La reutilización de software en Grails Framework

Sistemas de Información

Cornejo, V. E., Cázarez, P. C. A.

ecornejo@uaeh.edu.mx, shadowangel_1109@hotmail.com

Universidad Autónoma del Estado de Hidalgo, Centro de Investigación en Tecnologías de la Información y Sistemas

RESUMEN

El desarrollo de proyectos de software demanda el uso de herramientas de programación que apoyen a los arquitectos de aplicaciones en la atención de las limitantes de tiempo y costos. Sin poner en riesgo la calidad del producto final y permitiendo la construcción de soluciones robustas.

Grails Framework es un entorno de desarrollo que promueve la reutilización de código y, en particular, permite incorporar códigos elaborados por terceros a través de los plugins agrupados en un repositorio. Dichos plugins tienen funcionalidades muy diversas y en este trabajo se describe la integración de tres de ellos.

PALABRAS CLAVE

Grails, plugins, reutilización de código.

Introducción

Actualmente el proceso de desarrollo de software es condicionado por las restricciones de tiempo, la reducción de costos y, además, los clientes exigen productos de alta calidad.

En este contexto de trabajo, el arquitecto de aplicaciones requiere de metodologías que permitan manejar la complejidad de la lógica de negocios y de herramientas de programación que favorezcan la producción de productos de software de buena calidad.

La filosofía Ágil de Desarrollo de Software (Agile Alliance, 2001) favorece el desarrollo iterativo e incremental del software (Elhaibe, 2006) y permite al arquitecto de aplicaciones poder manejar las interacciones con el cliente, con los miembros del equipo y cierta flexibilidad en el proceso. El desarrollo ágil de aplicaciones se basan en

el principio de "*reutilización de código para aumentar la productividad y disminuir los riesgos de desarrollo*" (Rocher, 2006).

En la mayoría de los desarrollos reutilizamos código de alguna forma (funciones o clases de otros proyectos propios, framework o librería incluidas, etc.) ya que ahorra tiempo y esfuerzo (Gomis, 2010).

En este contexto, Grails Framework es una herramienta que agiliza y simplifica el desarrollo de aplicaciones Web (Brito, 2009), permitiendo fortalecer la construcción de proyecto robustos, aprovechando por un lado la reutilización de código, pero sobre todo acortando los tiempo de codificación, pruebas y puesta en línea.

Desarrollo

1. Reutilización de software.

La reutilización de Software "*es un proceso de la Ingeniería de Software que conlleva al uso recurrente de activos de software en la especificación, análisis, diseño, implementación y pruebas de una aplicación o sistema*" (Montilva, 2003).

Estudios relacionados con la reutilización de software han presentado los siguientes resultados (Sametinger, 1997):

- Entre el 40% y 60% del código fuente de una aplicación es reutilizable en otra similar.
- Aproximadamente el 60% del diseño y del código de aplicaciones administrativas es reutilizable.
- Aproximadamente el 75% de las funciones son comunes a más de un programa.
- Solo el 15% del código encontrado en muchos sistemas es único y novedoso a una aplicación específica. El rango recurrente potencial está entre el 15% y el 85%.

A partir de estos resultados se establece la importancia e impacto de la reutilización de código y la relación directa que tiene con: el costo, el tiempo y el esfuerzo requerido. El uso apropiado conlleva indiscutiblemente a una reducción muy significativa en ellos, además que también incrementa la calidad del software, aumenta la productividad de los grupos de desarrollo y la reducción de riesgos.

Para identificar y definir las partes del código que son reutilizables se hace uso de las siguientes características (Montilva, 2003):

- *Identificable.* Debe tener una identificación clara y consistente que facilite su catalogación y búsqueda.
- *Accesible solo a través de su interfaz.* Debe exponer al público únicamente el conjunto de operaciones que lo caracteriza y ocultar los detalles de implementación.
- *Servicios invariantes.* Las operaciones que ofrecen a través de su interfaz no debe variar. Los servicios pueden modificarse sin que afecten a la interfaz.

- *Documentado.* Debe tener documentación adecuada que facilite en la evaluación, adaptación de entornos, integración con distintos componentes e información de soporte.
- *Genérico.* Puede ser usado en una gran variedad de aplicaciones.
- *Autocontenido.* No dependa de otros componentes para cumplir su función.
- *Mantenido.* Esté en un proceso de mejoramiento continuo. Esto contribuye a ser seleccionado con mayor frecuencia para formar parte de otros sistemas.
- *Independiente a la plataforma (hardware y software), del lenguaje de programación y de las herramientas de desarrollo.*
- *Reutilizado dinámicamente.* Ser cargado en tiempo de ejecución.
- *Certificado.* Determina la calidad adquirida por agencias independientes o modelos de auto-certificación.
- *Acceso uniforme sin importar su localidad.*

2. Grails Framework.

Grails Framework es creado por Graeme Rocher en el año 2006 como una respuesta a la necesidad de agilizar, automatizar y simplificar el desarrollo de aplicaciones Web. Se basa en el principio "*mejoremos la rueda, no la reinventemos*".

Grails está basado Hibernate y Spring (Smith, 2009), que llevan mucho tiempo funcionando, han sido probadas y se ha demostrado su correcto funcionamiento. Además, se utiliza el lenguaje Groovy que permite realizar una gran cantidad de acciones en pocas líneas de código. En general, Grails toma las mejores prácticas de cada uno de los frameworks para formar un marco de trabajo estable, robusto, sencillo de usar y de fácil mantenimiento.

Una aplicación desarrollada con Grails Framework sigue la arquitectura Modelo/Vista/Controlador (MVC) (Koening, 2007). Donde el *Modelo* es el objeto que representa los datos del programa, la *Vista* es el objeto que maneja la presentación visual de los datos representados por el Modelo y el *Controlador* es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo.

3. Uso de plugins en Grails Framework.

El uso de plugins es una de las características principales de Grails Framework y que permiten lleva a cabo la reutilización de códigos que cumplen con las características descritas en la sección 1 de este artículo. En las siguientes secciones se presenta la forma en la que se pueden incorporar tres plugins en los proyectos de desarrollo de software y cómo favorecen a la reducción de tiempo de desarrollo y disminución de costos sin poner en riesgo la calidad del producto.

3.1 Spring Security Core plugin

Spring Security Core (Beckwith, 2012) es un plugin para Grails que permite implementar un mecanismo de autentificación, Figura 1, y de seguridad de acceso

basado en roles de usuario. Mediante el cual se establecen roles de usuario para la aplicación y la creación de cuentas de acceso con diferentes privilegios para la aplicación, favoreciendo la administración de perfiles de usuario.

Figura 1. Autentificación de usuario con Spring Security Core.

El primer paso para poder utilizar el *Spring Security Core* es instalarlo en nuestro proyecto de Grails, mediante el comando: `grails install-plugin spring-security-core`

El siguiente paso es la creación de 3 clases de dominio, una para representar a los usuarios (User), otra para los roles (Rol) y la tercera para relacionar los usuarios con sus roles, para conseguir lo anterior, ejecutamos el comando: `grails s2-quickstart User Rol`

Y para configurar el acceso a las URLs del proyecto, se hace uso de la *Intercepción de URL mediante un* mapa en el archivo de configuración, Config.groovy, donde se especifican las URLs y los roles de usuario que las pueden acceder. Las líneas que debemos agregar al archivo de configuración son las siguientes:

```
import grails.plugins.springsecurity.SecurityConfigType
grails.plugins.springsecurity.securityConfigType = "InterceptUrlMap"
grails.plugins.springsecurity.interceptUrlMap =
            [    '/programa/list':['ROLE_ADMIN', 'ROLE_USER'],
                 '/programa/index':['ROLE_ADMIN', 'ROLE_USER'],
                 '/programa/**':['ROLE_ADMIN', 'ROLE_USER']    ]
```

Este plugin tiene grandes ventajas, como se ha descrito, se puede configurar fácilmente, permite tomar decisiones en cuanto a la creación de los roles y así poder decidir qué contenidos se pueden mostrar dependiendo del rol asignado a cada usuario. Otras de las características sobresalientes del plugin es que las contraseñas son encriptadas para brindar mayor seguridad a la aplicación.

3.2 Export plugin

Export (Schmitt, 2012) es un plugin que permite exportar el resultado de una consulta efectuada a la base de datos a un archivo con el formato: CSV, Excel, ODS, PDF, XML y RTF, según sea requerido por el usuarios, Figura 2.

Figura 2. Barra de herramientas para exportar una vista de datos.

Para instalar el plugin se utiliza el comando: `grails install-plugin export`. Y se debe incorporar al archivo *Config.groovy*, en el apartado de *MimeTypes*, los formatos que se desean utilizar para la exportación.

Para implementar el plugin en las vistas es necesario colocar la etiqueta `<export:resource>` en la cabecera del archivo gsp. Y la etiqueta `<export:formats formats="['csv', 'excel', 'ods', 'pdf']">` en la sección de la vista en donde decidamos colocar la barra de exportación, Figura 2.
Finalmente, para que el plugin funcione es necesario agregar las siguientes líneas de código al controlador de la vista:

```
if(params?.format && params.format != "html"){
response.contentType =
          grailsApplication.config.grails.mime.types[params.format]
response.setHeader("Content-disposition", "attachment;
          filename=books.${params.extension}")
exportService.export(params.format,
          response.outputStream,Book.list(params), [:], [:])}
```

Con ello, podemos exportar todos los datos de los objetos que se encuentran dentro de una lista. Adicionalmente, es posible configurar los atributos que se desean incluir en la exportación.

3.3 File Uploader Plugin

El *File Uploader* (Hofman, 2011) es un plugin que permite trabajar con la carga y descarga de archivos de usuario en cualquier formato y tamaño dentro de nuestra aplicación, Figura 3.

Figura 3. Implementación de File Uploader.

Para instalar el plugin se utiliza el comando: `grails install-plugin file-uploader`
Y se debe incorporar al archivo *Config.groovy* la estructura siguiente:

```
fileuploader {
   docs {
          maxSize = 1000 * 1024 * 4
          allowedExtensions = ["doc", "docx", "pdf", "rtf"]
          path = "/tmp/docs/" }
}
```

En esta estructura se define el tamaño máximo y el formato de los archivos que pueden ser cargados o descargados y la ruta en dónde se ubican en el servidor.
Dentro del controlador donde se implementa el plugin es necesario instanciar la clase UFile utilizando `import com.lucastex.grails.fileuploader.UFile`, y dentro del archivo gsp de la vista se incorpora la siguiente etiqueta para subir el archivo:

```
<fileuploader:form upload="docs" successAction="success"
          successController="test" errorAction="error"
          errorController="test"  id="${test.id}"/>
```

Y para poder descargar el archivo se usa la siguiente etiqueta:

```
<fileuploader:download id="${f.id}" errorAction="error"
   errorController="mycontroller">Descargar</fileuploader:download>
```

En ambos casos, el plugin incorporar automáticamente un formulario a la vista con los controles necesarios para realizar la búsqueda del archivo para cargar y el vínculo para iniciar con la descarga, respectivamente.

Conclusiones

La reutilización de software en los proyectos desarrollados con Grails Framework potencializa la productividad y eficiencia del arquitecto de aplicaciones, al disminuir los tiempos y costos de desarrollo. Así, favorece la calidad y robustez del producto final. Debido a que los plugins disponibles en el repositorio cumplen satisfactoriamente con las características descritas por Montilva (2003) y como se ha mostrado, la integración e implementación de los tres plugins incluidos es satisfactoria.

Referencias

- *Agile Alliance*. (2001). Recuperado el 20 de Agosto de 2012, de www.agilealliance.org
- Beckwith, B. T. (Mayo de 2012). *Grails.* Recuperado el 18 de Septiembre de 2012, de http://grails.org/plugin/spring-security-core
- Brito, N. (2009). *Manual de desarrollo Web con Grails, JavaEE como siempre debió haber sido.* ImaginaWorks.
- Elhaibe, R. H. (2006). Metodologías Ágiles y Programación orientada a Aspectos. *FICCTE-Universidad de Morón.* Buenos Aíres, Argentina.
- Gomis, R. (2010). *Grails: un paso hacia el desarrollo web ágil* . Recuperado el 23 de Julio de 2012, de El taller digital: www.eltallerdigital.com/informacion.jsp?idArticulo=119
- Hofman, F. (2011). *Grails.* Recuperado el 18 de Septiembre de 2012, de http://grails.org/plugin/file-uploader
- Kent Beck, M. B. (2001). *Manifiesto por el Desarrollo Ágil de Software*. Recuperado el 9 de Septiembre de 2012, de http://www.agilemanifesto.org/iso/es/manifesto.html
- Koening, D. G. (2007). *Groovy in Action.* USA: Manning Publications Co.
- Montilva, J. A. (2003). Desarrollo de Software Basado en Componentes. IV Congreso de Automatización y Control (CAC03). Mérida.
- Pressman, R. S. (1998). Reutilización de software. En Ingeniería del Software-Un enfoque práctico. Mc Graw-Hill.
- Rocher, G. (2006). The Definitive Guide to Grails. USA: Editorial Apress.
- Sametinger, J. (1997). Software engineering with reusable component. Springer Verlag.
- Schmitt, A. (18 de Julio de 2012). Grails. Recuperado el 18 de Septiembre de 2012, de http://grails.org/plugin/export
- Smith, G. L. (2009). Grails in Action. USA: Manning Publications Co.