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

julio 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.

Por sobre todo quiero dar las gracias a Gabriela Guibaud sin cuya ayuda me habría sido imposible terminar este trabajo.

Índice

Resumen

Al momento de desarrollar sistemas web utilizando lenguajes que corren en la *JVM*, es a menudo necesario descargar una serie de bibliotecas y archivos desde diversos sitios de *Internet* para agregarlos a nuestro proyecto. Estos archivos son nuestras *dependencias* para la *capa de presentación*, la parte visible de nuestro sitio. Esta tarea suele llevar tiempo, ser tediosa y propensa a errores. Actualmente no hay herramientas que corran sobre la *JVM* 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 todos estos archivos, organizarlos en nuestro código y dejarlos listos para su uso.

1. Introducción

En la presente sección se intentará explicar brevemente el contexto actual del desarrollo de software y las motivaciones del presente trabajo, así como del lenguaje de programación elegido para el desarrollo.
Se comenzará por un análisis sobre la evolución de las tecnologías actuales y como estas han derivado a la generación de aplicaciones hechas mayormente en *HTML5*. Luego se enumeraran los distintos factores que inciden sobre la elección del lenguaje de programación a utilizar, mostrando como los lenguajes para la *JVM* son una buena elección debido a su popularidad y a la compatibilidad de código entre lenguajes que corren sobre la plataforma. Finalmente se explicará la estructura básica de las aplicaciones web desarrolladas sobre lenguajes de la *JVM*, y como se manejan las *dependencias* en estas aplicaciones.

1.1. Sobre los sistemas web

Si bien no es el objetivo de este documento explicar en detalle la historia de los cambios en las arquitecturas de software, este apartado intentará dar una breve muestra de las transformaciones que ha sufrido la industria de desarrollo de sistemas en los últimos tiempos, y como estas han llevado a que los sistemas web sean hoy en día una de las opciones más utilizadas en el ámbito empresarial.

1.1.1. 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, a la cual se accede desde una terminal remota, es decir, 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, por ejemplo, sistemas bancarios, programas de control de acciones financieras, sistemas petroleros o el software que controla misiones espaciales.

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.

1.1.2. Aparición de las computadoras personales

A partir de fines de los 70's y comienzos de los 80's, las Computadoras Personales (Personal Computer, 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.". Sin embargo, 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. Esto se debe a la necesidad de generar un cliente para cada sistema operativo, y la necesidad de actualizar los mismos en caso de futuros cambios[1]. Así, los autores demuestran que, 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.

[1] Recuerde el lector que el software se distribuía en esa época en medios físicos, por lo que la instalación de una nueva versión del sistema requería presencia física en el equipo.

1.1.3. 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 [Hunter y Crawford, 2001, pag 2], como *PHP*, *ASP* y los *Servlets* de *Java*. 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, esta vez, 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 Aplicacion de Internet Enriquecida (Rich Internet Application, RIA).

1.1.4. Rich Internet Applications

Las RIAs surgen para compensar aquellas faltantes que presentaban las aplicaciones web frente a las tradicionales de *cliente-servidor* de escritorio.
En un primer lugar, software en forma de complementos (*plugins*) que permitían correr algún lenguaje especial en los navegadores fueron creados. Uno de los más populares fue *Flash* de *Adobe*, que continua siendo utilizado hoy en día. *Java* tampoco se quedo atrás y presentó sus *Servlets*, una forma de embeber código Java en páginas *HTML*.
Las *RIA* comenzaron a copar los mercados empresariales, ya que permitían ahorrar en mantenimiento y desarrollo, a la vez que brindaban la alta usabilidad que se demandaba de un sistema empresarial.
Cada vez más soluciones comenzaron a surgir, *Adobe Flex*, *Microsoft Silverlight* y *JavaFX* [Clarke, Connors y Bruno, 2009]. Sin embargo, la falta de estándares llevaría a que se desarrollaran con fuerza CSS y JavaScript, tecnologías libres y 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].
La tendencia marca que las *RIAs* desarrolladas bajo el estándar *HTML5*

seguirán siendo la norma durante los años venideros. Múltiples *frameworks* han aparecido capaces de generar rápidamente sistemas web con una interfaz completamente realizado en *HTML5*.

1.2. Acerca de 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, algunos con una base creciente de usuarios, que corren sobre la plataforma, como también 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. 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*, codigo *Groovy* puede ser usado desde *Clojure*, etc.[2].
Finalmente, podemos destacar el hecho de que hoy en día existen numerosas implementaciones de la *JVM*, tanto propietarias como libres, muchas con fines muy específicos, como Dalvik, la *JVM* del sistema operativo Android [Wikipedia, 2012].

[2] Si bien mayoritariamente esto es cierto, ciertas caracteristicas especiales de algunos lenguajes no pueden ser empleadas desde otros, presentando así ciertas limitaciones de compatibilidad. Sin embargo, se espera que futuras versiones de la *JVM* permitan un grado de compartición de código entre lenguajes aún mayor.

1.3. Crecimiento de la *JVM* como plataforma

Java es, casi sin cuestionamientos [3], uno de los lenguajes más populares. Así lo demuestran las estadísticas de análisis de popularidad de lenguajes, como la renombrada estadística TIOBE que a la fecha de esta publicación ubica a *Java* en segundo lugar [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*[4] como de alto interés.
Queda entonces patente que los lenguajes que se ejecutan en la *JVM* tienen un uso cada vez mayor. Empero, la diferencia se volvería aún más grande si se considera el uso de los lenguajes para desarrollo de soluciones web.
Si bien no existen estadísticas que confirmen esto, es prácticamente reconocido por cualquier programador que la ventaja que se lee en las estadísticas por parte del lenguaje C, es producto del desarrollo, no de sistemas web, sino de pequeñas herramientas de escritorio. Asimismo, la popularidad creciente de Objective-C, se debe al uso del mismo en las aplicaciones móviles de iPhone y iPad. Finalmente, el amplio desarrollo que presenta en las encuestas *JavaScript*, no compite necesariamente con los sistemas hechos en *JVM* sino que de hecho, tiende a complementarlos, ya que es una parte importante de las *RIAs* [5].

[3] Se puede argumentar que la medición de la popularidad de los lenguajes de programación es una ciencia inexacta, pues los factores de medición no se encuentran determinados. Sumado a esto, la mayoría de los resultados se basa en cantidad de lineas de código y menciones online, hecho que no tiene en consideración las diferencias entre un lenguaje y otro, ya que algunos son necesariamente más verborragicos que otros (Por ejemplo, dos códigos que realizan exactamente la misma acción en dos lenguajes distintos pueden resultar en cantidades significativamente distintas de lineas de código entre uno y otro, solo por la sintaxis y la naturaleza misma del lenguaje.) Así, cada estadista toma distintas variables, le otorga distinto peso a las mismas y obtendrá distintos resultados.

[4] 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.

[5] ver sección 1.1.4.

1.4. Desarrollo de sistemas web en la *JVM*

Al desarrollar sistemas web se puede elegir cualquiera de los lenguajes para la *JVM*, y dentro del mismo, cualquiera de los numerosos *frameworks* que trabajan con *HTML5* existentes. Sin embargo hay ciertas constantes comunes de todos los lenguajes y *frameworks* antes mencionados. A continuación se analizaran estos componentes comunes para comprender la forma de desarrollar en los mismos.

1.4.1. Estructura común de sistemas web

La mayoría de los sistemas web suelen dividirse en capas (También llamadas *layers* o *tiers*). Estas capas suelen dividirse 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. Sin embargo, 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.
En los últimos años, el crecimiento de las tecnologías móviles han hecho que la arquitectura cambie a una más orientada a servicios, donde el servidor no genera documentos que serán procesados por el usuario, sino que este, consumirá solamente datos expuestos por el servidor, encargándose el cliente de la parte de la interfaz. De esta forma, un mismo sistema web puede ser utilizado para mostrar una página en un navegador, o por una aplicación para teléfonos móviles[MacVittie, 2008]. La figura 2 muestra la arquitectura con este nuevo paradigma.

1.4.2. Reutilización de código mediante dependencias

Prácticamente ningún desarrollador comienza una aplicación completamente desde cero, haciendo a la conocida frase *"Pararse sobre hombros de gigantes"* una constante en el desarrollo de software, en especial en el empresarial. Así, existen numerosos *frameworks*, *toolkits*, bibliotecas, y porciones de código que los desarrolladores pueden importar en el propio, de forma de

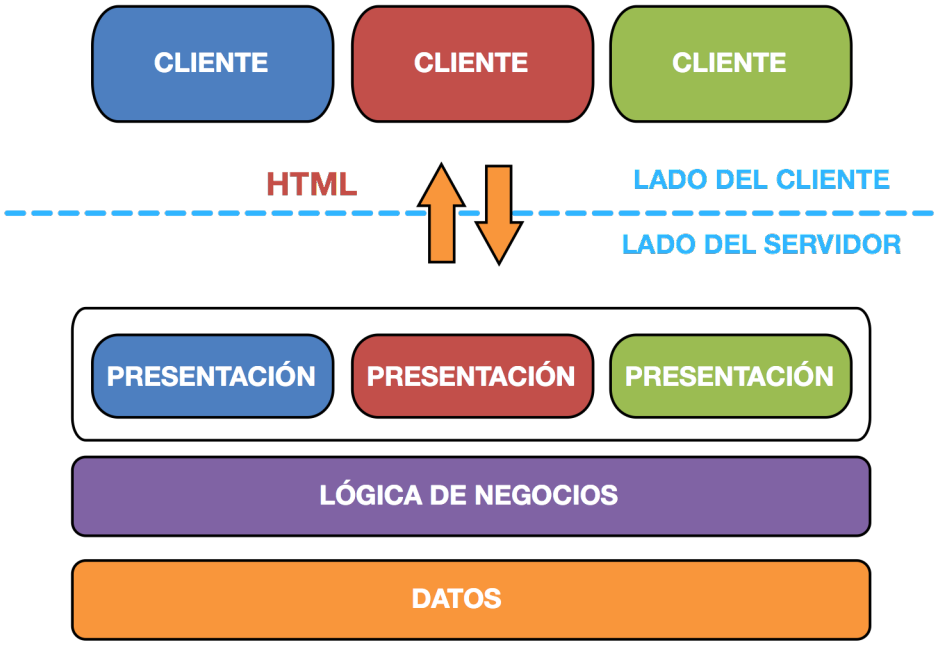

Figura 1: Arquitectura em capas básica de un sistema web.

obtener funcionalidades genéricas previamente desarrolladas.
Estas porciones de código de terceros que utilizan los desarrolladores en sus proyectos recibe el nombre de *dependencias*. Las mismas son necesarias para que un proyecto funcione correctamente.
Se pueden distinguir dos tipos de *dependencias*, las manejadas y las 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 manejadas, consisten en algún tipo de descripción en un archivo de configuración o símil que identificará a las *dependencias* requeridas. Un programa externo evaluara nuestro archivo de configuración y se encargará de instalar nuestras *dependencias* declaradas, estos programas son conocidos como *manejadores de dependencias*.
Las *dependencias* no manejadas suelen implicar 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 ahorrar el trabajo

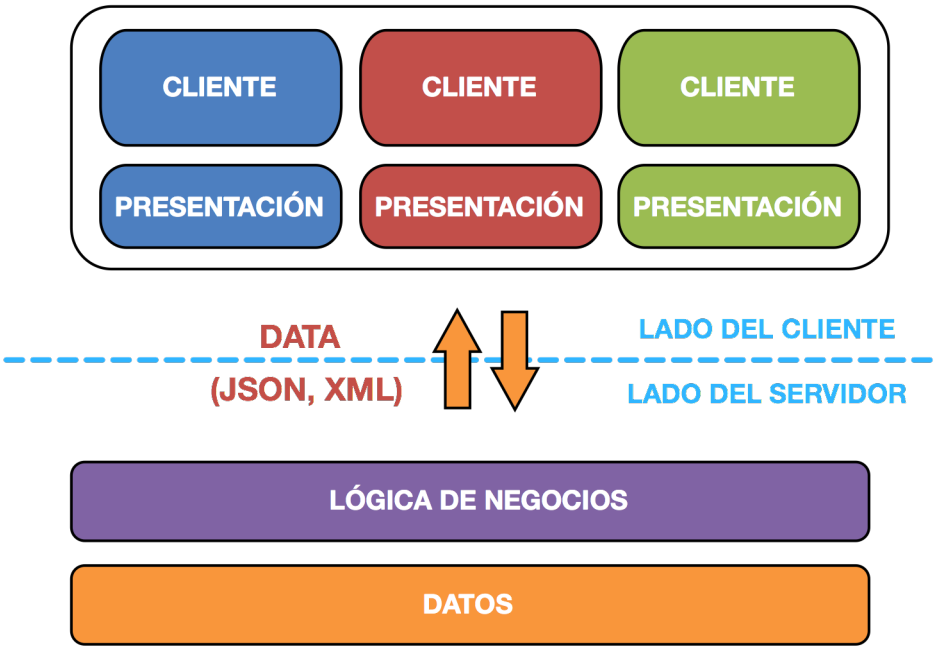

Figura 2: Arquitectura em capas de los nuevos sistemas web.

que implica instalar las *dependencias* anidadas[6] [Larman y Vodde, 2010].
Finalmente, en este escrito, se identificaran dos tipos más 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*.
En las subsecciones siguientes se verá como estas dependencias son abordadas por los desarrolladores que trabajan en lenguajes de la *JVM* en la actualidad.

1.4.3. Dependencias manejadas de la *capa de lógica de negocios*

El desarrollo en la plataforma *Java* se volvió muy popular rápidamente, y por tanto, un gran numero de herramientas surgieron para asistir al desarrollador al momento de programar.
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 posi-

[6] 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.

[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

```
<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 1: 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 2: Archivo de configuración de Ivy

cionado 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[8]. El código 1 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 **??** 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 y Leiningen para Clojure, y Apache Buildr como propuesta multilenguaje. Cada uno de ellos utilizan un archivo de configuración con distinta sintaxis [Alexander, 2013, pág. 569]. El código 3 muestra un ejemplo

vuelto un estándar.

[8] Este archivo es comunmente denominado POM, ya que el nombre del archivo debe ser pom.xml.

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

Código 3: Dependencia de SpringFramework 3.1.1 para *SBT*

de la declaración de una *dependencia* en el archivo de configuración de *SBT*.

A pesar de este hecho, todos los *manejadores de dependencias* trabajan bajo el estándar pre-impuesto por Maven, siendo así al punto tal de que el sitio MVNRepo es el centro de referencia al momento de buscar dependencias.

1.4.4. Dependencias no manejadas de *capa de presentación*

Lamentablemente, la falta de estándares para desarrollar *RIAs* en la *capa de presentación* durante largos años llevó a que no hayan herramientas sencillas para instalar *dependencias* que corran en la misma. La mayoría de estas *dependencias* consisten en archivos *CSS* y *JavaScript*, los cuales suelen ser 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*). Luego, los archivos deben ser copiados a alguna carpeta del proyecto (la ubicación exacta depende de la estructura del proyecto y por tanto debe ser recordada por el programador) para encontrarse finalmente disponibles para su importación y uso en el código del usuario.
Si bien el proceso parece sencillo, las aplicaciones web se han vuelto cada vez más y más complejas. Las posibilidades que brindan los navegadores aumentan y, por tanto, también aumentan las *dependencias* ya que se requieren cada vez cosas más especificas[9].
A su vez, los tipos de *dependencia* son más variados (No son solo archivos jar o war), dando por resultado un catálogo mucho más amplio, donde una misma *dependencia* no necesariamente consiste en un único archivo, 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 estas *dependencias*.

[9] Un claro ejemplo de esto son las Fuentes web (Webfonts). Las mismas no comenzaron a ser comunes sino hasta hace unos pocos años, y hoy en día son usadas en enorme cantidad de sitios.

```
<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-ui.css" />
    <script src="//ajax.googleapis.com/ajax/libs/jqueryui/1.11.0/
        jquery-ui.min.js"></script>
  </head>
  <body>
    ...
  </body>
</html>
```

Código 4: Dependencias agregadas mediante *CDN*

1.4.5. CDN

Las *Red de Distribución de Contenidos (Content Delivery Network, CDN)* surgen con la idea de agrupar el contenido disponible[10].
Las CDN permiten al usuario insertar en su codigo HTML una clausula que importa automáticamente la dependencia requerida desde el servidor más cercano.

El código 4 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* no solo brindan el beneficio de facilitar el manejo de dependencias, sino que a su vez le ahorra trabajo a los servidores de nuestra aplicación, que no requiere servir algunos archivos, y puede, en teoría, ser beneficioso para los usuarios, ya que el servidor de la *CDN* podría estar más cerca de su área que el de la aplicación, facilitando la carga del sitio.

[10] 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.

Como contrapartida, las *CDNs* suelen tener una gestión centralizada, con lo cual es difícil contar con las ultimas versiones, o con bibliotecas que los administradores del mismo decidan no colocar. Así, muchas bibliotecas y *frameworks* no se encuentran disponibles en las *CDNs* más populares. 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.

1.4.6. Dependencias manejadas de *capa de presentación*

Surgió entonces la necesidad de contar con *manejadores de dependencias* para la *capa de presentación* y que no requiera de la disponibilidad de terceros como las *CDNs*. Así surgieron soluciones que manejan las *dependencias* de forma similar a la forma en que lo hacen los manejadores de la *capa de lógica de negocios*.
Bower y *Component* [11], son casi con seguridad, los más populares de estos sistemas. Ambos son modulo de la plataforma Node.js y poseen formas de manejar las *dependencias* mediante un archivo de configuración. Sin embargo, ambas requieren en la mayoría de los casos contar con herramientas externas como git, ya que se desentienden del proceso de mantenimiento de repositorios Twitter, 2014.
RequireJS y *Browserify* son otras soluciones que manejan las dependencias *JavaScript* solamente y no utilizan la técnica de archivos de configuración [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. Como si esto no fuera suficiente, el desarrollador debe aprender una nueva tecnología, proceso que lleva tiempo y esfuerzo del que a veces no dispone.

2. Solución Propuesta

De analizar el apartado anterior se puede percibir que sería de amplio interés para los desarrolladores contar con una herramienta que corra sobre la *JVM* y que permita manejar las *dependencias* de la *capa de presentación*. Este trabajo propone la generación de una solución a este problema desarrollada en *Scala* y usando *SBT* para manejar las *dependencias* en el mismo.

[11] Component es más similar a Maven ya que se encarga de muchas otras cosas además del manejo de dependencias, como procesos de compilación.

2.1. Características deseadas

Se tomarán como requerimientos una serie de características que a los efectos parecen deseables en una herramienta que intenta posicionarse como el estándar para manejo de *dependencias* de la *capa de presentación* en lenguajes que corren sobre la plataforma. 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 lo cual, es necesario que se encuentre desarrollada en un lenguaje que compile a *bytecode Java*. Adicionalmente deberá poder ser utilizada desde cualquier otro lenguaje de la *JVM* de forma natural, es decir, que 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.
La herramienta deberá ser capaz de descargar y acomodar los paquetes que el usuario determine como *dependencias* en un archivo de configuración. Como se ha mencionado en la sección 1.4.6, distintas herramientas utilizan distintos tipos de archivos de configuración. En lugar de forzar un formato especifico para el mismo, el *manejador de dependencias* desarrollado deberá brindar al usuario la posibilidad de usar un formato para el mismo que se adecue a sus necesidades[12].
Al existir soluciones como *Bower* o *Component*[13], la herramienta debería complementarlas, permitiendo por ejemplo utilizar sus archivos de configuración y sus repositorios.
Finalmente, sería ideal contar con un repositorio propio, 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[14]. Algunos de estos repositorios podrían ser privados, y por tanto, se

[12] 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.

[13] Estas soluciones no cumplen todos los requerimientos deseados, pero si son utilizadas con éxito en proyectos de desarrollo.

[14] Muchas empresas de software suelen tener sus propios repositorios para garantizar disponibilidad de sus requerimientos o porque contienen código propietario usado internamente.

requeriría acceso mediante un usuario y contraseña que la herramienta debe enviar de forma correcta y segura.
Finalmente, podemos destacar el hecho de que, actualmente, muchas *dependencias* de la *capa de presentación* consisten en múltiples archivos. La mayoría de los *manejadores de dependencias* actuales se basan en archivos, haciendo que se requiera una declaración en el archivo de configuración por cada archivo de la dependencia, algo que no solo es poco práctico, sino propenso a errores ya que el olvido de una declaración invalida completamente la dependencia.
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[15].
En forma de resumen y para mayor claridad, se enumeran a continuación todas las características mencionadas:

- 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*

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 ha determinado que la tecnología a usar será *Scala*. La elección se basa en la facilidad que presenta este lenguaje para generar código compatible con otros de la *JVM* y la gran cantidad de características que presenta, las cuales permiten generar gran funcionalidad en pocas lineas de código. Además, *Scala* soporta características de análisis sintáctico (*parsing*) sencillas de usar

[15] 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.

desde el lenguaje, que resultan muy útiles cuando se requiere leer archivos de configuración. *SBT* permite automatizar las tareas de manejo de dependencias de forma rápida y sencilla, y es el *manejador de dependencias* de facto en *Scala*.
En algunos casos, tales como en el desarrollo en herramientas de integración con tecnologías existentes, es posible que sea necesario el uso de otros lenguajes, como *Java*, *Clojure* o *Groovy*.

2.3. Alcance del presente trabajo

El presente trabajo no pretende ser la solución completa a los requerimientos planteados, los cuales son muy amplios y llevarían demasiado tiempo para abordar por una sola persona. En cambio se enfocará en las características más importantes, dejando asentadas las bases para el desarrollo de las partes no concluidas. Así el enfoque será puesto en la estructura del sistema, orientada siempre a la extensibilidad.
Se presenta como objetivo el desarrollo de al menos las siguientes características de forma completamente funcionales:

- 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 Maven
- Lectura de archivos de configuración en XML estilo 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 Maven
- Integración de la herramienta con *SBT*

- Integración de la herramienta con Play! Framework

Si bien quedarán pendientes muchas tareas para realizar, estas serán abordadas en futuros trabajos. Además, 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.

3. Desarrollo

Explicación del Modelo Explicación de Código Ejemplos de uso como linea de comandos Ejemplos de uso con archivo de configuración Instalación de la herramienta como plugins Uso del plugin

4. Conclusiones

Que se gana, que se pierde Que se necesita Que cosas no se lograron

5. Trabajo a futuro

Que más se puede hacer???

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)

JVM Máquina Virtual de Java (Java Virtual Machine)

PC Computadora Personal (Personal Computer)

RIA Aplicacion de Internet Enriquecida (Rich Internet Application)

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, p. 17

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, p. 9

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, p. 16

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, p. 11

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, p. 9

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, p. 9

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, p. 9

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*, p. 16

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, p. 18

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, p. 9

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, p. 11

Referencias

Alexander, Alvin (ago. de 2013). *Scala Cookbook*. Sebastopol, CA, United States of America: O'Reilly Media (vid. pág. 15).

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. 8).

Baarish, Greg (2002). *Building Scalable and High-performance Java Web Applications using J2EE Technology*. Indianapolis, Indiana: Pearson Education (vid. pág. 12).

Carbonelle, Pierre (jun. de 2014). *PYPL PopularitY of Programming Language index*. URL: https://sites.google.com/site/pydatalog/pypl/PyPL-PopularitY-of-Programming-Language (vid. pág. 11).

CDN.js (s.f.). *CDN.js: The missing CDN for JavaScript and CSS*. URL: http://cdnjs.com/ (vid. pág. 17).

Clarke, Jim, Jim Connors y Eric J. Bruno (2009). *JavaFX: Developing Rich Internet Applications*. Santa Clara, California: Addison Wesley Proffessional (vid. pág. 9).

David, Matthew (2013). *HTML5: Designing Rich Internet Applications*. New York; London: Focal Press (vid. pág. 9).

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. 8).

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. pág. 8).

Franko, Greg (mayo de 2013). *Dependency Management with RequireJS Howto*. 35 Livery Street, Birmingham, UK: Packt Publishing (vid. pág. 18).

Google (s.f.). *Google Hosted Libraries*. URL: https://developers.google.com/speed/libraries/ (vid. pág. 17).

Hunter, Jason y William Crawford (abr. de 2001). *Java Servlet Programming*. The Java Series. Sebastopol, CA, United States of America: O'Reilly Media (vid. pág. 9).

jsDelivr (s.f.). *jsDelivr: A free super-fast CDN for developers and webmasters*. URL: http://www.jsdelivr.com/ (vid. pág. 17).

Kunst, Gerber (jun. de 2014). *Programming Language Popularity Chart*. URL: http://langpop.corger.nl/ (vid. pág. 11).

Larman, Craig y Bas Vodde (ene. de 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. 14).

MacVittie, Lori (jul. de 2008). *The New Distribution of The 3-Tiered Architecture Changes Everything*. URL: https://devcentral.f5.com/articles/the-new-distribution-of-the-3-tiered-architecture-changes-everything (vid. pág. 12).

Microsoft (s.f.). *Microsoft Ajax Content Delivery Network*. URL: http://www.asp.net/ajaxlibrary/cdn.ashx (vid. pág. 17).

Montmollin, Gautier de (jul. de 2013). *The Transparent Language Popularity Index*. URL: http://lang-index.sourceforge.net/ (vid. pág. 11).

Software, TIOBE (jun. de 2014). *TIOBE Index for June 2014*. URL: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (vid. pág. 11).

Sonatype (ago. de 2008). *Maven: The Definitive Guide*. Sebastopol, CA, United States of America: O'Reilly Media (vid. pág. 14).

Stephens, David (oct. de 2008). *What On Earth is a Mainframe?* Longpela Expertise (vid. págs. 7, 8).

Twitter, Bower by (2014). *Bower: A package manager for the web*. URL: http://bower.io/ (vid. pág. 18).

Wikipedia (abr. de 2012). *List of Java Virtual Machines*. URL: https://en.wikipedia.org/wiki/List_of_Java_virtual_machines (vid. pág. 10).