.js: The missing CDN for JavaScript and CSS*. URL: http://cdnjs.com/ (vid. pág. 52).

David, Matthew (2013). *HTML5: Designing Rich Internet Applications*. New York; London: Focal Press (vid. pág. 48).

Duncan, Doris G. y Sateesh B. Lele (1996). "Converting From Mainframe to Client/Server at Telogy Inc." En: *Journal of Software: Evolution and Process* 8 (5). DOI: 10.1002/(sici)1096-908x(199609)8:5<321::aid-smr135>3.0.co;2-4. URL: http://libgen.org/scimag/index.php?s=10.1002/(sici)1096-908x(199609)8:5%3C321::aid-smr135%3E3.0.co;2-4 (vid. pág. 47).

Elliott, Margaret S. y Kenneth L. Kraemer (2008). *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing*. ASIS&T monograph series. Information Today Inc. 143 Old Merlton Pike, Medford, New Jersey: American Society for Information Science y Technology (vid. págs. 47, 49).

Franko, Greg (2013). *Dependency Management with RequireJS How-to*. 35 Livery Street, Birmingham, UK: Packt Publishing (vid. pág. 54).

Google. *Google Hosted Libraries*. URL: https://developers.google.com/speed/libraries/ (vid. pág. 52).

Hunter, Jason y William Crawford (2001). *Java Servlet Programming*. The Java Series. Sebastopol, CA, United States of America: O'Reilly Media (vid. pág. 48).

jsDelivr. *jsDelivr: A free super-fast CDN for developers and webmasters*. URL: http://www.jsdelivr.com/ (vid. pág. 52).

Kunst, Gerber (2014). *Programming Language Popularity Chart*. URL: http://langpop.corger.nl/ (vid. pág. 45).

Larman, Craig y Bas Vodde (2010). *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*. The Java Series. Boston, MA, United States of America: Pearson Education, Inc (vid. pág. 10).
Microsoft. *Microsoft Ajax Content Delivery Network*. URL: `http://www.asp.net/ajaxlibrary/cdn.ashx` (vid. pág. 52).
Montmollin, Gautier de (2013). *The Transparent Language Popularity Index*. URL: `http://lang-index.sourceforge.net/` (vid. pág. 45).
Software, TIOBE (2014). *TIOBE Index for June 2014*. URL: `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html` (vid. pág. 45).
Sonatype (2008). *Maven: The Definitive Guide*. Sebastopol, CA, United States of America: O'Reilly Media (vid. pág. 50).
Stephens, David (2008). *What On Earth is a Mainframe?* Longpela Expertise (vid. págs. 46, 47).
Twitter, Bower by (2014). *Bower: A package manager for the web*. URL: `http://bower.io/` (vid. pág. 54).
Wikipedia (2012). *List of Java Virtual Machines*. URL: `https://en.wikipedia.org/wiki/List_of_Java_virtual_machines` (vid. pág. 44).
— (2014). *List of JVM Languages*. URL: `http://en.wikipedia.org/wiki/List_of_JVM_languages` (vid. pág. 44).

Anexos

A. Elección de la plataforma

En esta sección se explican brevemente los factores que motivan la elección de la *JVM* como la plataforma de trabajo. En particular son dos los motivos por los cuales se ha elegido esta plataforma, su popularidad y la gran cantidad de lenguajes que corren sobre la misma.

A.1. Acerca de los lenguajes para la Java Virtual Machine (*JVM*)

Con la popularidad de *Java*, comenzaron a surgir una serie de lenguajes que compilan a *bytecode* para la Máquina Virtual de Java. Estos lenguajes se presentan como alternativas para los programadores de la plataforma que buscan una opción "superior", en algún aspecto, a *Java*. *Groovy*, por ejemplo, se presenta como un lenguaje dinámico, similar a Ruby; *Clojure* es una extensión de Lisp apuntado a la Programación Concurrente; *Scala* es un lenguaje que presenta características tanto de Programación Funcional como de Programación Orientada a Objectos. También existen otra numerosa cantidad de lenguajes, algunos experimentales, otros con una base creciente de usuarios, que corren sobre la plataforma. A esto se suma una amplia cantidad de adaptaciones de lenguajes populares para que corran en la *JVM*, como JRuby y Jython, versiones de Ruby y Python que compilan a *bytecode Java* [Wikipedia,2014].
Una característica interesante de estos lenguajes es que el código compilado suele ser compatible entre si, es decir, código escrito en *Java* puede ser ejecutado en *Scala*, código *Groovy* puede ser usado desde *Clojure*, etc.[5].
Finalmente, podemos destacar el hecho de que hoy en día existen numerosas implementaciones de la *JVM*, capaces de correr todos estos lenguajes en distintas plataformas [Wikipedia,2012].

[5] Si bien mayoritariamente el código de un lenguaje para la *JVM* es compatible con otros lenguajes de la *JVM*, ciertas características especiales de estos suelen no poder ser empleadas desde otros lenguajes, presentando así ciertas limitaciones de compatibilidad.

A.2. Popularidad de la *JVM* como plataforma

Las estadísticas de análisis de popularidad de lenguajes, como la renombrada estadística TIOBE, demuestran que *Java* es uno de los lenguajes más populares [TIOBE Software, 2014]. La estadística *"transparente"* creada por Gautier de Montmollin, que publica el código de forma *Open Source* y sus reglas online, muestra al lenguaje de Oracle en la misma posición [Montmollin,2013], mientras que la otra herramienta *Open Source* que permite medir la popularidad de los lenguajes, PyPL, de Pierre Carbonelle, lo muestra en primer lugar [Carbonelle,2014].
Además, Gerber Kunst ha desarrollado un gráfico en donde muestra la cantidad de líneas de código en GitHub por sobre las menciones en StackOverflow, mostrando a *Java* por sobre los más populares, pero también a los lenguajes que corren sobre la *JVM*[6] como de alto interés.

[6] Entre los lenguajes para la *JVM* se encuentran *Scala*, *Clojure*, *Groovy*, AspectJ, Ceylon, Fantom, Fortress, Frege, Gosu, Ioke, Jelly, Kotlin, Mirah, Processing, X10, Xtend, Rhino, JRuby, Jython, entre otros. Aunque no todos se encuentran mencionados en la gráfica.

B. Historia de los sistemas empresariales

Este apartado dará una breve muestra de las transformaciones que ha sufrido la industria de desarrollo de sistemas en los últimos tiempos. Se verá también como estas transformaciones han llevado a que los sistemas web sean hoy en día una de las opciones más utilizadas en el ámbito empresarial.

B.1. Estado anterior a los sistemas web

En las décadas de 1960 y 1970, comienza en el mundo un proceso lento pero incremental de computarización de la información. Las computadoras dejan de ser *"juguetes"* científicos para pasar a ser complejas maquinarias con verdadera utilidad practica en distintas industrias.
Durante esos años varias empresas comienzan a *informatizarse*. Bancos, petroleras y otros compran grandes computadoras capaces de almacenar toda su información o procesar complejos cálculos matemáticos en poco tiempo.
En este contexto, múltiples empresas como *IBM*, *RCA*, y *General Electric*, comenzaron a fabricar gigantescas computadoras de precio cada vez más económico. Estas maquinas evolucionarían poco a poco hasta convertirse en lo que hoy se conocen como *mainframes*.
En este sentido, David Stephens en su libro *What On Earth is a Mainframe?* [2008] describe a un *mainframe* como una computadora muy grande, con una base de datos de alto rendimiento y a la cual se accede desde una terminal remota. Básicamente, una maquina *"tonta"*, sin mucha otra finalidad más que la de conectarse al servidor central para realizar peticiones de datos y mostrarle los resultados al usuario.
Stephens también destaca los beneficios de los *mainframes* por sobre una computadora personal a nivel empresarial. Las características que menciona son:

- **Integridad de datos:** Los datos DEBEN ser correctos.
- **Rendimiento:** Se debe procesar gran cantidad de datos.
- **Respuesta:** El procesamiento debe ser inmediato.
- **Recuperación ante desastres:** En caso de un fallo, se debe volver a estar operativo inmediatamente.
- **Usabilidad:** Debe hacer lo que se requiere, cuando se requiere.
- **Confiabilidad:** No debe fallar y siempre debe estar disponible.
- **Auditoria:** Ser capaz de saber quien realizó qué acción en el equipo.

- **Seguridad:** Solo aquellos que pueden realizar una acción pueden hacerlo.

Estas características son todavía buscadas en los grandes *mainframes* de la actualidad, y resultan fundamentales en emprendimientos críticos.
Margaret S. Elliott y Kenneth L. Kraemer ha descripto que las tendencias de la tecnología utilizada, no obedecen solamente a motivaciones económicas y operativas, sino a un complejo entramado de relaciones entre empresas que comparten una visión utópica sobre la tecnología en cuestión [Elliott y Kraemer,2008, pag 3]. Así, las empresas comercializadoras de *mainframes* enarbolando como bandera principal los beneficios planteados por Stephens, lograron generar una visión utópica en las empresas usuarias para posicionar la tecnología dominante durante largos años. Es por esto que los *mainframes* siguen teniendo hoy en día un lugar en el mercado.

B.2. Aparición de las computadoras personales

A partir de fines de los 70's y comienzos de los 80's, las PCs comienzan a ser cada vez más accesibles [Allan,2001, cap 4]. La liberación de *ARPANET* por parte de DARPA en 1983 da nacimiento a *Internet*. Estos dos cambios suponen un quiebre en las tecnologías de uso empresarial.
El PC, sumado a la conectividad y nuevas tecnologías tanto en sistemas como en lenguajes de programación llevaron a que los *mainframes* comenzaran a perder terreno.
La posibilidad de ejecutar programas en cada PC dio lugar a nuevas arquitecturas *cliente-servidor*. Ahora los usuarios corrían los programas en su propio equipo, permitiendo aumentar la velocidad de respuesta. El programa se conectaba a *Internet* y sincronizaba información con un servidor central solo cuando fuera necesario.
Esta arquitectura permitiría una más rápida evolución del software por sobre la opción de tener el sistema en un *mainframe*, como explican Doris G. Duncan y Sateesh B. Lele en su artículo "Converting From Mainframe to Client/Server at Telogy Inc.". Como contrapartida estos cambios suponen un alto costo de mantenimiento, y en caso de poseer equipos con distintos sistemas operativos, un gasto extra en el desarrollo. Independientemente de los costos, el factor más incidente en el cambio es la necesidad de mantener el sistema al día con las necesidades de la empresa, señalan Duncan y Lele.

B.3. Navegadores y primeros sistemas web

Tras la aparición de los navegadores web (como *Netscape* e *Internet Explorer*) a comienzos de 1990 se comenzaron a desarrollar los primeros sistemas web.
Los mismos consistían básicamente en documentos dinámicos, generados con información tomada de una base de datos. Gran influencia tuvieron en este tipo de soluciones la creación de lenguajes de programación y tecnologías pensadas para generar documentos *HTML* dinámicamente, como *PHP*, *ASP* y los *Servlets* de *Java* [Hunter y Crawford,2001, pag 2]. La posibilidad de generar *aplicaciones* que corran en el navegador solucionó las problemáticas asociadas a el mantenimiento de los sistemas y al desarrollo para múltiples sistemas operativos. Sin embargo, la naturaleza de *HTML* limitaba la capacidad de las aplicaciones a sencillos formularios, botones y enlaces.
Estas limitaciones hicieron que no fuera sino hasta con la aparición de las primeras tecnologías complementarias a *HTML* que las aplicaciones web comenzarían a cobrar relevancia, dando lugar a las RIA.

B.4. Rich Internet Applications

Las RIAs surgen para compensar aquellas faltantes que presentaban las aplicaciones web frente a las tradicionales de *cliente-servidor*.
En un primer lugar software en forma de complementos (*plugins*) que permitían correr algún lenguaje especial en los navegadores fueron creados.
Las *RIA* comenzaron a copar los mercados empresariales, ya que permitían ahorrar en mantenimiento y desarrollo a la vez que brindaban la usabilidad que se demandaba de un sistema empresarial.
Cada vez más soluciones comenzaron a surgir y los *plugins* privativos comenzaron a ser un problema para los desarrolladores. Esto llevaría a que se desarrollaran con fuerza *CSS* y *JavaScript*, tecnologías libres que permitían crear *RIAs*, disponibles en cualquier navegador sin necesidad de instalar software adicional.
Así, en los últimos años, y gracias también al auge de los smartphones y tablets, incapaces de correr los anteriormente mencionados *plugins*, el llamado *HTML5* pasaría a transformarse en el estándar para el desarrollo de *RIAs* [David,2013].

B.5. Conclusiones

Los sistemas web actuales permiten la robustez esperada de los primeros sis-

temas *mainframe*, pero sin los altos costos de adquisición y mantenimiento de los mismos. A su vez, las *RIAs* ofrecen la capacidad de tener aplicaciones con un alto grado de usabilidad. Además, la gran cantidad de *frameworks* capaces de generar rápidamente sistemas web con una interfaz completamente realizada en *HTML5* que han aparecido en los últimos años, permiten reducir los tiempos de desarrollo. Junto con las plataformas como servicio, como *Heroku* o *Rackspace* reducen significativamente la puesta en producción de los sistemas.
Todas estas características han llevado a que las aplicaciones web con una *capa de presentación* hecha en *HTML5* se hayan transformado en la nueva visión utopica señalada por Elliott y Kraemer.Así, este tipo de desarrollos seguirán siendo la norma durante los años venideros.

C. Manejo de distintos tipos de *dependencias*

En esta sección se explicará como se manejan las *dependencias* en las distintas capas de software. Se mostrará como la solución propuesta en este documento se adapta a técnicas ya probadas.

C.1. Dependencias manejadas de la *capa de lógica de negocios*

En el ámbito del manejo de *dependencias* en la *capa de lógica de negocios* la herramienta *Maven* de la fundación *Apache* se ha posicionado como el estándar de facto[7] [Sonatype,2008]. Otro de los proyectos que se han posicionado en esta área es *Apache Ivy*, que se concentra solamente en el manejo de *dependencias*.
Maven hace uso de un *archivo de configuración* escrito en *XML*, archivo comúnmente denominado POM, ya que el nombre del archivo es pom.xml. El código 7 muestra un ejemplo de este tipo de *archivo de configuración*. *Ivy* utiliza también archivos *XML*, pero estos son más compactos que los de *Maven*. El código 8 es un ejemplo del mismo.

Con la aparición de nuevos lenguajes para la *JVM* comenzaron a surgir nuevos *manejadores de dependencias*, como Grape o Gradle para *Groovy*, *SBT* para *Scala*, Leiningen para *Clojure*, y *Apache* Buildr como propuesta multilenguaje. Cada uno de ellos utiliza un *archivo de configuración* con distinta sintaxis [Alexander,2013, pág. 569]. El código 9 muestra un ejemplo de la declaración de una *dependencia* en el *archivo de configuración* de *SBT*.

A pesar de la gran cantidad de formatos de *archivo de configuración*, todos los *manejadores de dependencias* trabajan bajo el estándar pre-impuesto por *Maven*. Esto llega a un punto tal que el sitio MVNRepo es el centro de referencia al momento de buscar dependencias. Se puede entonces apreciar como todos los *archivos de configuración* especifican inequívocamente un paquete de código mediante tres campos clave para su búsqueda: el nombre de

[7] Si bien *Apache Maven* es en realidad no solo un *manejador de dependencias* sino un administrador de proyectos, provee la funcionalidad antes mencionada y la misma se ha vuelto un estándar.

```
<project>
  ...
  <dependencies>
    <dependency>
      <groupId>group-a</groupId>
      <artifactId>artifact-a</artifactId>
      <version>1.0</version>
    </dependency>
    <dependency>
      <groupId>group-a</groupId>
      <artifactId>artifact-b</artifactId>
      <version>1.0</version>
    </dependency>
  </dependencies>
</project>
```

Código 7: Configuración de *dependencias* en *Maven* mediante archivo POM

```
<ivy-module version="2.0">
  <info organisation="org.apache" module="hello-ivy"/>
  <dependencies>
    <dependency org="group-a" name="artifact-a" rev="1.0"/>
    <dependency org="group-b" name="artifact-b" rev="1.0"/>
  </dependencies>
</ivy-module>
```

Código 8: Archivo de configuración de Ivy

la compañía, el nombre del modulo y la versión del programa.

C.2. Dependencias no manejadas de *capa de presentación*

La mayoría de las *dependencias* de la *capa de presentación* consisten en archivos principalmente *CSS* y *JavaScript* (aunque también podrían ser fuentes web u otros recursos), los cuales son descargados manualmente a través de la página del creador del código (No existen repositorios centralizados como en el caso de las *dependencias* de la *capa de lógica de negocios*). Una vez

```
libraryDependencies += "org.springframework.data" %
    "spring-data-neo4j" % "3.1.1.RELEASE"
```

Código 9: Dependencia de SpringFramework 3.1.1 para *SBT*

descargados, el desarrollador debe mover los archivos a alguna carpeta de su proyecto para *instalarlo* (la ubicación exacta depende de la estructura del proyecto y por tanto debe ser recordada por el programador) para encontrarse finalmente disponibles para su uso en el código del usuario.
Los errores en esta etapa son muy comunes dado la variada cantidad de tipos de *dependencia*, donde cada uno debe ser copiado en una ubicación puntual. Además una misma *dependencia* no necesariamente consiste en un único archivo (como sucede en la *capa de lógica de negocios*), sino en múltiples archivos de distinto formato que deben ser agregados al proyecto de una forma especifica. La falta de un lugar que agrupe estos contenidos y estandarice sus nombres, versiones, formas de instalación, etc. hace que sea muy difícil manejar este tipo de *dependencias*.

C.3. CDN

Las *Redes de Distribución de Contenido*, (*CDN*) surgen con la idea de agrupar el contenido disponible online en un solo lugar[8].
Así, las *CDN* permiten al usuario insertar en el código *HTML* de un proyecto, una clausula que importa automáticamente una dependencia desde el servidor de la *CDN* más cercano.

El código 10 utiliza la *CDN* de Google (`https://developers.google.com/speed/libraries/`) una de las más populares y confiables junto con la de Microsoft (`http://www.asp.net/ajaxlibrary/cdn.ashx`) y los sitios `http://www.jsdelivr.com/` y `http://cdnjs.com/`.
Las *CDNs* permiten adicionalmente aumentar la velocidad de la aplicación que se desarrolla ya que los archivos de *dependencia* pueden ser descargados desde servidores más cercanos a la ubicación del usuario, y además ahorran carga de trabajo al servidor de la aplicación.
Como contrapartida, las *CDNs* suelen tener una gestión centralizada desde alguna entidad proveedora, la cual no necesariamente responde a los pedidos

[8] Su creación no responde necesariamente a procesos de manejo de dependencias en proyectos de desarrollo de software, sin embargo es uno de los usos posibles de este tipo de redes y ha sido uno de los usos más comunes.

```
<html>
  <head>
    ...
    <!-- JQuery -->
    <script
  src="//ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js">
    </script>
    <!-- JQuery Mobile -->
    <link rel="stylesheet"
  href="//ajax.googleapis.com/ajax/libs/jquerymobile/1.4.3/jquery.mobile.min.css
    <script
  src="//ajax.googleapis.com/ajax/libs/jquerymobile/1.4.3/jquery.mobile.min.js">
    </script>
    <!-- JQuery UI -->
    <link rel="stylesheet"
  href="//ajax.googleapis.com/ajax/libs/jqueryui/1.11.0/themes/smoothness/jquery
    <script
  src="//ajax.googleapis.com/ajax/libs/jqueryui/1.11.0/jquery-ui.min.js">
    </script>
  </head>
  <body>
    ...
  </body>
</html>
```

Código 10: Dependencias agregadas mediante *CDN*

o requerimientos del usuario). Esto hace que sea difícil contar con las ultimas versiones, o con bibliotecas que los administradores del mismo decidan no soportar. A esto se le agrega el alto costo de mantener una *CDN* por cualquier empresa que no posea grandes servidores a lo largo del globo.

C.4. Dependencias manejadas de *capa de presentación*

Otra forma de manejar *dependencias* en la *capa de presentación* consiste en utilizar el mismo mecanismo de *manejadores de dependencias* que se utiliza en la *capa de lógica de negocios*. Esto pone solución a las problemáticas que acarrean las *CDNs*.
Bower y *Component* son los *manejadores de dependencias* más populares para la *capa de presentación*. Ambos son módulos de la plataforma Node.js

y manejan las *dependencias* mediante un *archivo de configuración*.
Estas herramientas no se empaquetan de forma auto contenida, sino que requieren contar con herramientas externas como *Git* o *SVN* en la maquina en donde se va a utilizar [Twitter,2014]. Más aún, los usuarios de lenguajes que corren sobre la plataforma *Java* muchas veces carecen de acceso a la maquina física, por fuera de la *JVM*, haciendo imposible utilizar estas opciones.
Adicionalmente, existen soluciones que se enfocan en un tipo especifico de dependencia, como *RequireJS* y *Browserify* que permiten manejar solamente archivos *JavaScript*, sin utilizar archivos *archivos de configuración*, sino mediante una línea de código en el programa del usuario [Franko,2013].
Todas estas soluciones se presentan bastante incompletas en comparación a las disponibles en la *capa de presentación*. Además, al requerir herramientas que no son independientes del sistema operativo, generan muchas complicaciones al momento de configurar el entorno de desarrollo o de puesta en producción del sistema. Finalmente, fuerzan al desarrollador a aprender una nueva tecnología, proceso que lleva tiempo y esfuerzo.

C.5. Funcionamiento de Fronttier

La herramienta propuesta funciona mediante un *archivo de configuración* de forma similar a las opciones disponibles para la *capa de lógica de negocios*. Además, los *archivo de configuración* que utiliza la herramienta, pueden poseer formatos distintos, adaptándose a los formatos que los desarrolladores ya conocen de otras herramientas.
A diferencia de las opciones de manejo de dependencias que funcionan en la *capa de presentación* de esta forma, la herramienta propuesta no requiere herramientas externas, y todo corre sobre la *JVM*.