UNIVERSIDAD NACIONAL DE CHIMBORAZO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA EN SISTEMAS Y COMPUTACIÓN

"Trabajo de grado previo a la obtención del
Título de Ingeniero en Sistemas y Computación"

TRABAJO DE GRADUACIÓN

Título del proyecto

ANÁLISIS DE LA ARQUITECTURA OSGI EN EL DESARROLLO DE SISTEMAS INFORMÁTICOS MODULARES BASADOS EN SERVICIOS APLICADO A UN SISTEMA PARAMETRIZABLE QUE PROVEA INFORMACIÓN RELACIONADA CON LA EVALUACIÓN DE CARRERAS EN LA UNIVERSIDAD NACIONAL DE CHIMBORAZO EN EL PERIODO 2011-2012

AUTORES:

CATHERINE ALEXANDRA NARANJO DELGADO

FRANCISCO XAVIER ZABALA MIRANDA

Director: Ing. Lida Barba

Riobamba – Ecuador

2012

REVISIÓN HISTÓRICA					
DATOS DE VERSIÓN			CAMBIOS REALIZADOS		
Nombre	Fecha de Creación	Revisado	Objetivo	Descripción	Observación

UNIVERSIDAD NACIONAL DE CHIMBORAZO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA EN SISTEMAS Y COMPUTACIÓN

"Trabajo de grado previo a la obtención del
Título de Ingeniero en Sistemas y Computación"

TRABAJO DE GRADUACIÓN

Título del proyecto

ANÁLISIS DE LA ARQUITECTURA OSGI EN EL DESARROLLO DE SISTEMAS INFORMÁTICOS MODULARES BASADOS EN SERVICIOS APLICADO A UN SISTEMA PARAMETRIZABLE QUE PROVEA INFORMACIÓN RELACIONADA CON LA EVALUACIÓN DE CARRERAS EN LA UNIVERSIDAD NACIONAL DE CHIMBORAZO EN EL PERIODO 2011-2012

AUTORES:

CATHERINE ALEXANDRA NARANJO DELGADO

FRANCISCO XAVIER ZABALA MIRANDA

Director: Ing. Lida Barba

Riobamba – Ecuador

2012

Los miembros del Tribunal de Graduación del proyecto de investigación de título: **"ANÁLISIS DE LA ARQUITECTURA OSGI EN EL DESARROLLO DE SISTEMAS INFORMÁTICOS MODULARES BASADOS EN SERVICIOS APLICADO A UN SISTEMA PARAMETRIZABLE QUE PROVEA INFORMACIÓN RELACIONADA CON LA EVALUACIÓN DE CARRERAS EN LA UNIVERSIDAD NACIONAL DE CHIMBORAZO EN EL PERIODO 2011-2012"**, presentado por: Catherine Alexandra Naranjo Delgado y Francisco Xavier Zabala Miranda, dirigida por la Ing. Lida Barba.

Una vez escuchada la defensa oral y revisado el informe final del proyecto de investigación con fines de graduación escrito en la cual se ha constatado el cumplimiento de las observaciones realizadas, remite la presente para uso y custodia en la biblioteca de la Facultad de Ingeniería de la UNACH.

Para constancia de lo expuesto firman:

Ing. Fernando Molina
Presidente del Tribunal Firma

Ing. Lida Barba
Miembro del Tribunal Firma

Ing. Danny Velasco
Miembro del Tribunal Firma

AUTORÍA DE LA INVESTIGACIÓN

"La responsabilidad del contenido de este Proyecto de Graduación, corresponde exclusivamente a: Catherine Alexandra Naranjo Delgado, Francisco Xavier Zabala Miranda autores del proyecto, a la Ing. Lida Barba Tutora y al Departamento de Evaluación y Acreditación, en la persona de la Ing. Anita Maldonado Asesora de la Investigación Aplicativa; el patrimonio intelectual de la misma a la Universidad Nacional de Chimborazo.

AGRADECIMIENTO

Es mi deseo agradecer a la Universidad Nacional de Chimborazo y a la Escuela de Ingeniería en Sistemas por los conocimientos impartidos en sus aulas, al Departamento de Evaluación y Acreditación y a la tutora de nuestra tesis por ayudarnos a plasmar esta anhelada meta, a mis profesores y compañeros por ser parte de mi formación intelectual y personal. Pero de forma especial agradezco a mis padres, mi hijo, mi hermano y familia por confiar en mí y siempre brindarme su apoyo incondicional. Finalmente a Xavier por ser mi amigo y compañero en este arduo camino y a todos quienes siempre han estado junto a mí en los momentos más difíciles de mi vida.

Catherine Naranjo D.

Agradezco a mi Papá y a mi Mami por el apoyo económico en todos estos años, a mi Mami y a mis ñañas por todo el apoyo moral que me brindaron, por soportarme y enseñarme a ser una mejor persona. A mis compañer@s y amig@s que sin una sola palabra me han enseñado cosas valiosas para mi vida. A la Universidad Nacional de Chimborazo por haberme dado la oportunidad de instruirme en sus aulas. A todos mis profesores, tutores y todas aquellas personas que me brindaron su tiempo y conocimiento, no doy nombres por temor a omitir alguno. A Cathy por su ayuda y comprensión, por ser una buena amiga y tenerme paciencia en este trayecto. ***¡Gracias Mami por toda tu ayuda, ahora me toca a mí!***

Xavier Zabala M.

DEDICATORIA

Dedico este trabajo a mi familia, de forma especial a mi padre por ser un ejemplo de tenacidad y honradez, a mi madre por estar junto a mí brindándome su amor, apoyo y comprensión a pesar de la adversidad, a mi hermano por estar siempre junto a mí de forma incondicional y sobre todo a mi hijo David Marcelo por ser el motivo para luchar cada día. Gracias por no solo brindarme su apoyo a lo largo de mi carrera sino de mi vida

Catherine Naranjo D.

Dedico esta tesis a mi familia, en especial a mi Mami que con tenacidad me ha alentado en mis momentos de descreimiento, momentos que han sido bastante continuos en este extenso trayecto. Sin el apoyo que me han brindado, nunca habría tenido las fuerzas para culminar esta etapa de mi vida. Y aunque estoy consciente de que merecen algo mucho mejor por todo lo que han hecho por mi, aun así, mi retribución empieza con esto. ¡De todo corazón!

Xavier Zabala M.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICOS

RESUMEN

El presente trabajo tiene como objeto determinar la importancia de aplicar la arquitectura OSGi en el desarrollo de sistemas informáticos modulares basado en servicios.

De acuerdo con el análisis de la arquitectura y en contraste con sistemas desarrollados sin la arquitectura OSGi, mediante los indicadores de comparación: Modularidad, reusabilidad y complejidad en el desarrollo, los cuales son requisitos o características de los sistemas modulares, se determina que el desarrollo mediante la arquitectura OSGi es efectiva al 90% para el desarrollo de sistemas informáticos modulares basados en servicios, atribuyéndoles no solo modularidad lógica sino modularidad física; mientras que otras formas de desarrollo sin la arquitectura, son eficientes al 10%, permitiéndole a un sistema ser modular únicamente en forma lógica.

Al momento se puede considerar a la arquitectura OSGi como la adecuada para el desarrollo de sistemas informáticos modulares basados en servicios, permitiéndoles cumplir todas las características que debe tener un módulo: autónomo y auto – contenido, altamente cohesivos, débilmente acoplados, desplegable, administrable, verificables, componible, reusable, intercambiable, desarrollado en paralelo y mantenibles.

SUMMARY

This study aims to determine the importance of applying OSGi architecture in developing modular systems based on services.

According to the architecture analysis and in contrast with systems developed without OSGi architecture, through comparison indicators: modularity, reusability and complexity in development, which are requirements or characteristics of modular systems development, it is determined that the use of OSGi architecture is 90% effective for the development of modular systems based on services, attributing not only logical modularity but also physical modularity, while other forms of development without the architecture are 10% efficient allowing a system be modular only in logic form.

Nowadays OSGi architecture can be considered as an adequate tool for developing service-based modular systems, enabling them to meet all the characteristics required for a module: independent and self-content, highly cohesive, loosely coupled, deployable, manageable, verifiable, composable, reusable, interchangeable, developed in parallel and maintainable.

INTRODUCCIÓN

Desde hace muchos años se han estado probando técnicas de programación que permiten disminuir la complejidad en el desarrollo de sistemas informáticos, de ahí que han nacido iniciativas e ideas que actualmente son muy populares para el desarrollo de software, como por ejemplo, la programación estructurada, la programación orientada a objetos, la programación orientada a aspectos, el desarrollo por capas y/o por niveles (n-layers, n-tiers), los patrones de diseño, el MVC (modelo, vista, controlador), el MVP (modelo, vista, presentación), DDD (Domain Driven Desing), Granularidad (Martin, Granularidad, 1996), El principio de inversión de dependencia (Martin, Principio de Inversión de Dependencias, 1996), Object Oriented Desing (OOD) (Martin, Principios de la Orientación a Objetos, 1996).

Todos estos procesos ayudan a la modularización de una aplicación. Con modular se hace referencias a particionar un sistema de acuerdo a ciertos principios de diseño y a una estrategia de desarrollo, gobernando las dependencias entre las partes resultantes (Ramos, Modularización efectiva, 2010).

Pero a pesar de esto, el aumento en el tamaño y sobre todo en la complejidad del desarrollo de software ha ido creciendo vertiginosamente, en los últimos años la tendencia va hacia la computación en la nube (Cloud Computing) y todos los servicios que esta brinda: la Infraestructura como servicio (IaaS), la Plataforma como servicio (PaaS), y especialmente el Software como servicio (SaaS), el cual provee de aplicaciones directamente a los usuarios finales sin la necesidad de que ellos adquieran costosos sistemas de hardware o software, dando como resultado un desarrollo de software constante con la disponibilidad inmediata de nuevas versiones, menores costos de mantenimiento para usuarios y muchas otras ventajas (CloudSigma, 2011).

Agustín Ramos (Ramos, Modularización efectiva, 2010), nos da una explicación más amplia, sobre el porqué a pesar de que existen tantas opciones para un mejor desarrollo del software, este sigue incrementando su complejidad y lo que es más importante por qué las técnicas de programación antes mencionadas ya no cumplen con su principal objetivo, la modularidad:

A principios de los años 70, David Parnas, sentó las bases de la modularización en su ensayo “On the criteria to be used in decomposing systems into modules” (Parnas, 1972),

que tuvo gran impacto en la manera en que construimos software desde entonces. El problema deriva en que, en general, hemos fallado en aplicar de manera efectiva este conocimiento en el software que desarrollamos día a día. Parte de esta falla proviene del hecho de que el software es cada vez más complejo y utilizado en más contextos.

Otra razón tiene que ver con lo que Markus Völter (Völter, 2005) señala como "un énfasis en la tecnología en lugar del diseño": una gran cantidad de desarrolladores conocen los aspectos técnicos de la implementación de un Enterprise Java Bean (EJB) o un servicio web por ejemplo, pero se encuentran muy limitados al establecer criterios de granularidad, agregación y empaquetamiento que proveerán a un sistema con características tan deseadas como la facilidad para realizar cambios.

La tercera causa es que los conceptos planteados por Parnas y otros pioneros, son aplicados principalmente a nivel de clases y objetos. Sin embargo, el tamaño y complejidad del software desarrollado hoy en día exige nuevos niveles de agregación y una definición de módulo a un nivel de abstracción más alto.

Esta situación ha provocado que surjan esfuerzos para remediarla. Por un lado, un renovado énfasis de los gurús del software sobre buenos principios y prácticas de diseño más que sobre tecnologías de implementación. Por otro lado, el advenimiento de iniciativas como el proyecto Jigsaw (Corporation, 2012) para modularizar el propio JDK y la explosión de interés en OSGi (Alliance O. , 2012) como fundamento para implementar arquitecturas modulares sobre la Máquina virtual de Java (JVM).

Este documento se centrará en el desarrollo de software para sistemas informáticos modulares basados en Java. En las siguientes secciones se analizarán todos los aspectos relacionados con los principios y prácticas de diseño para que un sistema informático en la actualidad pueda ser llamado modular y además se analizará a OSGi como fundamento para implementar arquitecturas modulares sobre la máquina virtual de Java.

CAPÍTULO I

PROBLEMATIZACIÓN

1.1 Identificación y descripción del problema

Con el transcurso del tiempo, ha incrementado la importancia de contar con información confiable, íntegra y oportuna para lograr los objetivos estratégicos de las organizaciones.

La Universidad Nacional de Chimborazo en su proceso de Acreditación, tiene la desventaja de no disponer de un sistema de información que le permita obtener datos de todos y cada uno de los indicadores requeridos por el Consejo de Evaluación, Acreditación y Aseguramiento de la Calidad de la Educación Superior (CEAACES) los criterios bajo los cuales están regidos los mismos pueden cambiar en el tiempo por lo cual el sistema propuesto debe ser extensible y debe permitir la inserción o eliminación de nuevos criterios, tampoco cuenta con sistemas implementados que provean información y puedan ser integrados al mismo.

El proceso de obtención de información de estos indicadores y la generación de estadísticas se lo realiza de manera manual lo cual implica demoras.

La implementación de un sistema informático modular, basado en servicios, permitirá la automática e inmediata generación de estadísticas y resultados, además, por su diseño, ofrecerá la ventaja de que cada uno de los criterios de evaluación, trabajen de una forma independiente, pero a la vez, colaborativa dentro de la misma aplicación

1.2 Análisis crítico

El control para el cumplimiento de los indicadores en el proceso de Acreditación por parte de la Universidad Nacional de Chimborazo, al ser llevado en forma manual, trae consigo muchos inconvenientes, los cuáles se detallan a continuación:

- Pérdida de tiempo en la búsqueda y obtención de datos solicitados.
- Generación manual de documentación.
- Obtención incorrecta e incompleta de datos.
- Mal manejo y almacenaje de la información.
- No se garantiza la integridad y disponibilidad de los datos.
- Ausencia de copias de seguridad de la información.
- Problemas de seguridad de la información.
- Lentitud en la generación de estadísticas.

1.3 Prognosis

El desarrollo de un sistema modular basado en servicios, aumentará la adaptabilidad del sistema por su capacidad de inserción, actualización y eliminación de componentes en tiempo de ejecución, lo cual permitirá que el sistema se pueda adaptar a nuevos requerimientos en el transcurso del tiempo.

Además, a través del correcto almacenamiento y manejo de información sumado a una adecuada generación de estadísticas y reportes, ayudará a que la Universidad Nacional de Chimborazo alcance esa tan anhelada acreditación.

1.4 Justificación

Todas las organizaciones, sean educativas o de cualquier otra índole, en sus inicios realizan todos los procesos que involucran el funcionamiento de la empresa de forma manual y mientras el tiempo pasa y la organización crece, también crece la complejidad en sus procesos y por ende la complejidad en la realización del software que los automatice. Más aún cuando lo que se desea es que estos sistemas sean escalables y puedan tener un acoplamiento con sistemas que se desea desarrollar en el futuro.

Es en estos casos en que la modularización de sistemas informáticos se vuelve imprescindible para que una empresa pueda ir mejorando e incrementando sus sistemas mediante la reutilización, disminución del acoplamiento y aumento en la cohesión de cada módulo o sistema, OSGi provee el medio ambiente ideal para crear sistemas modulares con un alto grado de cohesión y bajos niveles de acoplamiento, permitiendo de

esta manera crear sistemas modulares que interactúen entre sí, sin importar el tiempo en el que se hayan creado.

Actualmente la Universidad Nacional de Chimborazo, se encuentra en un proceso de Acreditación en base a una Disposición Transitoria de la Constitución de la República del Ecuador, que señala que todas las Universidades y Escuelas Politécnicas, así como sus carreras, programas y posgrados, deberán haber cumplido con la evaluación y acreditación del Consejo de Evaluación, Acreditación y Aseguramiento de la Calidad de la Educación Superior, razón por la cual, se hace necesario la implementación del sistema de indicadores que permita gestionar la información de una manera adecuada y dé como resultado estadísticas claras y confiables de cada uno de los mismos.

1.5 Delimitación

El sistema a desarrollarse no incluye el proceso de los subsistemas aún no implementados en la Universidad Nacional de Chimborazo, tampoco la implementación de las vías de comunicación, como los servicios web hacia otros sistemas de la Universidad.

Los requerimientos del sistema parametrizable que provea información relacionada con la evaluación de carreras en la Universidad Nacional de Chimborazo en el periodo 2011-2012, se basarán en los documentos oficiales emitidos por la Comisión de Evaluación Interna de la Universidad, titulado “Evidencias y Tareas que se derivan del Modelo General para la Evaluación de Carreras con fines de Acreditación”, en el período 2011.

1.6 Formulación del problema

¿Permite el análisis de Arquitectura OSGi, desarrollar un sistema informático modular basado en servicios, aplicado a un sistema parametrizable que provea información relacionada con la evaluación de carreras en la Universidad Nacional de Chimborazo en el periodo 2011-2012, de manera eficiente?

1.7 Objetivos

1.7.1 General

Analizar la arquitectura OSGi en el desarrollo de sistemas informáticos modulares basados en servicios, aplicados a un sistema parametrizable que provea

información relacionada con la evaluación de carreras en la Universidad Nacional de Chimborazo en el periodo 2011-2012.

1.7.2 Específicos

- Estudiar la arquitectura OSGI "Open Services Gateway Initiative", para desarrollar sistemas informáticos modulares basados en servicios.
- Analizar los requerimientos de cada uno de los indicadores necesarios en el proceso de Acreditación por carreras de la Universidad Nacional de Chimborazo.
- Desarrollar el sistema parametrizable de indicadores de evaluación con fines de acreditación de carreras de la Universidad Nacional de Chimborazo que entregue los resultados de forma inmediata y las correspondientes estadísticas que se requieran.

1.8 Hipótesis

La Arquitectura OSGi permitirá el desarrollo eficiente de un sistema informático modular basado en servicios, aplicado a un sistema parametrizable que provea información relacionada con la evaluación de carreras en la Universidad Nacional de Chimborazo en el periodo 2011-2012.

CAPÍTULO II

FUNDAMENTACIÓN TEÓRICA

2.1 Sistemas Informáticos Modulares

2.1.1 Modularidad

La modularidad abarca una infinidad de aspectos de la programación que a menudo sedan por sentados. Y sin importar cuanta experiencia se tenga en el diseño de sistemas o lo mucho que se conozca sobre buenos diseños que tiendan a ser modulares, la realidad es que no se aplica al desarrollo de un sistema completo de un conjunto de piezas lógicamente independientes llamadas módulos.

Un módulo define un límite lógico que se pueda cumplir, ya sea que el código forme parte de un módulo (que se encuentre en su interior) o que no forme parte (que se encuentre en otro módulo por ejemplo). El código interno (implementación) de un módulo, es visible sólo para el resto de código que forma parte de ese módulo. Para el resto, los únicos detalles visibles son los que de forma explícita se declaran como públicos, como se muestra en la Figura 1. Este aspecto de módulos los hace parte integral de diseñar la estructura lógica de una aplicación.

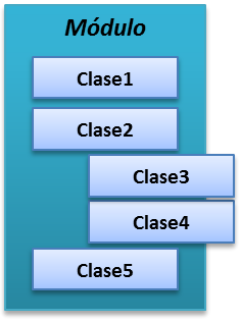

Figura 1. Un módulo define un límite lógico.

El módulo en si está de forma explícita en control de qué clases son públicas o están expuestas para el uso externo (como la clase 3 y 4), y, cuáles están encapsuladas o privadas para el uso interno (clases 1, 2 y 5). Fuente: OSGi in Action (Hall, Pauls, McCulloch, & Savage, 2011).

2.1.1.1 Tipos de Modularidad

Como se muestra en la figura 1, un módulo define un límite lógico en la aplicación, el cuál impacta en la visibilidad del código de forma similar a lo que pasa con la programación orientada a objetos, la cual indica la visibilidad o el alcance de accesibilidad hacia o desde un objeto a sus clases y/o atributos y métodos.

2.1.1.1.1 Modularidad Lógica

El diseño lógico define el cómo se construye el sistema en sí, si se usarán clases, operadores, interfaces, patrones de diseño, que métodos y atributos se usaran y en que clases se las utilizará. También se define la estructura de paquetes que tendrá el sistema (Knoernschild, Java Application Architecture: Modularity Patterns with Examples Using OSGi, 2012).

2.1.1.1.2 Modularidad Física

Se refiere a cómo el código (el diseño lógico) es empaquetado y/o puesto a disposición para su despliegue. Otros aspectos del diseño físico, son el seleccionar que clases pertenecen a una u otra unidad física, también el administrar las relaciones entre las unidades lógicas.

Es posible tener un diseño lógico sin tener un diseño físico (por ejemplo cuando tenemos solo un módulo) o empaquetar varios módulos lógicos en un único módulo físico (Hall, Pauls, McCulloch, & Savage, 2011). Por consiguiente, el diseño físico, actualmente, es igual o mucho más importante que el diseño lógico, ya que específicamente este trata de aplicar una real modularización a un sistema de software.

> *"La modularización es la capacidad de romper o dividir un sistema de software en módulos, los que operan de forma independiente uno del otro pero aún pueden combinarse para hacer un trabajo útil (Parnas, 1972), permitiendo que el sistema entero sea mucho más fácil de entender y mantener"*

2.1.2 Definición de módulo

"Módulo es un componente autónomo y auto-contenido que pertenece o no a un sistema mucho más grande, tiene alto grado de cohesión y bajo grado de acoplamiento, puede ser desplegable, administrable, intercambiable, reutilizable. No debe manejar estado (stateless), debe poseer componibilidad (composable), debe ser verificable y proveer una interfaz concisa a los consumidores".

Se debe tener en cuenta de que si la cohesión es una métrica interna del enfoque de un módulo, el acoplamiento es una métrica externa de cómo interactúan los módulos entre sí. Para entender mejor la definición de módulo, a continuación se detalla cada una de las características que debe poseer un módulo para que pueda estar bien definido.

2.1.2.1 Autónomos y Auto-Contenidos

Un módulo es un todo lógico, no necesita de referencias externas para su existencia y funcionamiento. Puede ser instalado y desinstalado como una única unidad. No es atómico, consta de partes o piezas pequeñas, las que no pueden estar solas, y si se removiera cualquiera de estas, el módulo dejaría de funcionar.

2.1.2.2 Altamente Cohesivos

Un módulo que es altamente cohesivo, debe tener una responsabilidad y funcionalidad perfectamente delimitada dentro de un sistema mayor en el que se encuentre integrado, en otras palabras se centra en una única tarea y no contiene nada más que no contribuya a ese enfoque. Como resultado, un módulo cohesivo tiende a ser específico, robusto, reusable y mucho más entendible (Walls, 2010).

2.1.2.3 Débilmente Acoplados

Los módulos débilmente acoplados dependen de otros módulos solo a través de abstracciones estables, desconociendo totalmente la implementación que hay por debajo. En otras palabras, un módulo debe evitar cualquier dependencia con el resto de módulos para realizar la tarea que tiene asignada. Como resultado, los cambios en la implementación de un módulo, rara vez tienen un impacto con los otros módulos que interactúan con él (Walls, 2010).

2.1.2.4 Desplegables

Los módulos son una unidad de despliegue. A diferencia de otras entidades de software como las clases y paquetes, un módulo es una unidad discreta de la implementación que puede ser desplegado junto a otros módulos. En este sentido, los módulos representan algo más físico que las clases o paquetes, los cuales son unidades de software intangibles (diseño lógico).

2.1.2.5 Administrables

Los módulos son unidades de administración. En presencia de un sistema de tiempo de ejecución para módulos, estos pueden ser instalados, desinstalados y actualizados. Durante el desarrollo, dividir el sistema en módulos ayuda a aliviar una serie de actividades que normalmente son complicadas. Administrar los módulos permite a los desarrolladores crear o construir independientemente módulos autónomos, además que mejora los límites que hay entre ellos.

2.1.2.6 Comprobables o Verificables (Testables)

Un módulo puede ser verificable, la verificabilidad (testability) se refiere a la facilidad con la que puede demostrarse el correcto funcionamiento del software mediante su test o prueba (Paez, 2010). Al igual que una clase puede ser probada de forma

independiente, a través del desarrollo basado en pruebas (TDD), un módulo también puede ser probado de forma independiente.

2.1.2.7 Componibilidad (Composable)

Los módulos pueden estar compuestos de otros módulos o se los puede combinar libremente para generar módulos más complejos. Este criterio tiene como finalidad la reusabilidad. Se busca que el trabajo desarrollado en un proyecto pueda ser aprovechado en los siguientes proyectos.

2.1.2.8 No deben manejar estado (Stateless)

Los módulos no manejan estado. Existe solo una única instancia de una versión específica de un módulo. No se crea una instancia de los módulos como se lo hace con las clases contenidas en los módulos, las que una vez instanciadas conservan su estado, no así con los módulos. Ejemplos de entidades de software que se adhieren a esta definición son los archivos EAR, WAR y los JAR.

2.1.2.9 Reusabilidad

La reusabilidad es la capacidad de reutilización de un sistema o partes de él (módulos), es decir, hasta qué punto se puede volver a emplear un módulo en otras aplicaciones, en relación al empaquetamiento y alcance de las funciones.

2.1.3 Beneficios de la Modularización.

2.1.3.1 División del problema

Facilita el desarrollo de sistemas informáticos de gran tamaño. Divide el problema en unidades lógicas pequeñas y permite resolverlo de forma incremental, reduciendo la complejidad del desarrollo.

2.1.3.2 Intercambiabilidad

Si cada módulo en una aplicación, es conocido solamente por su interfaz pública y no por su implementación interna, entonces será fácil intercambiar un módulo con otro (la actualización o mejora de un módulo en un sistema, por ejemplo), siempre y cuando ambos tengan la misma interfaz pública, aunque su implementación sea diferente. Al

separar la implementación de la solución de cada dominio, es posible utilizar esta implementación en un contexto distinto.

2.1.3.3 Compresibilidad

Los módulos cohesivos con límites bien definidos son mucho más fáciles de entender y estudiar individualmente, fundamentalmente esto conduce a una mejor comprensión de la aplicación entera.

2.1.3.4 Desarrollo Paralelo

Los módulos pueden ser desarrollados virtualmente sin depender unos de otros, permitiendo distribuir las tareas de desarrollo entre diferentes personas / equipos.

2.1.3.5 Mantenibilidad

Los sistemas modulares, cuyos módulos son altamente cohesivos y débilmente acoplados, son más fáciles de modificar y extender. Incrementando de esta manera la extensibilidad del sistema.

Figura 2. Modularidad

A mayor grado de modularidad, el sistema es más fácil de mantener y extender. Fuente: Kirk Knoernschild (Knoernschild, Osgi en la Empresa, 2010)

2.1.4 Efectos negativos de la modularización

Maximizar la modularización y por consecuencia, el reuso, tiene otros efectos no tan deseables. Específicamente, existe la llamada paradoja del reuso, la cual se ilustra en la figura 3. Entre más modular y reusable es un diseño, más difícil es usar partes del mismo. Esto se debe a que, a diferencia de un diseño monolítico, donde para usar el sistema simplemente debemos copiarlo o instalarlo en el lugar adecuado, en un sistema modular debemos estar conscientes de las dependencias entre y hacia otros módulos y saber manejarlas adecuadamente.

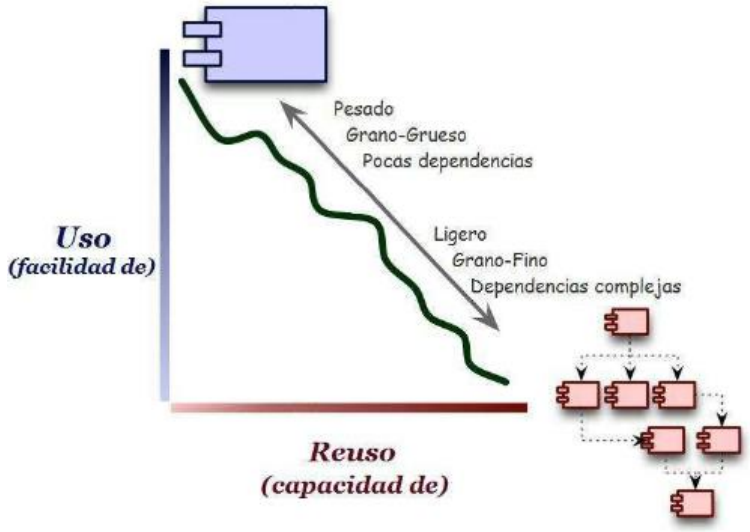

Figura 3. Paradoja del re-uso

De no hacerlo, es probable que nos involucremos en lo que coloquialmente conocemos como “infierno de dependencias”, que puede tomar distintas formas:

2.1.4.1 Demasiadas dependencias

Una aplicación con muchas dependencias puede tener características negativas como dificultad de instalar y configurar, gran tamaño, inestabilidad y fragilidad relativa al cambio de estas dependencias.

2.1.4.2 Dependencias cíclicas

Se da cuando un módulo A depende de otro módulo B, el cual a su vez depende directa o indirectamente de A. Esta situación denota que no existe una correcta separación de las responsabilidades, ya que siempre que usemos A necesitaremos usar B o viceversa. No hay posibilidad de reuso más que en conjunto. Frecuentemente esto no quiere decir que no pueda existir reuso, solamente que el código no fue colocado en el lugar adecuado para permitirlo de manera más granular.

2.1.4.3 Largas cadenas de dependencias

Se da cuando una cadena de dependencias transitivas es muy larga. Una dependencia transitiva es aquella que se obtiene de manera indirecta, cuando un módulo que utilizamos, hace a su vez, uso de otros módulos. Al tener largas cadenas de dependencias, puede resultar muy laborioso determinar cuáles son todas las dependencias necesarias para poder usar el módulo que realmente es de nuestro interés.

2.1.4.4 Dependencias en conflicto

Se da cuando queremos usar dos módulos, cada uno de los cuales tiene dependencias transitivas hacia versiones específicas pero diferentes de un tercero. En esta situación no podemos simplemente descartar una de las versiones porque, o bien el sistema no compilará, o peor aún, tendrá defectos que no serán visibles hasta el momento de ejecución, pudiendo incluso permanecer escondidos por largo tiempo.

2.1.5 Modularidad en java

A menudo se cree que los archivos JAR son la unidad de modularidad en Java, ya que cumplen con la mayoría de las características de modularidad antes mencionadas, pero existe un problema con estos archivos, o con este concepto de módulos dentro de Java. Y es que solo funciona o solo se cumple en tiempo de desarrollo, pero en tiempo de ejecución, pierde todo el sentido de modularidad.

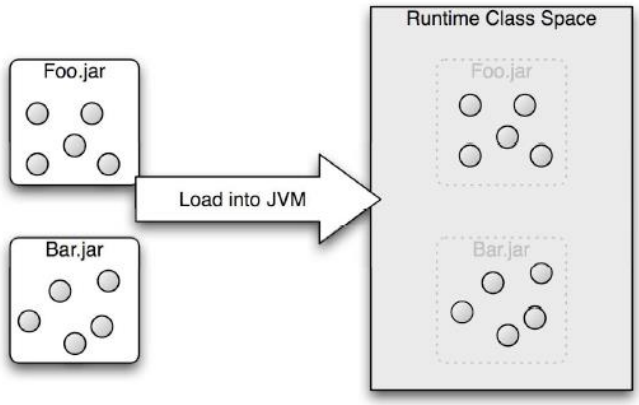

Figura 4. Modularidad en JAVA

Los límites impuestos por los archivos JAR, son artificiales y se desvanecen en tiempo de ejecución. Fuente (Walls, 2010)

Esto se debe al proceso mediante el cual es cargado un fichero JAR en la Máquina Virtual de JAVA (JVM). Una JVM tiene un único classloader responsable de cargar en esta, cualquier clase que se encuentre en el classpath, todo en un mismo espacio en común, perdiendo el sentido de modularidad debido a que los límites débiles de un archivo JAR fallan al restringir el acceso a su implementación interna desde los otro módulos, haciéndolos vulnerables al mal uso y a que exista un fuerte acoplamiento entre ellos. Además, otro problema que presentan estos archivos es que aparte de añadir el número de

versión en el nombre del archivo, no ofrecen otra opción para el versionado, dificultando el conocer con cuál versión de un archivo JAR dado, se está lidiando.

Aquí entra en juego OSGi, brindando una verdadera modularización, tanto en tiempo de desarrollo como en tiempo de ejecución, fortificando los límites de un archivo JAR, para que el resto de módulos puedan ver solo lo que les corresponde y nada más. El cómo lo hace, es mediante el framework de OSGi que define un sistema de módulos dinámicos para JAVA. A estas unidades de modularización se las llama bundles y no son otra cosa que un archivo JAR, pero que contiene dentro de sí un fichero llamado MANIFEST.MF en el que se define los límites del bundle, cuáles son los paquetes que se tienen que importar o exportar, además de la versión del bundle. El siguiente segmento analizará a profundidad OSGi.

2.2 OSGi

2.2.1 Historia

En Marzo de 1999 un grupo de empresas especializadas en áreas como informática, electrónica, telecomunicaciones, automotrices y fabricantes de electrodomésticos, se unieron y formaron la Alianza OSGi, encargada de definir, crear y promover una plataforma abierta de servicios de software que permita diseñar plataformas compatibles que puedan proporcionar múltiples servicios. Inicialmente fue aplicado en redes domóticas y para dispositivos embebidos, pero gracias a sus objetivos basados en la modularidad y el dinamismo, han hecho que OSGi sea confiable para encarar un mercado mucho más amplio. Y es así como ha sido adoptada para mejorar el desarrollo de aplicaciones empresariales (Daniel Haischt, 2010).

- OSGi es una especificación que ofrece una arquitectura abierta y común para los proveedores de servicios, desarrolladores, proveedores de software, operadores y proveedores de equipos de puerta de enlace, al desarrollar, implementar y administrar los servicios de una manera coordinada (Alliance T. O., 2011).
- OSGi es una tecnología basada en componentes, esto es un conjunto de elementos (bundles) con interfaces definidas, que habitan en un medioambiente de tiempo de ejecución (runtime).
- OSGi es un entorno de runtime/hosting que controla el ciclo de vida de los bundles (los instala, inicia, para, desinstala) y controla las dependencias entre estos.

- OSGi es una tecnología basada en servicios, ya que las interfaces de un bundle son separadas de su implementación. La interfaz es publicada por el bundle en la capa de servicios de OSGi.

Después de liberar la tercera versión de la especificación, OSGi que hasta ese entonces era el acrónimo de Open Services Gateway Initiative, pasó a convertirse en una marca de la tecnología, dejando atrás las siglas que lo representaban. La especificación de OSGi que actualmente se encuentra disponible, es la versión 4.3. Fue liberada en Abril de 2011 y es nombrada por la Alianza como Plataforma de Servicios de OSGi (OSGI Service Plataform).

La Plataforma de Servicios OSGi se compone de dos partes: el framework de OSGi y los Servicios estándar (como se muestra en la figura 5).

Figura 5. Plataforma de servicios de OSGi

En la figura 6, se muestra la arquitectura de OSGi, y cómo encajan estos dos aspectos de la Plataforma de servicios, tanto el framework de OSGi como los servicios estándar dentro de la arquitectura.

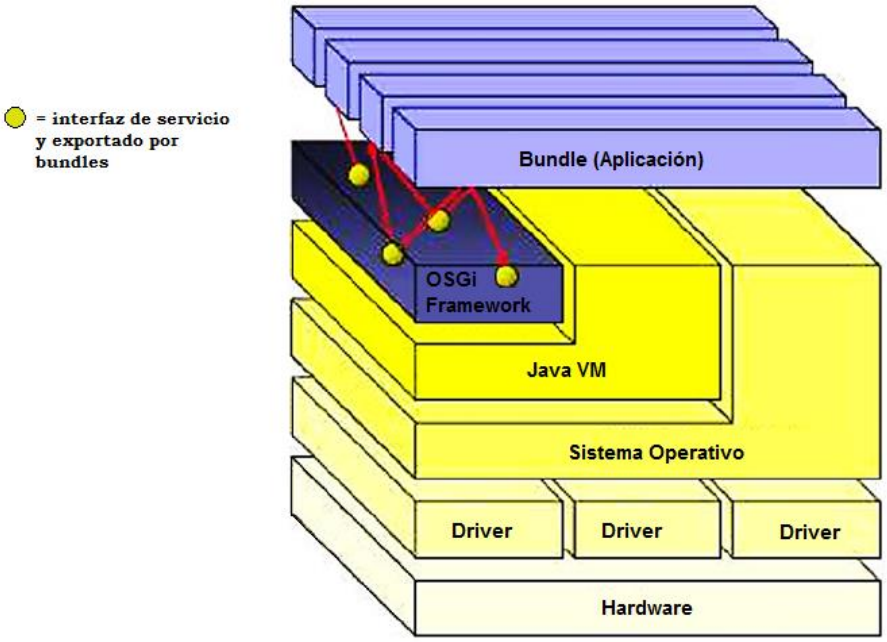

Figura 6. Arquitectura de OSGi

Fuente (Rodríguez, 2010)

Los servicios estándar definen las API reutilizables para las tareas comunes, tales como el registro de actividad (Logging) y configurar las preferencias o características de los servicios, además proporcionan una determinada funcionalidad a otros servicios o al usuario final, al ejecutarse sobre el Framework. También será el encargado de la interacción entre dispositivos o redes de dispositivos para su comunicación.

2.2.2 Framework

El framework es el medio ambiente que gestiona el ciclo de vida de una aplicación. Por esta razón juega un papel muy importante en el desarrollo de aplicaciones basadas en OSGi. Debido a que la especificación del framework de OSGi, es una tecnología abierta, existen muchas implementaciones de su núcleo.

2.2.2.1 JEF FREE

(Java Embedded Framework FREE) Aplicación de código abierto para las pasarelas residenciales, que quedó abierta la licencia en el 2003, y es para la especificación OSGi 2.0.

2.2.2.2 OSCAR

(Open Service Container Arquitecture) Es una herramienta para generar paquetes, se denomina Mangen, para funcionalidades con HTTP y JMX.

2.2.2.3 Knopflerfish

(Anteriormente Gatespace Telematic) Es una distribución libre de Knopflerfish OSGi, Makeware es el principal inversor y promotor de Knopflerfish, cumple con la especificación OSGI R4.

2.2.2.4 Equinox

Desde un punto de vista de código, Equinox es una aplicación de la framework OSGi especificaciones R4, un conjunto de paquetes que implementan diversas funciones y servicios de OSGi para el funcionamiento de los sistemas basados en OSGi.

2.2.2.5 Félix

Para el desarrollo del proyecto se ha decidido utilizar Félix, sus origen fue un proyecto de la Fundación Apache, que se esfuerza por implementar una plataforma de servicio para la especificación R4 de OSGi, la cual incluye el Framework de OSGi y los servicios estándar, además de proveer soporte para otras interesantes tecnologías relacionadas con OSGi. El objetivo final de Félix es ofrecer una implementación totalmente compatible del Framework de OSGi y sus servicios estándares, además de dar respaldo a toda la comunidad existente alrededor de esta tecnología.

Actualmente, Félix implementa una gran parte de la versión de especificación completa. A pesar de este hecho, las funcionalidades que ofrece Félix del Framework de OSGi resultan muy estables por que pueden ser utilizadas dentro de otros proyectos, además de su uso como un plugin o mecanismo de extensión dinámico. Esta posibilidad le da a Félix mayor ventaja frente a otros sistemas utilizados para tareas similares, como el Java Management Extensions (JMX).

2.2.3 Componentes

La plataforma de servicio OSGi tiene una arquitectura en capas y está diseñada para ejecutarse en distintos perfiles Java estándar, como se muestra en la figura 7.

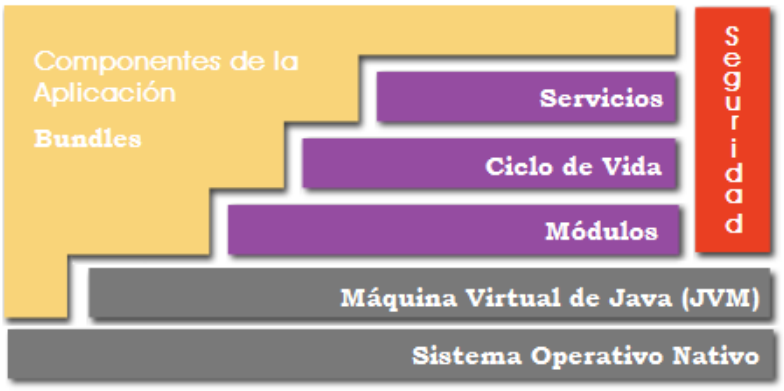

Figura 7. Framework de OSGi. Fuente (Cogoluègnes, Templier, & Piper, 2011)

Cada capa juega un papel importante en hacer de OSGi una plataforma modular, aplicando reglas de visibilidad entre los módulos, haciendo que los módulos evolucionen o pasen a través de un conjunto de estados (instalar, iniciar, parar, actualizar y desinstalar) y proveyendo de un agente o servicios para que los módulos se comuniquen unos con otros. Cada contenedor OSGi se basa o está sobre la máquina virtual de Java (JVM) y específicamente, implementa tres capas que permiten la modularidad dinámica, estas son:

- ✓ **Servicios** – Se centra en la interacción y comunicación entre bundles, ofreciendo un modelo de publicación-búsqueda-enlace de los archivos contenidos en ellos.
- ✓ **Ciclo de Vida** – Provee el API (Application Programming Interface) para instalar, iniciar, parar, actualizar y desinstalar bundles.
- ✓ **Módulos** – Esta capa define el cómo un bundles puede importar y exportar código, además del empaquetado y compartición del mismo.

A continuación se detalla a profundidad cada una de estas capas:

2.2.3.1 Primera Capa: Módulos

La capa de módulos es donde la infraestructura de OSGi procesa los aspectos modulares de un paquete. Redefine el concepto de módulo, nombrándolo como bundle. Lo que diferencia a un bundle JAR de un archivo JAR típico, es el archivo de Manifiesto (Manifest.mf) que es usado por el framework de OSGi, para administrar las características de modularidad.

Esta capa hace cumplir las reglas de visibilidad entre módulos, lo logra gracias a que aísla el classloader utilizando uno por bundle, a diferencia de la plataforma estándar de Java que utiliza un único classloader para todos los módulos. Por defecto un bundle actúa como una caja negra en OSGi. Si se desea exportar recursos, clases o interfaces hacia

otros bundles, se tendrá que especificarlo explícitamente en el archivo de manifiesto, por el contrario, si se desea utilizar clases o paquetes de otros bundles, primero se deberá especificarlos como visibles en el manifiesto del bundle que contenga esos archivos. Además provee un sistema de versionado permitiendo varias versiones de un mismo bundle, que pueden estar disponibles para distintos clientes.

La figura 8 muestra la estructura interna de un bundle, a continuación se describe cada uno de los componentes de este archivo:

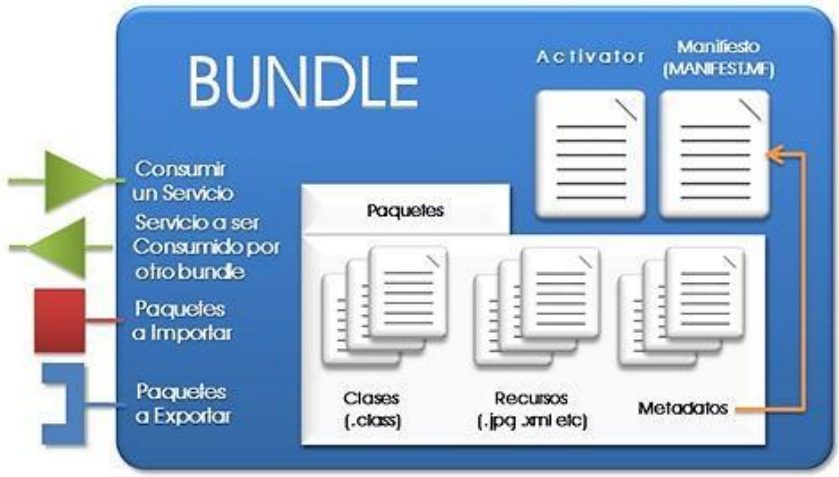

Figura 8. Estructura de un bundle

2.2.3.1.1 Bundle

Un bundle es una unidad física de modularidad en la forma de un archivo JAR, que:

- ✓ Contiene los recursos necesarios para proporcionar alguna funcionalidad. Estos recursos pueden ser clases (class), así como archivos HTML, archivos de ayuda, iconos, imágenes, etc.
- ✓ Un bundle JAR también puede incorporar archivos JAR adicionales que contengan recursos y clases. Esto, sin embargo, no es algo recursivo.
- ✓ Contiene un archivo de manifiesto que describe el contenido del bundle JAR y provee información sobre el paquete. Este archivo utiliza cabeceras para especificar la información de que el Framework necesita para instalarse y activar correctamente un paquete. Por ejemplo, establece las dependencias sobre otros recursos, como paquetes de Java, que deben estar disponibles para el bundle antes de que este se ejecute.

Una vez que el paquete se ha iniciado, su funcionalidad es suministrada y los servicios están expuestos a otros paquetes instalados en la Plataforma de Servicios OSGi.

2.2.3.1.2 Archivo MANIFEST.MF

Este archivo permite configurar las diferentes propiedades de un bundle y su comportamiento dentro de un contenedor OSGi. El Framework define las cabeceras del manifiesto de OSGi como Export-Package y Bundle-ClassPath, las cuales, los desarrolladores usan para suministrar información descriptiva acerca de un bundle.

El siguiente fragmento de código muestra el contenido de un archivo de manifiesto para un bundle OSGi, compuesto por parejas con el formato nombre: valor, los que deben estar separados por dos puntos y un espacio.

- ✓ El nombre no distingue entre mayúsculas y minúsculas y puede contener caracteres alfanuméricos, subrayados, guión medio y bajo.
- ✓ Una sola línea nombre: valor no puede superar los 72 caracteres, si lo hace se continuará en la siguiente línea, para lo cual se debe empezar con un espacio en blanco, seguido del valor.

El framework tendrá que:

- ✓ Procesar la sección principal del manifiesto. Las secciones individuales del manifiesto solo son usadas durante la verificación de firmas del bundle.
- ✓ Ignorar las cabeceras del manifiesto que no se reconozcan. El desarrollador del bundle puede definir cabeceras adicionales como necesite.

2.2.3.2 Segunda Capa: Ciclo de Vida

La capa de gestión del ciclo de vida en OSGi permite que los paquetes se instalen, se lancen, se detengan y se desinstalen de forma dinámica, independientemente del ciclo de vida de la máquina virtual Java. La capa de ciclo de vida garantiza que los paquetes se inicien únicamente si sus dependencias están resueltas, lo que reduce el número de excepciones Class Not Found Exception en tiempo de ejecución. Si hay dependencias sin resolver, la infraestructura OSGi informa del problema y no lanza el paquete.

Cada paquete puede proporcionar una clase de activador de paquete, que se identifica en el manifiesto de paquete y que la infraestructura llama para iniciar y detener sucesos. (IBM, 2011).

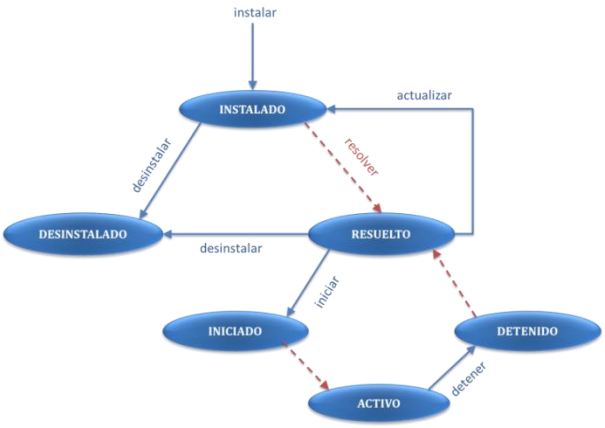

Figura 9: Ciclo de vida de un bundle

Cuando un bundle se instala por primera vez en el framework de OSGi, su estado es INSTALADO. Si todas las dependencias del bundle (paquetes importados o bundles requeridos) ya están disponibles en el framework, automáticamente pasará al estado RESUELTO. De lo contrario, permanecerá en estado de INSTALADO hasta que todas sus dependencias estén disponibles o hasta que el bundle se desinstale.

Si decide desinstalar un bundle, éste será removido del framework. Una vez que es removido, no quedará rastro de su existencia en el framework. Por lo tanto, es poco probable ver un bundle en el estado DESINSTALADO. Este estado es solamente un lugar en el que se colocan los bundles que ya no están disponibles.

Un bundle en el estado RESUELTO tiene todas sus dependencias disponibles, pero éste aún no se ha iniciado. El comando start inicia un bundle, cambiándolo a ACTIVO, a través del estado INICIADO. Del mismo modo, la emisión del comando stop en un bundle lo regresa al estado RESUELTO, por medio del estado de DETENIDO. Iniciar y parar un bundle es típicamente una actividad rápida. Esto significa que a pesar de que es posible ver un bundle en el estado INICIADO o DETENIDO, es poco probable que se puedan ver los bundles en uno de esos estados de transición.

Lo que no es evidente en el diagrama de estados es que algunas de las implementaciones del framework OSGi permitirán desinstalar un bundle activo. Pero incluso si el paquete se encuentra en estado ACTIVO cuando se desinstala, pasará del estado ACTIVO a RESUELTO a través del estado DETENIDO y luego a DESINSTALADO. Esta serie de

transiciones es importante porque puede ser necesario para el método stop() de la clase activator que se invoca antes de la desinstalación, de forma que el bundle puede liberar recursos y realizar otra actividad de cierre.

2.2.3.3 Tercera Capa: Servicios

La capa de servicios de OSGi admite intrínsecamente una arquitectura orientada a servicios mediante su componente de registro de servicios no duraderos. Los paquetes publican servicios en el registro de servicio y otros paquetes pueden descubrir estos desde el registro de servicio. Estos servicios son los medios primarios de colaboración entre paquetes. Un servicio OSGi es un Plain Old Java Object (POJO), publicado en el registro de servicio bajo uno o más nombres de interfaz, con metadatos opcionales almacenados como propiedades personalizadas (pares de nombre/valor).

Un paquete descubridor puede buscar un servicio en el registro de servicio por un nombre de interfaz y, potencialmente, puede filtrar los servicios que se buscan en función de las propiedades personalizadas.

Los servicios son completamente dinámicos y, generalmente, tienen el mismo ciclo de vida que el paquete que los proporciona (IBM, 2011).

2.2.3.3.1 Tipos de Dependencias (Eclipse, 2010)

En un medio ambiente OSGi, hay dos tipos de dependencias entre varios bundles: dependencias de tipo (type) y dependencias de servicios (service).

- ✓ **Dependencia de tipo (type dependency)**: Un bundle puede depender de un tipo exportado por otro bundle, creando así una dependencia de tipo. Estas dependencias se gestionan a través de las directivas Import-Package y Export-Package del manifiesto de un bundle OSGi.
- ✓ **Dependencias de servicios (Service dependency)**:Un bundle puede también publicar servicios (preferiblemente usando Spring DM) y otro bundle puede consumir esos servicios. Si dos bundles dependen de un mismo servicio, ambos se comunicarán de forma eficaz con ese objeto.

  De manera más específica, cualquier estado de ese servicio, se repartirá entre todos los clientes de ese servicio. Este tipo de arreglos es similar a la comúnmente interacción vista en un modelo cliente-servidor a través de mecanismos tales como

Servicios web y RMI (Java Remote Method Invocation) que es un mecanismo ofrecido por Java para invocar un método de manera remota.

2.2.4 Ventajas OSGi

2.2.4.1 Reducción de la Complejidad

Desarrollar con OSGi tecnología, implica el desarrollo de paquetes: los componentes de OSGi (bundles) esconden sus componentes internos de otros paquetes y se comunican a través de servicios bien definidos. Ocultamiento interno significa más libertad para cambiar más adelante.

Esto no sólo reduce el número de errores, sino que también hace más fácil desarrollar bundles, porque éstos si están correctamente dimensionados, implementan una funcionalidad a través de interfaces bien definidas.

2.2.4.2 Reutilización

El modelo de componentes de OSGi facilita el uso de componentes de terceros en una aplicación. Un número cada vez mayor de proyectos de código abierto proporciona sus JAR confeccionados para OSGi.

2.2.4.3 Adaptación al mundo real

El framework de OSGi es dinámico. Puede actualizar los paquetes sobre la marcha y los servicios pueden ir y venir. Los desarrolladores que utilizan tradicionalmente Java ven esto como una característica muy problemática y no ven la ventaja. Sin embargo, resulta que el mundo real es muy dinámico y tener servicios dinámicos que pueden ir y venir los hace un complemento perfecto para muchos escenarios del mundo real. Por ejemplo, un servicio puede modelar un dispositivo en la red. Si el dispositivo es detectado, el servicio está registrado. Si el dispositivo se va, el servicio deja de estar registrado.

Hay un sorprendente número de escenarios reales que responden a este modelo de servicios dinámicos. Las aplicaciones por lo tanto puede volver a utilizar el código que el registro de servicio brinda a través de primitivas (funciones como register, get, list y esperar a que los servicios aparezcan y desaparezcan) en su propio dominio. Esto no sólo

ahorra la escritura de código, sino que también proporciona una visibilidad global, herramientas de depuración, y más funcionalidades de la que se han puesto en práctica una solución dedicada. Escribir código en un entorno tan dinámico suena a pesadilla, pero afortunadamente, existe clases de soporte y frameworks que eliminan la mayoría, si no todo, el trabajo pesado.

2.2.4.4 Fácil despliegue

OSGi no es sólo un estándar para los componentes. También específica cómo los componentes se instalan y administran. Esta API ha sido utilizada por muchos bundles para proporcionar un agente de administración. Este agente de administración puede ser tan simple como un shell de comandos, o un protocolo de administración de interfaces muy complejo. La API de gestión normalizada hace que sea muy fácil de integrar OSGi en los sistemas existentes y futuros.

2.2.4.5 Actualizaciones dinámicas

El modelo de componentes de OSGi es un modelo dinámico. Los bundles se pueden instalar, iniciar, detener, actualizar y desinstalar sin la preocupación de que todo el sistema se caiga.

Muchos desarrolladores de Java no creen que esto se pueda hacer de forma fiable y por lo tanto, inicialmente no lo utilizan en tiempo de producción (cuando se despliega el sistema). Sin embargo, después de utilizar el modelo de componentes de OSGi, la mayoría se da cuenta de que en realidad funciona y reduce significativamente los tiempos de implementación.

2.2.4.6 Adaptable

El modelo de componentes de OSGi está diseñado desde sus bases para permitir la mescla y combinación de componentes.

Esto requiere que sus dependencias deben ser especificadas para habitar un ambiente donde sus dependencias opcionales no están siempre disponibles. El servicio de registro de OSGi es un registro dinámico en el que los paquetes pueden registrar, obtener y escuchar servicios.

Este modelo de servicio dinámico permite a los paquetes saber qué funciones están disponibles en el sistema y adaptar la funcionalidad que pueden proporcionar. Esto hace que el código sea más flexible y adaptable a los cambios.

2.2.4.7 Transparencia

La API de gestión proporciona acceso al estado interno de un bundle, así como a la forma en que se conecta a otros bundles. Por ejemplo, la mayoría de los frameworks proporcionan un intérprete de comandos que muestra este estado interno.

Partes de la aplicación pueden ser detenidas para depurar algún problema, o diagnosticar que bundles pueden ser iniciados parados o actualizados. En lugar de mirar a millones de líneas de archivos de log y demorosos periodos de reinicio del sistema cuando se corrige algún erros, las aplicaciones OSGi a menudo se puede depurar con un shell de comandos en tiempo real.

2.2.4.8 Versiones

La tecnología de OSGi resuelve el infierno de los archivos JAR, que es el problema que se genera cuando, por ejemplo, la librería A trabaja con la librería B que se encuentra en la versión 2, pero la librería C sólo puede funcionar con la librería B que se encuentra en la versión 3. En el estándar de Java, estos problemas son comunes y afectan el trabajo del desarrollador. En el ambiente OSGi, en cambio, todos los paquetes son cuidadosamente versionados y solamente los bundles que pueden colaborar, se conectan entre sí en el mismo classspace. Esto permite a ambos bundles A y C, funcionar con sus propias librerías.

2.2.4.9 Sencillo

El API de OSGi es sorprendentemente simple. El núcleo de la API es sólo un paquete de menos de 30 clases / interfaces. El núcleo de esta API es suficiente para escribir paquetes, instalarlos, iniciarlos, detenerlos, actualizarlos y desinstalarlos e incluye todas las clases de escuchadores y de seguridad. Hay muy pocos APIs que proporcionan tanta funcionalidad para un API tan pequeño como el de OSGi.

2.2.4.10 Pequeño

El Framework OSGi en su versión 4 se puede implementar en un fichero JAR de aproximadamente 300 KB. Esta es una pequeña sobrecarga de la cantidad de

funcionalidad que se añade a una aplicación mediante la inclusión de OSGi. OSGi por lo tanto se ejecuta en una amplia gama de dispositivos: desde muy pequeños, pasando por pequeños, a mainframes. Sólo se pide un mínimo de espacio para ejecutarlo en la máquina virtual de Java.

2.2.4.11 Velocidad

Una de las principales responsabilidades del framework OSGi, es cargar las clases de están contenidas en los bundles. En el estándar de Java, los archivos JAR son completamente visibles y se colocan en una lista. Buscar una clase requiere buscar a través de esta lista, que a menudo es muy larga. Por el contrario, OSGi pre-enlaza los bundles y sabe exactamente lo que cada bundle proporciona a la clase, por lo que no es necesario realizar búsqueda alguna. Esta falta de búsqueda es un factor significativo en el incremento de la velocidad al arranque.

2.2.4.12 Lazy (Baja actividad, demora)

Lazy en software es en cierto modo buena y la tecnología OSGI tiene muchos mecanismos para hacer las cosas sólo cuando son realmente necesarios. Por ejemplo, los bundles se pueden iniciar tempranamente, pero también pueden ser configurados para iniciar solamente cuando otro paquete los requiera. Los servicios pueden ser registrados, pero sólo se crean cuando se los necesitan. Las especificaciones se han optimizado varias veces para permitir este tipo de escenarios de baja actividad que pueden ahorrar enormes costos en tiempo de ejecución.

2.2.4.13 Seguridad

En el fondo Java tiene un muy poderoso modelo de seguridad, pero ha resultado muy difícil de configurar en la práctica. El resultado es que las aplicaciones de Java más seguras se están ejecutando con una elección binaria: sin seguridad o con capacidades muy limitadas. El modelo de seguridad de OSGi, aprovecha el modelo de seguridad de grano fino, pero mejora la facilidad de uso. En general, OSGi proporciona probablemente uno de los entornos de aplicaciones más seguras que todavía se puede utilizar por debajo de plataformas de hardware de computación.

2.2.4.14 Flexibilidad

Muchos frameworks se hacen cargo de la totalidad de la máquina virtual, esto sólo permiten ejecutar una instancia en una máquina virtual. La flexibilidad de las especificaciones de OSGi se demuestra por la forma en que puede incluso funcionar dentro de un servidor de aplicaciones J2EE. Muchos desarrolladores desean correr OSGi pero sus empresas no les permiten desplegar JARs normales. En su lugar, se incluyó un framework OSGi en su archivo WAR y cargaron sus bundles desde el sistema de archivos o en la red. OSGi es tan flexible que un servidor de aplicaciones puede alojar múltiples frameworks OSGi.

2.2.4.15 Ejecutable en cualquier entorno

El objetivo original de Java era correr en cualquier entorno. Obviamente, no es posible ejecutar todo el código en todas partes debido a las capacidades de la maquina virtual de Java. Una máquina virtual en un teléfono móvil, es probable que no soporte las mismas bibliotecas como un mainframe de IBM que ejecuta una aplicación bancaria.

Hay dos temas a cuidar. En primer lugar, la API de OSGi no debe utilizar las clases que no están disponibles en todos los ambientes. En segundo lugar, un paquete no debe comenzar si contiene código que no está disponible en el entorno de ejecución. Estas dos opciones han sido tomadas en cuenta en la especificación de OSGi.

2.2.4.16 Ampliamente usado

La especificación de OSGi se inició en el mercado de domótica integrada, pero desde 1998 ha sido ampliamente utilizada en muchas industrias: automotriz, telefonía móvil, automatización industrial, gateways y routers, los gateways privados de sucursales, telefonía de línea fija, y muchos más. Desde 2003, el popular entorno de desarrollo integrado Eclipse se ejecuta sobre la tecnología de OSGi, y proporciona un amplio soporte para el desarrollo en conjunto.

En los últimos años, OSGi ha sido adoptada por los desarrolladores de las empresas. Los desarrolladores de Eclipse descubrieron no solo el poder de la tecnología de OSGi, sino también el poder de su infraestructura. La empresa Spring ayudó a popularizar esta tecnología mediante la creación de una extensión específica para OSGi. Hoy en día la tecnología de OSGi se puede encontrar como base de aplicaciones ampliamente usadas a

nivel empresarial como Websphere de IBM, Spring Source Application Server, Oracle (anteriormente BEA) Weblogic, Sun Glass Fish y Red Hat JBoss.

2.2.4.17 Apoyo empresarial

OSGi cuenta con el apoyo de algunas de las mayores compañías de computación, como de un conjunto muy diverso de industrias. Al momento de redactar este documento el número de empresas que daban apoyo a OSGi suman mas de 151 empresas, los nombres de todas estas empresas se las puede encontrar en la página web: http://goo.gl/w99BJ. Pero se puede mencionar entre las más importantes a Oracle, IBM, Samsung, Nokia, Motorola, Siemens, Hitachi, Redhat, Ericsson, VMware entre otras más.

2.2.5 Tecnologías

2.2.5.1 Websphere IBM

OSGi es una tecnología modular para Java que se ha utilizado internamente en IBM® WebSphere® Application Server y la plataforma de desarrollo integrado Eclipse durante muchos años, y fue diseñada para permitir el desarrollo y la ejecución de aplicaciones dinámicas, modulares y extensibles. Estas dos IDEs permiten simplificar drásticamente el desarrollo, montaje y despliegue de aplicaciones modulares basadas en OSGi.

2.2.5.1.1 Características del paquete para aplicaciones OSGi

El servidor de aplicaciones WebSphere para OSGi brinda la oportunidad de desarrollar y desplegar aplicaciones empresariales de forma modular, introduciendo repositorios configurables de bundles OSGi dentro del proceso administrativo de WebSphere. Esto habilita a los bundles comunes a ser un factor fuera de cada uno de los archivos de las aplicaciones empresariales individuales y puede ser gestionado de forma centralizada en un repositorio de bundles. Múltiples versiones de bundles pueden ser gestionados en un repositorio, y la versión apropiada asociada con las aplicaciones empresariales individuales pueden especificarse en los metadatos para estas aplicaciones.

Las razones por las que optar por desarrollar e implementar una aplicación OSGi son:

- ✓ Múltiples versiones de clases se pueden cargar de forma simultánea en la misma aplicación.

- ✓ Aplicaciones implementadas pueden ser administrativamente actualizadas de forma modular, al nivel de bundle.
- ✓ En tiempo de desarrollo, los proyectos empresariales de OSGi en IBM Rational® Application Developer, hacen cumplir las reglas de visibilidad de OSGi, para que los proyectos sólo puedan acceder a los paquetes de otros proyectos que explícitamente declaran como parte de los aspectos externos del proyecto, proporcionando con esto, un ambiente de soporte para desarrollar mejores prácticas.
- ✓ Las aplicaciones pueden ser diseñadas para actualizaciones extensibles y dinámicas a través de la utilización de servicios OSGi.
- ✓ En tiempo de ejecución, las aplicaciones sólo se iniciarán correctamente si todas sus dependencias se puede resolver. Con ello se consigue la reducción de apariciones de Class Not Found Exceptions, mientras una aplicación está procesando una carga de trabajo.

Los beneficios del uso de OSGi es cada vez más evidente a medida que la aplicación crece en complejidad, o la suite de aplicaciones implementadas crece en tamaño, aumentando el desafío de administrar las actualizaciones de los módulos de las aplicaciones y las librerías de servicios públicos que utilizan.

2.2.5.2 Virgo Server

Desde hace uno años el servidor de módulos dinámicos cuyo soporte estaba a cargo de SpringSource paso a cargo de la empresa Eclipse RT y su nombre cambio por Virgo Server esto para dar más impulso al desarrollo de aplicaciones con la utilización de OSGi.

El servidor Virgo está diseñado para correr aplicaciones empresariales Java y aplicaciones Spring con un alto grado de flexibilidad y fiabilidad. Ofrece una plataforma simple pero completa, para desarrollar, implementar y generar servicios y aplicaciones empresariales Java.

El núcleo del servidor Virgo es compatible con los conceptos básicos de Virgo y no está predispuesto hacia el servidor web, lo que permite que otro tipo de servidor pueda ser creado. El núcleo también puede utilizarse de forma autónoma como una plataforma para aplicaciones OSGi. En tiempo de ejecución del servidor, puede ser fácilmente construido en la parte superior del núcleo, mediante la implementación de bundles adecuados.

Virgo posee las siguientes características:

- ✓ Consola de administración.- Esta despliega y gestiona los bundles, examina los volcados de diagnóstico y examina los enlaces entre bundles, ya sea en tiempo de ejecución o de un error de resolución.
- ✓ Extensión para la consola de Equinox.- Gestiona el servidor Virgo y despliega los artefactos.
- ✓ Planificación.- Define los objetos que componen una aplicación, opcionalmente hace atómica a la aplicación, para enlazar los ciclos de vida de los artefactos cohesivos, y permite delimitarlos de otras aplicaciones.
- ✓ Aprovisionamiento.- Suministran automáticamente las dependencias de una aplicación, incluyendo bundles, archivos planos (PAR), y configuraciones, desde los repositorios locales y remotos.
- ✓ Contenedor web.- Soporta archivos WAR con todas sus dependencias en WEB-INF/lib, y paquetes de aplicaciones web, que importan sus dependencias a través de metadatos con manifiesto OSGi, a través de la implementación de referencia de la especificación del contenedor web para OSGi basado en Apache Tomcat embebido y configurado con el estándar server.xml de Tomcat.
- ✓ Región usuario.- Aísla el núcleo de las aplicaciones instaladas por el usuario, y permite a los administradores centrarse en los artefactos y dependencias de la aplicación, sin ver los módulos utilizados por el núcleo de Virgo.
- ✓ Spring.- Paquetes de Spring para Virgo, que pueden ser fácilmente configurados para diferentes versiones del framework.
- ✓ Despliegue en caliente.- Implementa artefactos hacia Virgo mediante la copia de estos en el directorio pickup ya sea en forma de bundles únicos o en forma de archivos PAR para desplegar uno o varios bundles a la vez, esto como una alternativa a la implementación a través de la consola de administración.

2.2.5.3 Uso de Spring en un entorno OSGi con Spring DM para Eclipse

2.2.5.3.1 Spring DM

Spring DM proporciona una potente solución modular para el desarrollo de aplicaciones basadas en Spring que se pueden implementar en un entorno de ejecución de OSGi.

El objetivo de esta tecnología es hacer el trabajo de Spring y OSGi de una manera sencilla, así como enfrentar con éxito las limitaciones de las dos tecnologías.

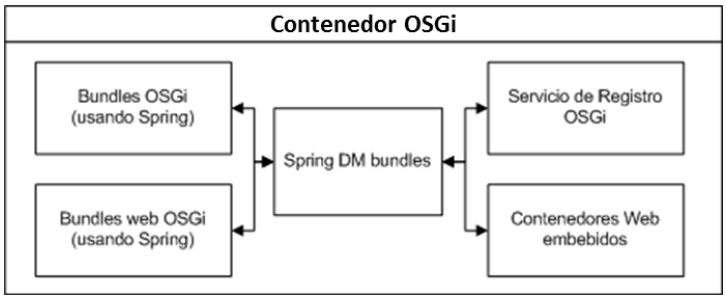

Figura 10 Contenedor OSGi

Spring DM se compone de un conjunto de bundles OSGi (Figura 10), que se convierten en parte de la infraestructura cuando se instalan en un contenedor OSGi. A continuación, puede administrar un contexto de aplicación Spring para paquetes, o incluso iniciar un contenedor web en el contenedor OSGi.

A pesar de los beneficios del Framework, Spring adolece de varios inconvenientes en los casos de uso particular. El primero ocurre cuando se trata de ampliar las aplicaciones grandes y complejas, en este caso, una gran cantidad de beans necesitan ser configurados dentro del recipiente Spring, haciendo su configuración más difícil de mantener. Spring sufre de una falta de modularidad y no ofrece un apoyo real para mejorarlo, por ejemplo, no hay manera de limitar la visibilidad de un bean en Spring, y cualquier otro bean se puede utilizar, y puede ser visto desde el contexto de la aplicación incluso si no debe ser utilizado directamente.

Aunque se puede dividir su configuración en varios archivos XML, todos ellos sirven para configurar un contexto de aplicación única, y la alternativa de la integración de varios contenedores Spring puede ser difícil de lograr.

Otra limitación de Spring es la naturaleza estática de la gráfica de dependencia configurada mediante la inyección de dependencia. Tan pronto como el bean "A" se inyecta en el bean B, el bean B depende de A. Una aplicación puede tener cientos de beans, y las dependencias entre los beans forman un gráfico de dependencias. La limitación se debe a que los vínculos entre los beans se crearán una vez, principalmente en la puesta en marcha del contenedor, y no hay soporte integrado para ponerlos al día en tiempo de ejecución. Para volver a cargar el gráfico de la dependencia y adoptar todas las actualizaciones, se debe reiniciar toda la aplicación.

Este problema de acoplamiento estático se ve agravado por problemas con el orden de instancias de los beans. Spring calcula un gráfico de dependencias complejas para determinar el orden en que los beans se crearán en una instancia. Este cálculo es interno a

Spring, y los desarrolladores de aplicaciones no pueden poner sus manos en él. Si algo va mal, la depuración puede ser especialmente compleja, sobre todo cuando se tiene muchas dependencias.

Además, hay muy poco apoyo para las dependencias circulares, poniendo una carga aún más pesada a los desarrolladores, al configurar sus beans correctamente.

Por otro lado, la especificación del núcleo de OSGi no proporciona ningún apoyo para los patrones o herramientas en el diseño e implementación de los paquetes. Esto es por diseño, dejando al programador libre de elegir una arquitectura adecuada y el framework de trabajo.

Usando un framework como Spring se puede aumentar el poder de la plataforma, ya que permite el uso de la inyección de dependencia, AOP, y otros paradigmas modernos. Pero aquí hay problemas evidentes en la gestión de los ciclos de vida diferentes de Spring y los paquetes OSGi.

Otro desafío en el uso de OSGi se refiere a la configuración explícita y el comportamiento dinámico de los servicios. Los servicios son esenciales en la creación de cualquier aplicación OSGi porque ellos son la única forma segura de acceder a las funciones a través de fronteras de paquetes. Pero el uso del servicio de gestión de la API es tedioso, y la gestión de la naturaleza dinámica de los servicios en código de usuario es muy propensa a errores y es aquí en donde entra Spring DM para facilitar las cosas.

2.2.5.3.2 Incorporación de Spring dentro de un contenedor OSGi

Spring DM es un framework, hecho de un conjunto de paquetes OSGi que se pueden desplegar en cualquier contenedor OSGi. Estos paquetes no traen cualquiera de las funciones de negocio para una aplicación OSGi, sino que apuntan a mirar otros paquetes y la creación de contextos de aplicación Spring para ellos. Esta es la razón por la que a Spring DM a veces se le llama un puente entre las aplicaciones de Spring y OSGi.

2.2.5.3.3 Gestión de Spring con Spring DM

En una aplicación Spring clásica, se utiliza un contenedor Spring dedicado, y el framework está configurado a partir de sus contextos envolventes (desde el entorno classpath o web). El framework también proporciona una relación jerárquica entre los contextos de aplicación mientras los vincula. Con Spring DM, las cosas funcionan de forma ligeramente diferente, un contenedor Spring dedicado está asociado con cada

paquete de Spring DM accionado. Cada contenedor es independiente del resto y es interno a su paquete de inclusión; lógicamente con recipientes separados, ya que cada paquete tiene su propio ciclo de vida y puede aparecer o desaparecer en cualquier momento.

Spring DM gestiona todos estos contenedores en una forma conveniente y no intrusivo. Ningún código o configuración se requiere dentro de un paquete para configurar, iniciar o destruir a un contenedor de Spring, todo lo que se requiere es que los paquetes Spring DM estén instalados en el contenedor OSGi, y que el paquete incluye una aplicación Spring con su archivo de configuración. El paquete Spring DM a cargo de la gestión de paquetes Spring alimentados, se llama extensor Spring DM.

Spring DM controla el ciclo de vida del paquete a fin de determinar cuándo hay que activar las acciones apropiadas. Una vez que Spring DM se ha implementado dentro de un contenedor OSGi, puede detectar automáticamente los componentes de Spring sobre la base de la presencia del archivo de la aplicación de configuración de Spring, así como a través de ciertas cabeceras especializadas en el archivo de manifiesto del paquete.

Cuando un paquete con la configuración adecuada se ha iniciado, un contenedor Spring también está configurado y embebido dentro del componente. De esta manera, Spring DM implementa el patrón extensor.

2.2.5.3.4 OSGi. Apoyo de servicio en Spring DM

Además de gestionar la creación de contenedores Spring, Spring DM también proporciona una manera conveniente para vincular los paquetes usando el servicio de registro de OSGi. En cualquier aplicación OSGi, el simple uso de las importaciones y exportaciones del paquete no es suficiente, ya que éstos sólo especifican el comportamiento de los vínculos de clases dentro de la aplicación. No hacen nada para gestionar el ciclo de vida de instancias de objetos creados a partir de estas clases; esto es lo que hace que el registro de servicios OSGi sea tan esencial, pues proporciona un límite bien definido a través del cual, las instancias de objetos se pueden acceder mientras se mantiene la clase adecuada y los límites del ciclo de vida.

El corazón de esta configuración es un espacio de nombres XML exclusivo de Spring, gracias a este mecanismo los POJOs simples pueden ser expuestos como servicios OSGi,

y estos servicios OSGi se puede inyectar en beans de Spring regulares mediante el uso de los elementos apropiados.

2.2.5.3.5 Beneficios de Spring DM para aplicaciones OSGi

Spring DM permite que las aplicaciones basadas en Spring puedan implementarse como paquetes OSGi, esto permite modular la aplicación real, sin dejar de aprovechar el poder del Framework de Spring. Spring DM también proporciona una manera fácil de vincular los componentes de forma declarativa mediante el suministro y consumo de servicios OSGi, todos estos aspectos permiten que las aplicaciones tomen ventaja de la naturaleza dinámica de OSGi.

Spring DM aborda varios aspectos, con el fin de hacer más fácil de usar Spring dentro de un contenedor OSGi:

- ✓ Hace cumplir los límites de módulos y añade la dinámica en tiempo de ejecución.
- ✓ Proporciona servicios relacionados con la gestión del servicio y su uso.
- ✓ Maneja de manera transparente los aspectos dinámicos de OSGi.

Otro aspecto importante a entender, es que no están atados a una herramienta específica en el desarrollo de los componentes de OSGi, ni siquiera a Spring DM. Se puede interactuar con los paquetes existentes que utilizan una variedad de diferentes tecnologías. Debido a que los vínculos entre los paquetes se basan en el registro de servicio estándar OSGi, un consumidor de servicios no sabe y no le importa que tecnología se utilizó en la creación del servicio.

En es mismo espíritu del diseño del Framework de Spring, Spring DM proporciona un modelo de programación basado en las mejores prácticas y patrones que le permiten utilizar la tecnología OSGi en una forma óptima y eficiente.

CAPÍTULO III

METODOLOGÍA

3.1 TIPO DE ESTUDIO

3.1.1 Según el objeto de estudio

Investigación aplicada: La investigación aplicada, guarda íntima relación con la básica, pues depende de los descubrimientos y avances de la investigación básica y se enriquece con ellos, se caracteriza por su interés en la aplicación, utilización y consecuencias prácticas de los conocimientos. La investigación aplicada, busca el conocer para hacer, actuar, construir y modificar.

3.1.2 Según la fuente de información

Investigación bibliográfica: Conjunto de técnicas y estrategias que se emplean para localizar, identificar y acceder a aquellos documentos que contienen la información pertinente para la investigación.

3.1.3 Según las variables:

Descriptiva Aplicada: Mediante la descripción y análisis del objeto de estudio, determinar su forma de aplicación en un entorno real.

3.2 POBLACIÓN MUESTRA

Debido a que se trata de una investigación aplicada, la población se ha considerado como el conjunto de aspectos que se relacionan con los sistemas informáticos modulares, y, la eficiencia en su desarrollo, que son:

- ✓ Modularidad física
- ✓ Modularidad lógica
- ✓ Acoplamiento

- ✓ Cohesión
- ✓ Desplegabilidad
- ✓ De módulos
- ✓ De servicios
- ✓ Intercambiabilidad
- ✓ De integración
- ✓ Administración en tiempo de ejecución
- ✓ Mantenibilidad

Se trabajará con toda la población=10

3.3 OPERACIONALIZACIÓN DE VARIABLES

Tabla 1: Operacionalización de Variables

Variable	Tipo	Definición Conceptual	Dimensión	Indicadores
La arquitectura OSGi	Independiente	La tecnología OSGI proporciona los estándares más básicos para el lenguaje de programación Java, que permite a las aplicaciones ser construidas desde componentes pequeños, reusables y colaborativos.	Análisis	Análisis descriptivo
Eficiencia en el desarrollo de sistemas informáticos modulares basados en servicios	Dependiente	La modularización posibilita un alto grado de cohesión entre sistemas informáticos permitiendo la adaptabilidad a cambios futuros. El objetivo será, por tanto, maximizar la eficiencia en el desarrollo de sistemas.	Modularidad Reusabilidad Complejidad	✓ Modularidad física ✓ Modularidad lógica ✓ Acoplamiento ✓ Cohesión ✓ Desplegabilidad ✓ De módulos ✓ De servicios ✓ Intercambiabilidad ✓ De integración ✓ Administración en tiempo de ejecución ✓ Mantenibilidad

Los presentes indicadores son tomados en base a las condiciones o características que deben cumplir un sistema para que pueda ser considerado modular según Walls, Craig en su libro*Modular Java, Creating flexible applications with OSGi and Spring.*

3.3.1 Indicador: Modularidad

La modularización es la capacidad de romper o dividir un sistema de software en módulos, los que operan de forma independiente uno del otro, pero aún pueden combinarse para hacer un trabajo útil, permitiendo que el sistema entero sea mucho más fácil de entender y mantener.

3.3.1.1 Parámetro: Modularidad física

Se refiere a como el código (el diseño lógico) es empaquetado y/o puesto a disposición para su despliegue. Otros aspectos del diseño físico son el seleccionar qué clases pertenecen a una u otra unidad física, también el administrar las relaciones entre las unidades lógicas.

3.3.1.2 Parámetro: Modularidad lógica

El diseño lógico define el cómo se construye el sistema en sí, si se usarán clases, operadores, interfaces, patrones de diseño, que métodos y atributos se usaran y en que clases se las utilizará.

3.3.1.3 Parámetro: Acoplamiento

El acoplamiento mide el grado de relacionamiento de un módulo con los demás. A menor acoplamiento, mejor: el módulo en cuestión será más sencillo de diseñar, programar, probar y mantener.

3.3.1.4 Parámetro: Cohesión

La cohesión tiene que ver con que cada módulo del sistema se refiera a un único proceso o entidad. A mayor cohesión, mejor: el módulo en cuestión será más sencillo de diseñar, programar, probar y mantener.

3.3.1.5 Parámetro: Desplegabilidad

Los módulos son una unidad de despliegue. A diferencia de otras entidades de software como las clases y paquetes, un módulo es una unidad discreta de la implementación, que puede ser desplegado junto a otros módulos. En este sentido, los módulos representan algo más físico que las clases o paquetes, los cuales son unidades de software intangibles.

3.3.2 Indicador: Reusabilidad

Dependiendo del alcance de un módulo, se puede utilizar uno diseñado para una aplicación y reusarlo en una aplicación totalmente diferente. A mayor escala es incluso concebible que una sección de módulos usados en una aplicación puedan ser re-ensamblados en un contexto diferente.

3.3.2.1 Parámetro: De módulos

La reusabilidad es la capacidad de reutilización de un sistema o partes de él (módulos), es decir hasta qué punto se puede volver a emplear un módulo en otras aplicaciones, en relación al empaquetamiento y alcance de las funciones que realiza.

3.3.2.2 Parámetro: De servicios

Varios EJBs o aplicaciones web pueden usar exactamente el mismo objeto como servicio.

3.3.2.3 Parámetro: Intercambiabilidad

Si cada módulo en una aplicación es conocido solamente por su interfaz pública y no por su implementación interna, entonces será fácil intercambiar un módulo con otro, siempre y cuando ambos tengan la misma interfaz pública, aunque su implementación sea diferente. Al separar la implementación de la solución de cada dominio, es posible utilizar esta implementación en un contexto distinto.

3.3.3 Indicador: Complejidad

Un sistema complejo en arquitectura, codificación y/o funcionalidad, resulta difícil de mantener (no solo por el aspecto técnico, sino también por su alcance) y por tanto de ser adaptado a las necesidades y expectativas del usuario.

3.3.3.1 Parámetro: De integración

Una segmentación bien definida del proyecto, asegura la modularidad del sistema. Cada tarea constituye un módulo independiente, un programa distinto.

En el momento de la implementación de cada módulo y si sus entradas y salidas están bien definidas, no hay confusión en la interacción de la interfaz con otros módulos del sistema. En tiempo de prueba, la integridad del módulo es testeada de forma independiente; hay pocos problemas de programación en la sincronización de la realización de varias tareas antes que pueda iniciar la comprobación.

3.3.3.2 Parámetro: Administración en tiempo de ejecución

En tiempo de ejecución la administración se utiliza únicamente para aplicaciones que lo requieren, y pueden implementarse actualizaciones, sin necesidad de realizar una reinstalación completa del sistema. De esa forma se reducen los tiempos de arranque, el consumo de memoria y los requisitos de espacio en disco.

3.3.3.3 Parámetro: Mantenibilidad

La mantenibilidad debe formar parte del proceso de desarrollo del software, las técnicas utilizadas deben de intervenir lo menos posible.

3.4 PROCEDIMIENTOS

3.4.1 Sistemas a evaluar

- ✓ Sistemas que usan la arquitectura de OSGi
- ✓ Sistemas que no usan la arquitectura de OSGi (.NET)

3.4.2 Indicadores a Evaluar

- ✓ Modularidad
    - Modularidad física
    - Modularidad lógica
    - Acoplamiento
    - Cohesión
    - Desplegabilidad
- ✓ Reusabilidad
    - De módulos
    - De servicios
    - Intercambiabilidad
- ✓ Complejidad
    - De integración
    - Administración en tiempo de ejecución
    - Mantenibilidad

3.4.3 Procesamiento y Análisis

La comparación se usa para determinar y cuantificar las relaciones entre dos o más variables, al observar diferentes grupos que ya sea por elección o circunstancia están expuestos a tratamientos diferentes.

Para la realización de la evaluación se ha decidido utilizar una escala cuantitativa (escala concreta o continua) para evaluar cada una de las arquitecturas.

Tabla 2: Indicadores y Parámetros

Indicador	Parámetro	%	Justificación
Descomponibilidad	Modularidad física	10	Los sistemas deben ser fragmentables haciéndolos más fáciles de entender y mantener.
	Modularidad lógica	10	
	Acoplamiento	10	
	Cohesión	10	
	Desplegabilidad	10	
Reusabilidad	De módulos	10	Un módulo bien definido puede ser reutilizado en diferentes contextos sin que la modificación de su implementación afecte al resto del sistema
	De servicios	10	
	Intercambiabilidad	10	
Complejidad	De integración	10	No importa la complejidad de la implementación de un módulo, su interfaz pública permitirá la rápida y sencilla integración con el resto del sistema
	Administración en tiempo de ejecución	10	
	Escalabilidad	10	
Total		**100**	

La calificación de **Poco Eficiente** será para aquella arquitectura que puntúe bajo en los parámetros establecidos dentro de la variable citada en la comparación, **Medianamente eficiente** será la Arquitectura que maneja bien el indicador pero que no se comporta efectiva al ciento por ciento; mientras que la calificación **Eficiente** será dada a la Tecnología que cumpla con la variable definida y se acople cien por cien. Se ha determinado un rango de calificación para generalizar los resultados y es el siguiente:

Tabla 3. Rango de calificaciones

Peso	Equivalencia
1-3	Poco eficiente
4-7	Medianamente eficiente
8-10	Eficiente

3.4.4 Evaluación de los indicadores

3.4.4.1 Modularidad

3.4.4.1.1 Modularidad física

- ✓ *Sistemas que usan la arquitectura de OSGi*

  Los módulos son una única unidad de despliegue que se comunican a través de servicios, los cuales se publican en un componente de registro de servicios, bajo uno o más nombres de interfaz. En el momento de despliegue, se utiliza un classloader para cada módulo, lo que aumenta su seguridad al ocultar su implementación. Son sistemas altamente mantenibles y extensibles

- ✓ *Sistemas que no usan la arquitectura de OSGi*

  Dependencia de módulos que se comunican a través de servicios web, sockets por lo que son difíciles de mantener y extender. Tienen un único classloader para todos los módulos, haciendo que en tiempo de ejecución, pierdan sus límites de modularidad.

Tabla 4: Parámetro Modularidad Física

Parámetro	**Peso con OSGi**	**Peso sin OSGi**
Modularidad física	10.00	2.00

3.4.4.1.2 Modularidad lógica

- ✓ *Sistemas que usan la arquitectura de OSGi*

  Permite la separación de contextos de código a través de interfaces públicas o privadas y su respectiva implementación.

- ✓ *Sistemas que no usan la arquitectura de OSGi*

  Permite la separación de contextos de a código a través de interfaces públicas o privadas y su respectiva implementación

Tabla 5: Parámetro Modularidad Lógica

Parámetro	**Peso con OSGi**	**Peso sin OSGi**
Modularidad lógica	10.00	10.00

3.4.4.1.3 Acoplamiento

- ✓ *Sistemas que usan la arquitectura de OSGi*

Los módulos débilmente acoplados, dependen de otros módulos solo a través de abstracciones estables, desconociendo totalmente la implementación que hay por debajo, lo que provoca un bajo impacto cuando se cambia la implementación de un módulo.

- ✓ *Sistemas que no usan la arquitectura de OSGi*

  Presentan módulos altamente acoplados los que supone un alto impacto al cambiar su implementación, sin embargo, al aplicar técnicas como el de diseño, expuestos en el libro GoF, este acoplamiento puede disminuir.

Tabla 6: Parámetro Acoplamiento

Parámetro	**Peso con OSGi**	**Peso sin OSGi**
Débilmente acoplados	10.00	7.00

3.4.4.1.3 Cohesión

- ✓ *Sistemas que usan la arquitectura de OSGi*

  Permite la separación de los contextos, físicamente a través de módulos, disminuyendo la complejidad del sistema.

- ✓ *Sistemas que no usan la arquitectura de OSGi*

  Permite la separación de los contextos, físicamente a través de módulos, pero esto aumenta la complejidad del sistema.

Tabla 7: Parámetro Cohesión

Parámetro	**Peso con OSGi**	**Peso sin OSGi**
Altamente cohesivos	8.00	3.00

3.4.4.1.4 Total del indicador Modularidad

Tabla 8: Indicador Modularidad

Parámetro	**Peso con OSGi**	**Peso sin OSGi**
Modularidad física	10.00	2.00
Modularidad lógica	10.00	10.00
Acoplamiento	10.00	7.00
Cohesión	8.00	3.00
Promedio del indicador	**9.50**	**5.50**

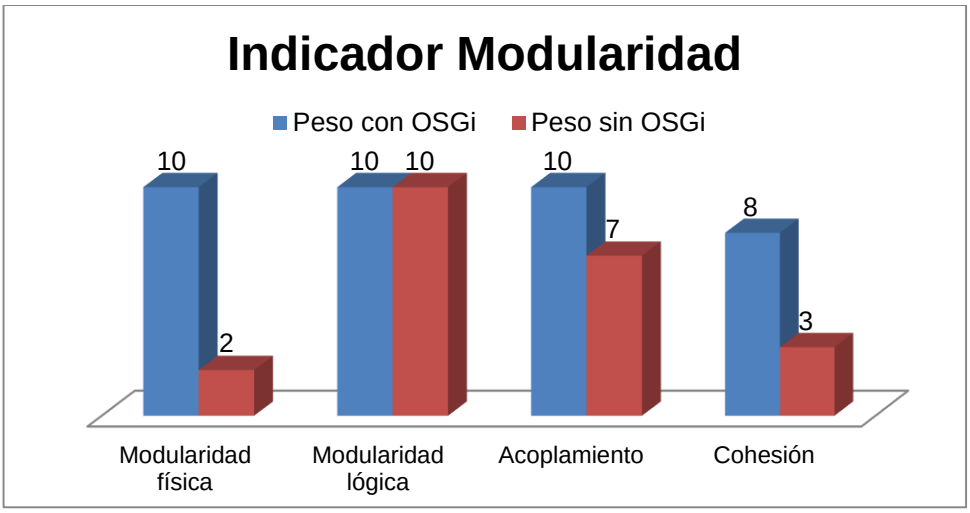

Gráfico 1: Indicador Modularidad

3.4.4.2 Reusabilidad

3.4.4.2.1 De módulos

- ✓ *Sistemas que usan la arquitectura de OSGi*

  Alta simplicidad de reutilización de uno o varios módulos en contextos diferentes, al utilizarse la referencia al módulo, publicado en el registro de servicios.

- ✓ *Sistemas que no usan la arquitectura de OSGi*

  Un módulo o varios módulos pueden ser reutilizados en contextos diferentes, sin embargo, este proceso suele ser complejo, ya que necesariamente su codificación debe ser adaptada

Tabla 9: Parámetro Reusabilidad de módulos

Parámetro	**Peso con OSGi**	**Peso sin OSGi**
Reusabilidad de módulos en contextos diferentes	9.00	3.00

3.4.4.2.2 De servicios

- ✓ *Sistemas que usan la arquitectura de OSGi*

  Proveen dinamismo a la aplicación, ya que son integrados durante su ejecución, los bundles pueden registrar escuchadores para ser informados si un servicio es iniciado o detenido a través del Service Binder.

- ✓ *Sistemas que no usan la arquitectura de OSGi*

  Suelen utilizar orientación a componentes, lo que disminuye el dinamismo y aumenta la complejidad en el desarrollo de la aplicación, ya que estos son ensamblados en el momento de construcción de la aplicación.

Tabla 10: Parámetro Reusabilidad de Servicios

Parámetro	**Peso con OSGi**	**Peso sin OSGi**
De servicios	8.00	3.00

3.4.4.2.3 Intercambiabilidad

- ✓ *Sistemas que usan la arquitectura de OSGi*

  Facilidad en la intercambiabilidad de un módulo con otro ya que únicamente se trabaja con las interfaces públicas y no con la implementación interna.

- ✓ *Sistemas que no usan la arquitectura de OSGi*

  Para cambiar un módulo por otro, se debe re-ensamblar la aplicación o cambiar la codificación de los servicios web que permiten la comunicación.

Tabla 11: Parámetro Intercambiabilidad

Parámetro	**Peso con OSGi**	**Peso sin OSGi**
Intercambiabilidad	9.00	6.00

3.4.4.2.4 Total del indicador Reusabilidad

Tabla 12: Indicador Reusabilidad

Parámetro	**Peso con OSGi**	**Peso sin OSGi**
De módulos	9.00	3.00
De servicios	8.00	3.00
Intercambiabilidad	9.00	6.00
Promedio del indicador	**8.67**	**4.00**

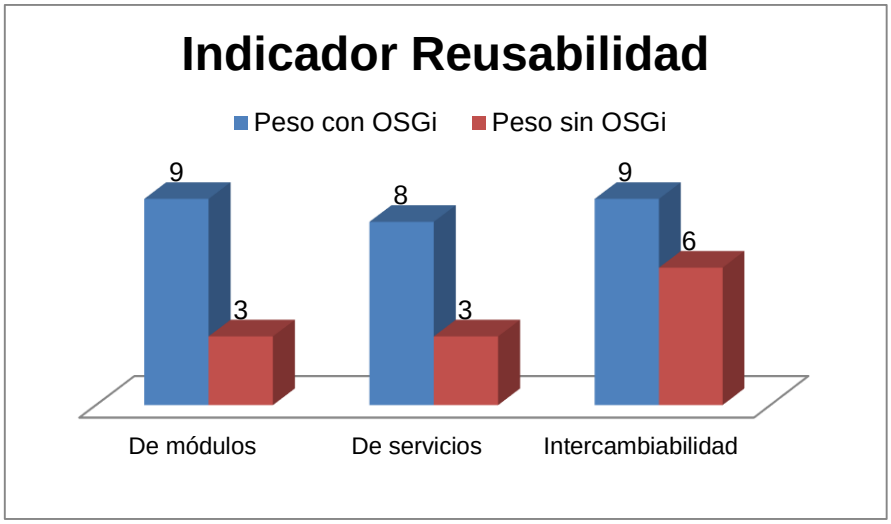

Gráfico 2: Indicador Reusabilidad

3.4.4.3 Complejidad

3.4.4.3.1 De integración

- ✓ *Sistemas que usan la arquitectura de OSGi*
  - Al utilizar servicios, provee una fácil integración entre módulos.
  - Al utilizar la Máquina Virtual de Java, es multiplataforma
  - Para comunicarse con aplicaciones desarrolladas en otros lenguajes de programación, se necesita implementar un middleware llamado RabbitMQ

- ✓ *Sistemas que no usan la arquitectura de OSGi*
  - Necesariamente se debe contar con un middleware para la comunicación entre módulos y aplicaciones, por lo que no es multiplataforma.

Tabla 13: Parámetro Complejidad de Integración

Parámetro	**Peso con OSGi**	**Peso sin OSGi**
Complejidad en la integración entre módulos	10.00	4.00
Multiplataforma	10.00	0.00
Complejidad en la integración con otras aplicaciones	5.00	5.00
Promedio del indicador	**8.30**	**3.00**

3.4.4.3.2 Administración en tiempo de ejecución

- ✓ *Sistemas que usan la arquitectura de OSGi*

  Los módulos pueden ser actualizados, iniciados o detenidos de forma individual.

- ✓ *Sistemas que no usan la arquitectura de OSGi*

  Para actualizar un módulo necesariamente la aplicación debe ser detenida en su totalidad.

Tabla 14: Parámetro complejidad de Administración en tiempo de ejecución

Parámetro	**Peso con OSGi**	**Peso sin OSGi**
Administración en tiempo de ejecución	9.00	0.00

3.4.4.3.3 Mantenibilidad

- ✓ *Sistemas que usan la arquitectura de OSGi*

  Los sistemas modulares, cuyos módulos son altamente cohesivos y débilmente acoplados, son más fáciles de modificar y extender, incrementando de esta manera la extensibilidad del sistema.

- ✓ *Sistemas que no usan la arquitectura de OSGi*

  El tener módulos poco cohesivos y altamente acoplados, dificulta las tareas de modificación y ampliación de los sistemas

Tabla 15: Parámetro Mantenibilidad

Parámetro	**Peso con OSGi**	**Peso sin OSGi**
Mantenibilidad	10.00	3.00

3.4.4.3.4 Total del indicador complejidad

Tabla 16: Indicador Complejidad

Parámetro	**Peso con OSGi**	**Peso sin OSGi**
Integración	8.30	3.00
Administración en tiempo de ejecución	9.00	0.00
Escalabilidad	10.00	3.00
Promedio del indicador	**9.10**	**2.00**

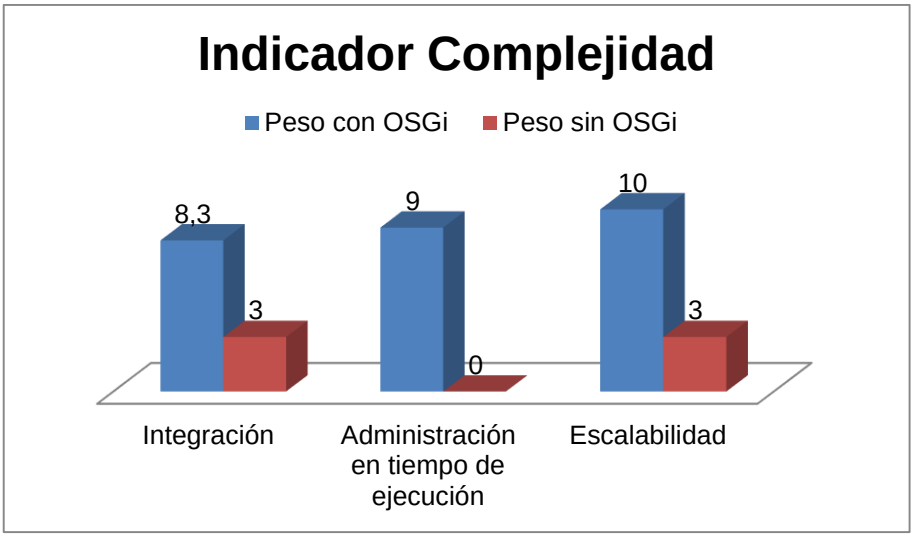

Gráfico 3: Indicador Complejidad

CAPÍTULO IV

RESULTADOS

4.1 Resultados Finales

Hasta este punto, los resultados obtenidos se basan en la investigación teórica de la Arquitectura OSGi y de los conocimientos previos de las otras arquitecturas que no utilizan OSGi para el desarrollo de aplicaciones modulares.

Además, estos resultados son producto de la aplicación de los indicadores descritos en páginas anteriores y que han sido analizados bajo ciertos parámetros, que permiten determinar, en forma comparativa, el desempeño de estas dos arquitecturas.

Este estudio analítico permite determinar con certeza, que la arquitectura OSGi facilita el desarrollo eficiente de sistemas modulares basados en servicios.

A continuación se presenta la comparación:

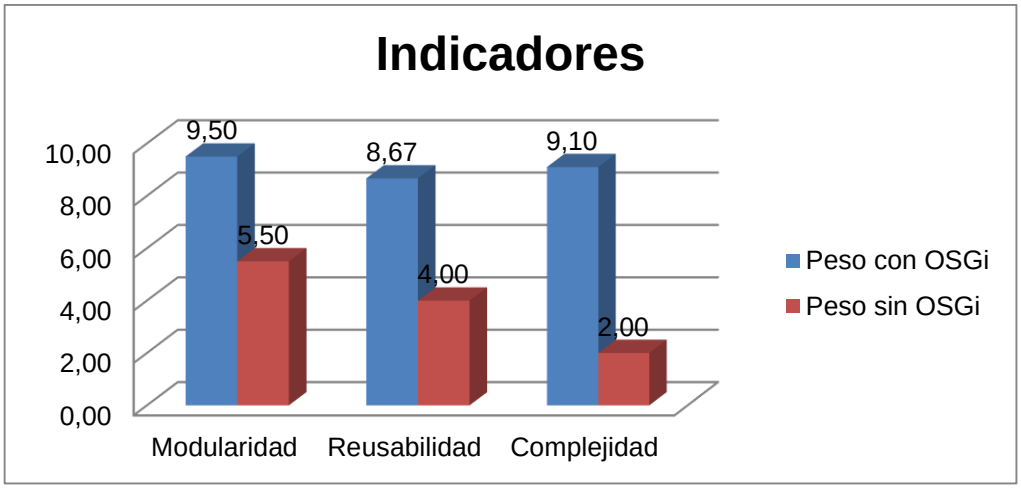

Gráfico 4: Comparación indicadores

4.2 Resumen de Resultados

Tabla 17: Resultados Indicadores y Parámetros

Indicadores	Parámetros	con OSGi	Sin OSGi	Promedio con OSGi	Porcentaje con OSGi	Promedio sin OSGi	Porcentaje sin OSGi
Modularidad	Modularidad física	10,00	2,00				
	Modularidad lógica	10,00	10,00				
	Acoplamiento	10,00	7,00				
	Cohesión	8,00	3,00				
				9,50	38,00	5,50	22,00
Reusabilidad	De módulos	9,00	3,00				
	De servicios	8,00	3,00				
	Intercambiabilidad	9,00	6,00				
				8,67	26,00	4,00	12,00
Complejidad	Integración	8,30	3,00				
	Administración en tiempo de ejecución	9,00	0,00				
	Escalabilidad	10,00	3,00				
				9,10	27,30	2,00	6,00
Resultados					**91,30**		**40,00**

4.3 Análisis de Resultados

Los resultados son presentados en términos de porcentajes, de la siguiente manera:

- Al aplicar la arquitectura OSGi en el desarrollo de sistemas modulares, se obtiene el 91.30 % equivalente a Excelente en la Tabla de rangos (ver. Tabla 4. Rango de calificaciones).
- La arquitectura sin OSGi tiene un total de 10% equivalente a Poco Eficiente en la Tabla de rangos (ver. Tabla 4. Rango de calificaciones).

En la siguiente gráfica se puede ver los indicadores evaluados, y las calificaciones obtenidas de acuerdo al porcentaje determinado para cada uno de ellos.

Con OSGi

Modularidad = 38.00%, Reusabilidad = 26.00%, Complejidad= 27.30%

Sin OSGi

Modularidad = 22.00%, Reusabilidad = 12.00%, Complejidad= 6.00%

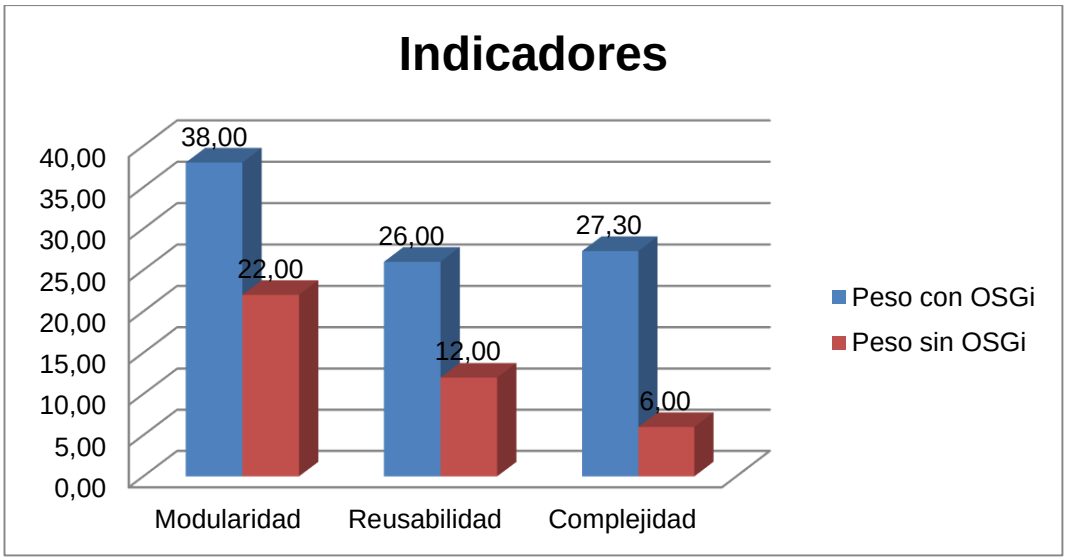

Gráfico 5. Indicadores evaluados, y las calificaciones obtenidas

4.4 Comprobación de la hipótesis de la investigación

En los temas abordados anteriormente, se han podido identificar las falencias de no utilizar la arquitectura OSGi en la construcción de sistemas informáticos modulares, y que constituyen fuertes razones para decidir por su utilización.

4.4.1 HIPÓTESIS GENERAL

La Arquitectura OSGi permitirá el desarrollo eficiente de un sistema informático modular basado en servicios, aplicado a un sistema parametrizable, que provea información relacionada con la evaluación de carreras en la Universidad Nacional de Chimborazo, en el periodo 2011-2012.

4.4.2 HIPÓTESIS ESPECÍFICAS:

4.4.2.1 Hipótesis de Investigación

Hi: El porcentaje de parámetros eficientes de los sistemas que usan la arquitectura de OSGi supera al porcentaje de parámetros eficientes de los sistemas que no lo usan.

4.4.2.2 Hipótesis Nula

Ho: El porcentaje de parámetros eficientes de los sistemas que usan la arquitectura de OSGi es igual al porcentaje de parámetros eficientes de los sistemas que no lo usan.

Tabla 18: Comparativo de eficiencia

Indicador	Parámetro	Peso con OSGi	Eficiente	Peso sin OSGi	Eficientes
Modularidad	Modularidad física	10,00	1	2,00	0
	Modularidad lógica	10,00	1	10,00	1
	Acoplamiento	10,00	1	7,00	0
	Cohesión	8,00	1	3,00	0
Reusabilidad	De módulos	9,00	1	3,00	0
	De servicios	8,00	1	3,00	0
	Intercambiabilidad	9,00	1	6,00	0
Complejidad	Integración	8,30	1	3,00	0
	Administración en tiempo de ejecución	9,00	1	0,00	0
	Escalabilidad	10,00	1	3,00	0
	Suma Total	**91,30**	**100%**	**40,00**	**10%**
	Promedio	**9,13**		**4,00**	

4.4.2.3 Planteamiento de las hipótesis específicas:

Hi: El porcentaje de parámetros eficientes de los sistemas que usan la arquitectura de OSGi supera al porcentaje de parámetros eficientes de los sistemas que no lo usan.

Ho: El porcentaje de parámetros eficientes de los sistemas que usan la arquitectura de OSGi es igual al porcentaje de parámetros eficientes de los sistemas que no lo usan.

4.4.2.3.1 Modelo estadístico

El modelo estadístico utilizado, corresponde a la diferencia de proporciones y la prueba a aplicar será la prueba z.

Hi: $\pi_C > \pi_S$

Ho: $\pi_C = \pi_S$

4.4.2.3.2 Nivel de significación

$\alpha = 0.05$, valor crítico $z = 1,96$

4.4.2.3.3 Criterio

Rechace la hipótesis nula si $z_c > 1,96$

4.4.2.3.4 Cálculos

Tabla 19: Porcentaje de parámetros eficientes

	Con OSGi	Sin OSGi
% Parámetros Eficientes	100%	10%

p_C= 1 **p_S**=0,1

n_1= 10 **n_2**=10

q_c= 1 - p_c **q_c**= 1 -1=0

q_s= 1 – p_s **q_s**= 1 -0,1=0,9

$$z_c = \frac{p_C - p_S}{\sqrt{\frac{p_C q_C}{n_1} + \frac{p_S q_S}{n_2}}}$$

Z_c = 9,48 donde,

p_c: Proporción muestral con OSGi

p_s: Proporción muestral sin OSGi

n_1, n_2: Tamaño de la muestra

Z_c: Distribución de la proporción

4.4.2.3.5 Decisión

Como $Z_c = 9.48 > 1.96$, se rechaza la hipótesis nula y se acepta la de investigación.

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

- ✓ A pesar de que la arquitectura OSGi fue creada para otros tipos de propósitos tales como la domótica, actualmente esta arquitectura confiere modularidad física al desarrollo de sistemas informáticos, lo que hace que estos sean altamente cohesivos, al tener su funcionalidad bien delimitada. Permite la reusabilidad de un módulo en un contexto distinto. Al estar débilmente acoplados al utilizar interfaces públicas que están separadas de su implementación, permite que un cambio realizado en ésta, no afecte al funcionamiento del módulo, ni de otros que dependan de él. Al cumplir con estas dos condiciones, el sistema se torna desplegable, administrable y verificable.

- ✓ La arquitectura OSGi permite un desarrollo en paralelo a través del encadenamiento de repositorios, lo que significa que cada grupo de trabajo puede desarrollar un módulo indistintamente de su ubicación física.

- ✓ La ventaja de tener módulos altamente cohesivos, es que a largo plazo aumenta la productividad en cuanto a la escalabilidad del sistema, gracias a la facilidad de aumentar y modificar funcionalidades.

- ✓ El éxito de la modularización radica en la delimitación de cada módulo, es decir que en un sistema modular, cada módulo debe cumplir con una función específica.

- ✓ Que los módulos sean verificables y testeables antes del despliegue, aumenta la seguridad no sólo en sí mismos, sino en el sistema en conjunto.

- ✓ La modularización reduce la complejidad de actualización de los sistemas en el tiempo. Con arquitecturas diferentes a OSGi, la actualización resulta compleja hasta

el grado de preferirse migrar a un nuevo sistema. OSGi permite aislar un módulo, lo que evita que los cambios realizados en este componente, afecten al resto del sistema.

- ✓ El desarrollo del sistema propuesto, facilita la gestión de los indicadores de la acreditación por carreras, en la Universidad Nacional de Chimborazo, no solo porque dinamiza los cálculos, sino porque además, permite realizar un seguimiento y respaldar la documentación utilizada como evidencia.

5.2 Recomendaciones

- ✓ Es necesario que la Universidad Nacional de Chimborazo actualice la bibliografía para que sirva de apoyo teórico en los posteriores temas de investigación.

- ✓ Aunque no es una prioridad para el sistema desarrollado en esta tesis, pero si lo es para posteriores implementaciones la necesaria actualización de los frameworks usados para el desarrollo de sistemas informáticos como son Spring DM, Hibernate, Java Server Pages, Spring Framework, Spring Security, para el efecto se sugiere revisar el manual técnico.

- ✓ Con fines de aumentar la seguridad y la legibilidad del código del sistema en el lado del cliente, se recomienda utilizar el framework Bootstrap, que es un modelo vista – controlador para Javascript.

- ✓ Por tratarse de un sistema que maneja gran cantidad de información procesada por otros sistemas implementados en la Universidad Nacional de Chimborazo, se recomienda la integración con estos, una opción es realizarlo a través de RabbitMQ que funciona como middleware entre OSGi y sistemas desarrollados en distintos lenguajes de programación.

- ✓ Para mejorar el rendimiento, la seguridad y aumentar el soporte a los clientes, proveyendo de disponibilidad al sistema, es necesario el encadenamiento de repositorios del Servidor Virgo.

CAPÍTULO VI

PROPUESTA

6.1 Título de la propuesta

Modelo de desarrollo de software para aplicaciones empresariales basadas en la arquitectura OSGi.

6.2 Introducción

Este capítulo detalla el modelo de desarrollo de software para aplicaciones empresariales basadas en la arquitectura OSGi, para el "SISTEMA PARAMETRIZABLE QUE PROVEA INFORMACIÓN RELACIONADA CON LA EVALUACIÓN DE CARRERAS EN LA UNIVERSIDAD NACIONAL DE CHIMBORAZO EN EL PERIODO 2011-2012", el cual ha sido elaborado tomando en cuenta las necesidades presentes en la Institución. Se basa en las directrices dadas por el documento: "Propuesta para el diseño de Sistemas", desarrollado por el Grupo de Diseño, adscrito a la Unidad de Desarrollo de la Dirección de Servicios de Información Administrativa de la Universidad de los Andes, Venezuela.

Con el transcurso del tiempo, se ha ido incrementando la importancia de contar con información confiable, íntegra y oportuna para lograr los objetivos estratégicos de las organizaciones.

La Universidad Nacional de Chimborazo en su proceso de Acreditación, tiene la desventaja de no disponer un sistema de información que le permita obtener todos y cada uno de los indicadores requeridos por el Consejo de Evaluación, Acreditación y Aseguramiento de la Calidad de la Educación Superior (CEAACES). Los criterios bajo los cuales están regidos los mismos, pueden cambiar en el tiempo, por lo cual el sistema propuesto debe ser extensible y debe permitir la inserción o eliminación de nuevos criterios.

El proceso de obtención de estos indicadores y la generación de estadísticas se lo realiza de manera manual, lo cual implica demora en la obtención de los mismos.

El desarrollo de un sistema modular basado en servicios, aumentará la adaptabilidad del sistema por su capacidad de inserción, actualización y eliminación de componentes en tiempo de ejecución, lo cual permitirá que el sistema se pueda adaptar a nuevos requerimientos en el transcurso del tiempo, a través del correcto almacenamiento y manejo de información, sumado a una adecuada generación de estadísticas y reportes. Este sistema no dependerá de ningún otro que haya sido implementado en la Universidad Nacional de Chimborazo, además, por su diseño ofrecerá la ventaja de que cada uno de los criterios de evaluación trabajen de una forma independiente pero a la vez colaborativa dentro de la misma aplicación.

6.3 Objetivos

6.3.1 Objetivos Generales

Mejorar el desarrollo de sistemas modulares, reduciendo la complejidad en la implementación y proveyendo al sistema de escalabilidad, reusabilidad, seguridad, y facilidad de uso y aprendizaje.

6.3.2 Objetivos Específicos

- Dividir el sistema en módulos que sean altamente cohesivos y débilmente acoplados.
- Diseñar interfaces públicas sencillas para la comunicación entre módulos.
- Aplicar reglas que permitan a los módulos comunicarse unos a otros a través del registro de servicios del framework de OSGi.
- Acoplar al framework de OSGi otros frameworks que permitan el desarrollo de, en este caso, una aplicación web.

6.4 Fundamentación Científico –Técnica

Contendrá una versión resumida y actualizada del estado del conocimiento en que se encuentra el tema específico de la propuesta.

6.5 Descripción de la propuesta

6.5.1 Análisis de requisitos

El análisis de cada uno de los requerimientos relacionados con los indicadores que se necesitan automatizar se encuentra en el documento: *"Especificación de Requerimientos de Software para el Sistema de Control de Indicadores en el proceso de evaluación de carreras con fines de acreditación*", documentos realizados por los desarrolladores de esta tesis. Lo que a continuación se detalla es un resumen del análisis de cada uno de los módulos del sistema y cómo cada requerimiento encaja en los módulos a desarrollarse.

6.5.1.1 Módulo Indicadores

Este módulo tiene por finalidad automatizar la administración de todos y cada uno de los indicadores presentes en el documento "Evidencias y Tereas que se derivan del Modelo General para la Evaluación de Carreras con fines de Acreditación en el período 2011 - 2012". Este control se lo hace actualmente en forma manual y se lleva un registro en Excel de los resultados para los datos relacionados con los indicadores, cómo son las evidencias que cada indicador debe cumplir.

Este módulo tiene como finalidad llevar un control de los indicadores, sus cálculos, ponderaciones, tareas, evidencias y responsables de gestionar la información de cada indicador. Es necesario mencionar que a todas las Facultades se las evalúa con los mismos indicadores, pero los cálculos y resultados de cada indicador es diferente por cada Facultad. Estos cálculos y resultados se los debe realizar cada cierto tiempo, siendo lo ideal una vez por semestre. Cada indicador tendrá su respectiva ponderación, evidencia, y resultados.

Además, este módulo generara informes o reportes del estado de todos o de cada uno de los indicadores. Esta información será de gran utilidad para la Facultad que se esté evaluando, ya que se conocerá en qué punto se está fallando y así poder realizar los ajustes necesarios a fin de obtener los resultados que se espera.

En el siguiente gráfico podremos ver cuáles son los casos de uso que intervienen en este módulo y quiénes son los actores.

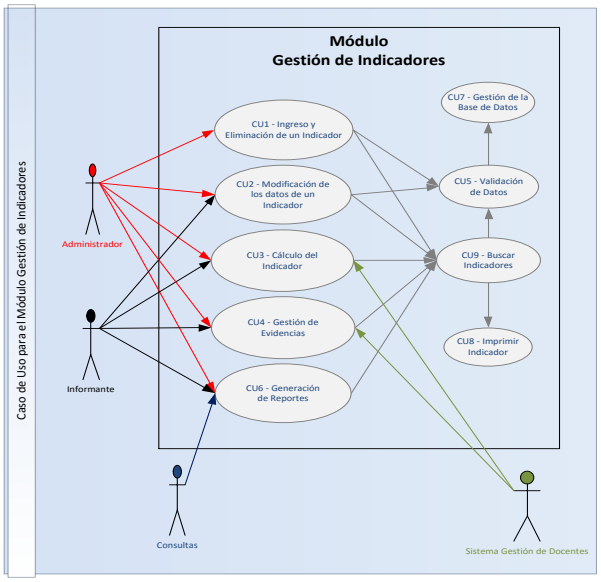

Figura 11. Caso de uso para el Módulo Gestión de Indicadores

El análisis de cada uno de los casos de uso presentados en la Ilustración 1, se analiza en el documento: "Ingeniería de software para el sistema para el control de indicadores en el proceso de evaluación de carreras con fines de acreditación" adjunto en un archivo digital.

6.5.1.2 División del sistema en módulos

La ventaja de utilizar OSGi, radica en que permite dividir un sistema grande en partes pequeñas llamadas módulos, que sean altamente cohesivos y débilmente acoplados, incrementando así la facilidad en el desarrollo del sistema. Tomando esto como punto de partida, podemos dividir el sistema Gestión de indicadores mostrado en la Ilustración 1, en módulos que cumplan con las características de OSGi. Así el sistema quedaría dividido de la siguiente manera:

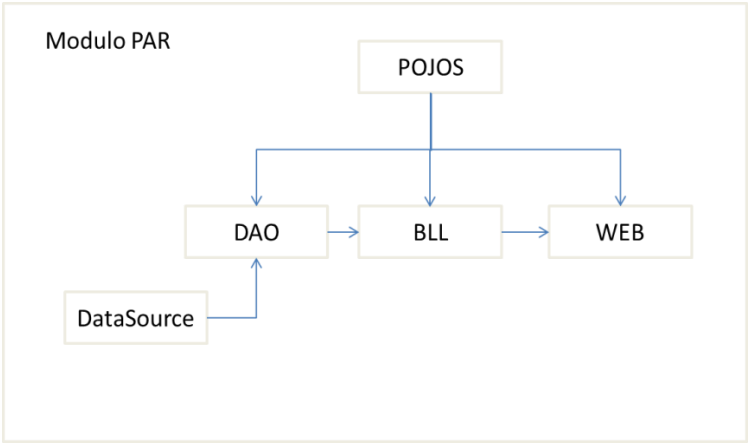

Figura 12. División del sistema en módulos

6.5.1.2.1 Módulo Pojos

Este módulo tiene la única función de clonar el diseño de la base de datos y modelarlo en forma de clases u objetos (pojos) los cuales luego serán convertidos a un lenguaje SQL para la manipulación de datos que se vaya a almacenar en cualquier RDBMS que se deseé utilizar y esto se lo hace sin la necesidad de cambiar nada de código compatible con el RDBMS que se esté usando.

6.5.1.2.2 Módulo DataSource

Este módulo tiene la función de administrar las conexiones a cualquier RBDMS del que se disponga el driver que se utilizará para la comunicación. Este módulo puede contener uno o varias cadenas de conexión a uno o varios RDBMS y puede ser utilizado por uno o varios módulos al mismo tiempo.

6.5.1.2.3 Módulo DAO

Este módulo se encarga de gestionar el ingreso, modificación, eliminación, búsquedas de datos. Utiliza los módulos DAO y DataSource para su funcionamiento.

6.5.1.2.4 Módulo BLL

Este módulo se encarga de pasarela entre en módulo web y el módulo DAO, haciendo posible el acceso a los datos únicamente a través de este módulo, también sirve como medio de verificación de y validación de datos brindando seguridad al sistema entero.

6.5.1.2.5 Módulo Web

Este módulo se encarga de gestionar las solicitudes de los clientes, las que pueden venir de una aplicación web o de escritorio.

6.5.1.2.6 Módulo PAR

Este módulo se encarga de brindar un nivel más de seguridad al sistema completo ya que encierra a cada módulo en un único envoltorio haciéndolos completamente invisibles al resto de aplicaciones.

Cómo se mencionó, el detalle de cada uno de los requerimientos de software para los cinco primeros módulos se encuentra en el documento que se encuentra anexo de forma digital titulado "Especificación de requerimientos de software para el sistema para el control de indicadores en el proceso de evaluación de carreras con fines de acreditación".

6.6 Diseño

6.6.1 Diseño de la interfaz

6.6.1.1 Pantalla principal

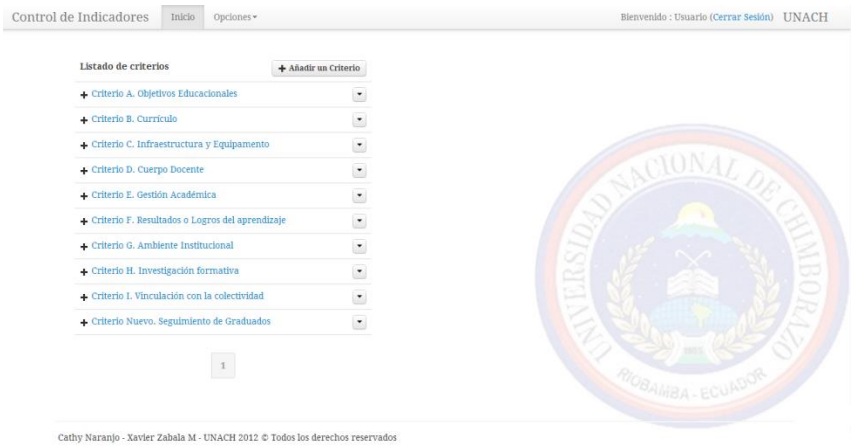

Figura 13. Pantalla principal

Esta pantalla contiene dos elementos principales que serán explicados uno a uno a continuación.

- ✓ Barra de navegación
- ✓ Listado de criterios, sub-criterios, indicadores.

6.6.1.1.1 Barra de navegación

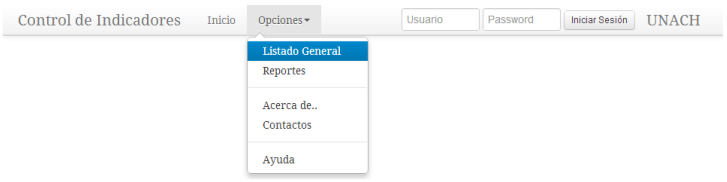

Figura 14. Barra de navegación

La barra de navegación está dividida en cinco partes

6.6.1.1.2 Nombre del sistema y botón de inicio

Figura 15. Nombre del sistema y botón de inicio

Es el nombre de la aplicación y dando un click encima de este nombre o en el botón de inicio se puede acceder a la página principal del sistema.

6.6.1.1.3 Menú de opciones

Figura 16. Menú de opciones

Contiene un menú de acceso a diferentes partes del sistema, como:

- ✓ El listado general de los criterios, listando por cada criterio su correspondiente sub-clasificación tanto de sub-criterios como de indicadores, además por cada indicador se muestra la última calificación realizada; en la sección Listado general se analizará esta opción.
- ✓ La opción Reportes, que permite generar reportes personalizados de un indicador, permitiéndole al usuario imprimir los seguimientos realizados en un rango de fechas.
- ✓ La opción acerca de… muestra la información del sistema, de los desarrolladores, colaboradores, tutores y responsables del sistema.

- ✓ La opción ayuda permite ver una guía en línea de cada una de las partes del sistema.

6.6.1.1.4 Inicio de sesión

Figura 17. Inicio de sesión

Esta parte de la barra de navegación permite al usuario iniciar sesión en el sistema, solo los usuarios registrados podrán modificar la información que el sistema muestra a los usuarios no registrados.

Si un usuario que no está registrado intenta acceder a alguna parte del sistema que necesita de permisos especiales para la navegación, el sistema automáticamente le redirigirá hacia la pantalla de registro en la que tendrá que ingresar sus datos.

Figura 18. Formulario para el inicio de sesión

Una vez que el usuario haya iniciado sesión en la parte superior derecha de la pantalla se mostrará lo siguiente:

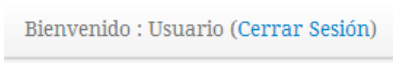

Figura 19. Sesión iniciada, cerrar sesión

6.6.1.1.5 Botón UNACH

Figura 20. Botón UNACH

Este botón permite acceder a la página principal de la Universidad Nacional de Chimborazo.

6.6.1.2 Listado de criterios, sub-criterios e indicadores.

Figura 21. Listado de criterios, sub-criterio, indicadores

Esta pantalla está dividida en:

- ✓ Listado de Criterios
- ✓ Botón añadir un nuevo criterios
- ✓ Botón Expandir
- ✓ Botón opciones
- ✓ Descripción del criterio

6.6.1.2.1 Listado de Criterios

Figura 22. Listado de criterios

Listado de todos los criterios registrados en el sistema.

6.6.1.2.2 Botón añadir un nuevo criterios

Figura 23. Botón añadir criterios

Muestra el formulario para el ingreso de un nuevo indicador al sistema.

6.6.1.2.3 Botón Expandir

Figura 24. Botón mostrar subcriterios e indicadores

El signo "mas" que se encuentra al lado izquierdo del nombre del criterio, permite visualizar el listado de sub-criterios e indicadores del criterio seleccionado, esto se muestren de la siguiente manera:

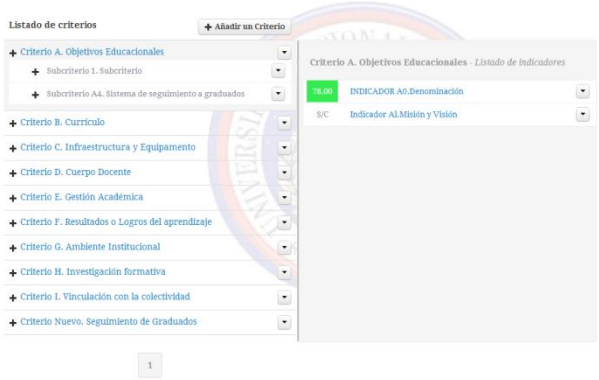

Figura 25. Sub-clasificación de un criterio en sub-criterios e indicadores

Bajo el nombre del criterio se mostrará un listado de los sub-criterios que pertenezcan al criterio seleccionado y en la parte derecha del listado se mostrará el listado de indicadores que pertenezcan al criterio seleccionado.

En la parte izquierda del cada elemento del listado de indicadores representa la última calificación realizada a ese indicador y el color muestra el estado actual del indicador.

Figura 26. Listado de indicadores

6.6.1.2.4 Botón opciones

Figura 27. Opciones para un criterio

Figura 28. Opciones para un sub-criterio

Tanto los criterios como los sub-criterios tendrán el mismo menú de opciones que consta de los siguientes elementos:

- ✓ Eliminar criterio o sub-criterio
- ✓ Modificar criterio o sub-criterio
- ✓ Añadir sub-criterio
- ✓ Añadir indicador o categoría

6.6.1.2.5 Descripción del criterio, sub-criterio e indicador

Al lado derecho en el caso de los criterios y sub-criterios e izquierdo en el caso de los indicadores se mostrará un cuadrado de información que contenga la descripción del indicador. Este cuadro de información aparecerá cuando el cursor del mouse se coloque encima del criterio sub-criterio o indicador y desaparecerá cuando se quite el cursor del mouse.

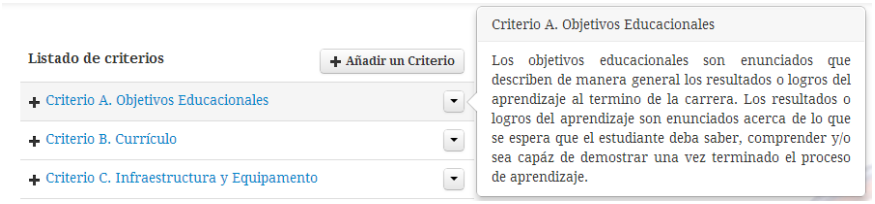

Figura 29. Cuadro de información para la descripción de un criterio

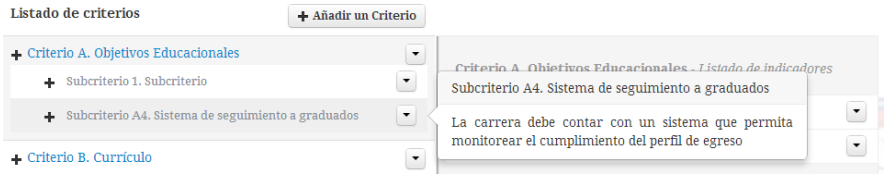

Figura 30 Cuadro de información para la descripción de un sub-criterio

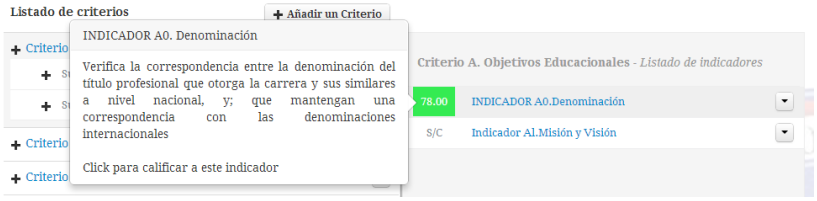

Figura 31. Cuadro de información para la descripción de un indicador

6.6.1.3 Ingreso de un criterio

Figura 32. Formulario de ingreso de un indicador

El botón guardar registrará en el sistema los datos del criterio que se haya ingresado. El botón cancelar retornará a la página principal del sistema y cualquier cambio hecho en los campos del formulario serán eliminados.

Los campos prefijo, nombre y descripción son obligatorios para que el sistema pueda registrar un nuevo criterio así que si alguno de ellos no se ha editado se pintarán sus bordes de rojo indicando que todavía no se ha ingresado ningún dato.

Figura 33. Validación de datos. No se ha ingresado datos

Cuando el usuario haya ingresado algún dato el sistema automáticamente cambiará el color de los bordes del campo de texto a azul.

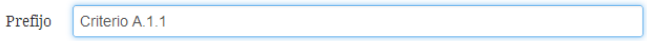

Figura 34. Validación de datos, datos correctos

Si el usuario oprime el botón guardar sin haber ingresado algún dato que es obligatorio el sistema no permitirá el ingreso hasta que se hayan realizado las sugerencias que se mostrarán de la siguiente manera:

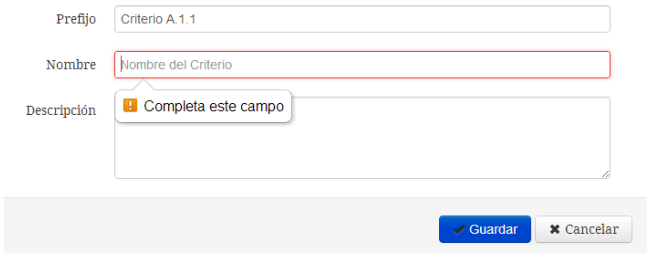

Figura 35. Validación de datos. Error en los datos

Cuando todos los datos estén correctos y se los haya registrado, el sistema automáticamente mostrará la pantalla principal y el registro ingresado se mostrará al final del listado o en la última página.

6.6.1.4 Ingreso de un sub-criterios

Figura 36. Formulario para el ingreso de un nuevo sub-criterio

El proceso de ingreso es el igual que el proceso de ingreso de un criterio.

6.6.1.5 Ingreso de un indicador

6.6.1.5.1 Datos generales

Figura 37. Formulario para el ingreso de un nuevo indicador. Parte 1

Si algún campo obligatorio no se llenara el sistema mostraría el siguiente mensaje:

Figura 38. Validación de datos. Mensajes de error

6.6.1.5.2 Ingreso de puntajes

Figura 39. Formulario para le ingreso de un nuevo indicador. Parte 2

El ingreso de puntajes consta de dos partes:

6.6.1.5.3 Ejemplos de Puntajes

Los ejemplos de puntajes detallan los tipos de calificaciones que se puede realizar a un indicador, estos pueden ser:

6.6.1.5.3.1 Valor Único

Figura 40. Descripción de los tipos de puntajes. Valor único

6.6.1.5.3.2 Peso - Rango

Figura 41. Descripción de los tipos de puntajes. Peso - Rango

6.6.1.5.3.3 Peso - Valor único

Figura 42. Descripción de los tipos de puntajes. Peso - Valor Único

6.6.1.5.3.4 Escala - Peso

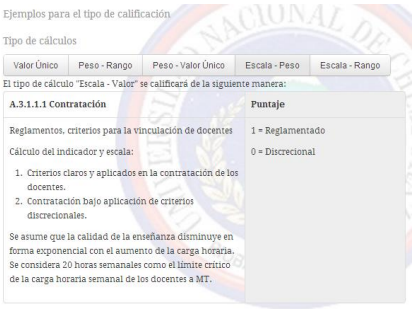

Figura 43. Descripción de los tipos de puntajes. Escala - Peso

6.6.1.5.3.5 Escala -Rango

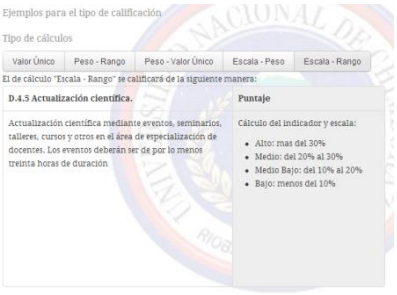

Figura 44. Descripción de los tipos de puntajes. Escala – Rango

6.6.1.5.4 Ingreso de Puntaje

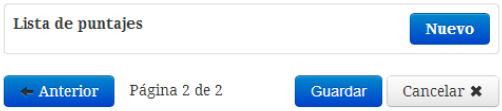

Figura 45. Ingreso de puntajes para el indicador

Cuando damos un click en el botón nuevo se mostrará el siguiente recuadro en el que se muestra las opciones para el nuevo puntaje o calificación:

Figura 46. Formulario de ingreso de puntajes

La opción Tipo de puntaje muestra un listado de los puntajes mencionados anteriormente, como es muestra a continuación:

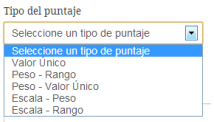

Figura 47. Opciones de puntaje

Y tras la selección de cualquiera de ellos se mostrará una de las siguientes imágenes de acuerdo a la selección.

6.6.1.5.4.1 Valor único

Figura 48. Formulario de ingreso de un nuevo puntaje. Valor Único

6.6.1.5.4.2 Peso - Rango

Figura 49. Formulario de ingreso de un nuevo puntaje. Peso - Rango

6.6.1.5.4.3 Peso - Valor único

Figura 50. Formulario de ingreso de un nuevo puntaje. Peso - Valor Único

6.6.1.5.4.4 Escala -Peso

Figura 51. Formulario de ingreso de un nuevo puntaje. Escala – Peso

6.6.1.5.4.5 Escala - Rango

Figura 52. Formulario de ingreso de un nuevo puntaje. Escala - Rango

Después de llenar cada uno de los campos, de acuerdo al tipo de puntaje que se haya seleccionado y presionar el botón guardar se actualizará la sección ingreso puntajes y se mostrará de la siguiente manera:

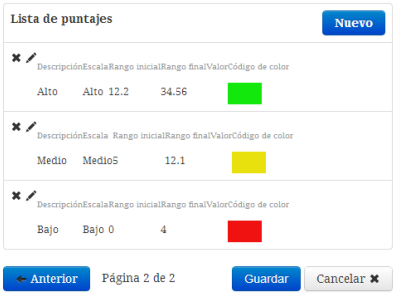

Figura 53. Listado de puntajes ingresados

6.6.1.5.5 Modificación de puntajes

Para modificar un puntaje se debe dar un click en el icono en forma de lápiz y se mostrará el siguiente recuadro en el que modificaremos lo que necesitemos cambiar

Figura 54. Formulario para la modificación de un puntaje

Una vez hechos los cambios y después de presionar el botón guardar, los datos automáticamente se actualizarán en el listado de puntajes

6.6.1.5.6 Eliminación de puntajes

Para eliminar un puntaje se debe dar un click en el icono en forma de cruz que se encuentra ubicado en el lado izquierdo del puntaje y confirmar la eliminación del puntaje. El puntaje seleccionado para eliminar se pintara de rojo hasta que se confirme o descarte la eliminación.

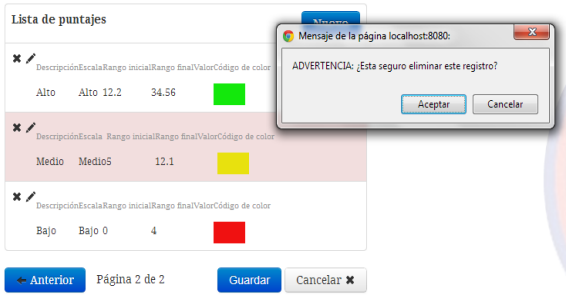

Figura 55. Eliminación de un puntaje

Una vez eliminado automáticamente se actualizará el listado de puntajes.

6.6.1.6 Modificación de un criterio y de un sub-criterio

Figura 56. Formulario para la modificación de un criterio y un subcriterio.

6.6.1.7 Modificación de un indicador

6.6.1.7.1 Datos generales

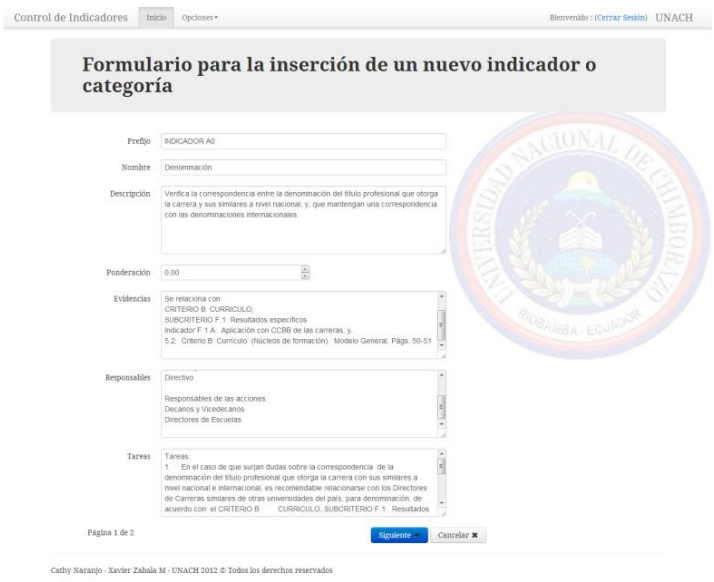

Figura 57. Formulario para la modificación de un indicador

6.6.1.7.2 Modificación de Puntajes

Para modificar un puntaje se debe dar un click en el icono en forma de lápiz y se mostrará el siguiente recuadro en el que modificaremos lo que necesitemos cambiar

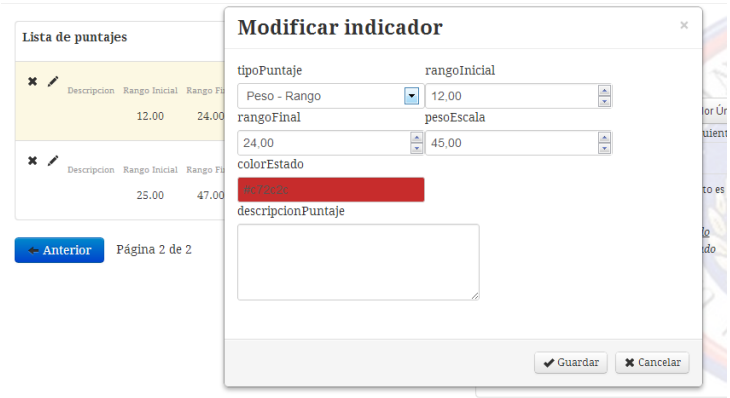

Figura 58. Formulario para la modificación de un puntaje

Una vez hechos los cambios y después de presionar el botón guardar, los datos automáticamente se actualizarán en el listado de puntajes

6.6.1.8 Eliminación de un criterio, sub-criterio e indicador.

En cualquiera de los tres casos después de elegir la opción eliminar al pulsarla con el puntero del mouse, se debe confirmar o descartar la eliminación del registro, en caso de confirmar la eliminación el sistema automáticamente eliminará el registro y actualizará el listado.

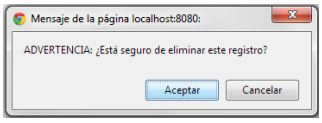

Figura 59. Cuadro de confirmación para la eliminación de un criterio o sub-criterio

6.6.1.9 Calificación de un indicador

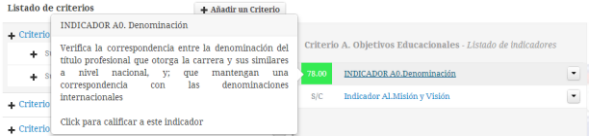

Figura 60. Ingreso al formulario de las calificaciones o seguimientos de un indicador

En el listado de los indicadores, desde el nombre de cada registro se puede acceder a la calificación del indicador, seleccionando esta opción se mostrará el siguiente formulario:

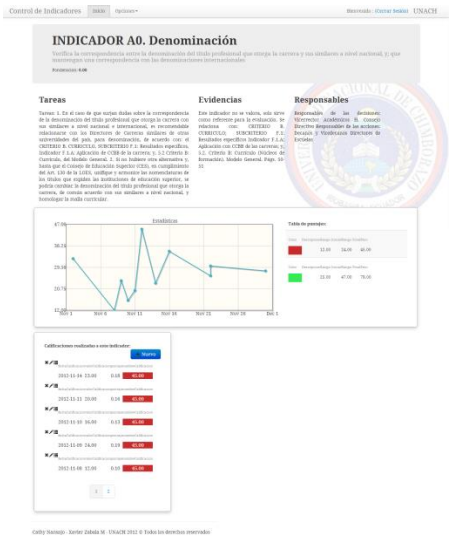

Figura 61. Formulario de seguimientos o calificaciones de un indicador

En este formulario se puede diferenciar tres partes:

- ✓ Información del Indicador.
- ✓ Estadísticas de las calificaciones realizadas al indicador

- ✓ Ingreso, modificación y eliminación de las calificaciones realizadas a un indicador.

6.6.1.9.1 Información del indicador

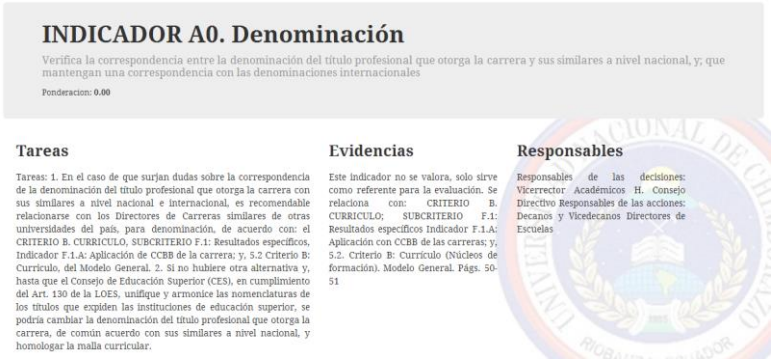

Figura 62. Información del indicador

En esta parte se muestra la información general de cada indicador, como son el prefijo y título del indicador "Indicador A0. Denominación". Bajo el título está la descripción de este indicador y la ponderación.

Y en la parte final se encuentra dividida en tres partes, las tareas o lo que se debe hacer para calificar este indicador, las evidencias o respaldos que demuestren que se han realizado las tareas y los responsables de la calificación de este indicador.

6.6.1.9.2 Estadísticas de las calificaciones realizadas al indicador.

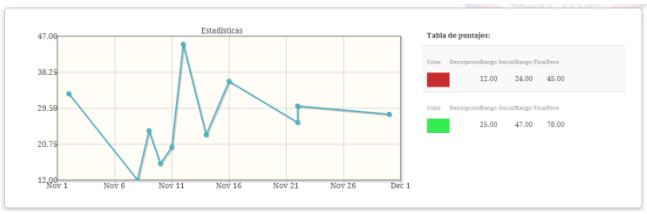

Figura 63. Estadísticas de las calificaciones

Las estadísticas están divididas en dos partes:

1. Gráfico de estadísticas.- muestra el progreso de las calificaciones a través del tiempo.
2. Tabla de Puntajes.- es un tabla informativa que detalla el cómo se debe leer el gráfico de las estadísticas.

6.6.1.9.3 Ingreso, modificación y eliminación de las calificaciones realizadas a un indicador.

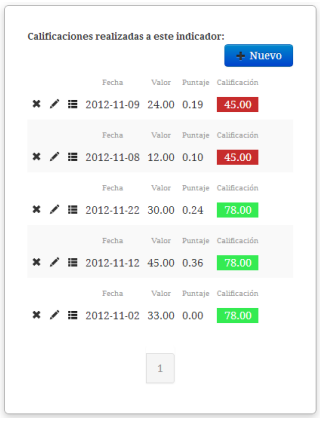

Figura 64. Ingreso de calificaciones

Esta es una de las partes más importantes del sistema, ya que desde aquí se lleva el seguimiento de los indicadores, en el gráfico anterior podemos ver un listado de los seguimientos o calificaciones realizadas a un indicador. Desde aquí se puede acceder al ingreso eliminación y modificación de un nuevo seguimiento, además del ingreso de las evidencias por cada seguimiento.

6.6.1.9.4 Ingreso de un nuevo seguimiento

Al dar un click en el botón nuevo 

, se mostrará el siguiente formulario, desde el cuál podremos dar una calificación en la fecha actual al indicador:

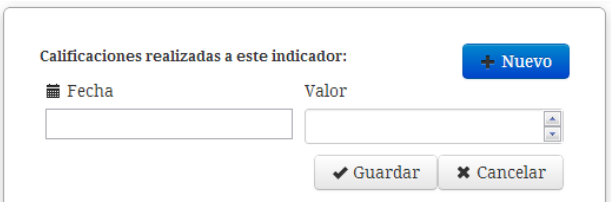

Figura 65. Ingreso de un nuevo seguimiento

Al dar click en el campo de texto bajo el título Fecha, se mostrará el siguiente calendario desde el que podemos seleccionar la fecha en la que deseamos calificar a este indicador, cabe señalar que como predeterminado está seleccionada la fecha actual.

Figura 66. Selección de la fecha para el nuevo seguimiento

Para calificar un indicador se debe ingresar la fecha y el valor de la calificación, si el valor ingresado no está dentro de un rango específico se mostrará el siguiente mensaje:

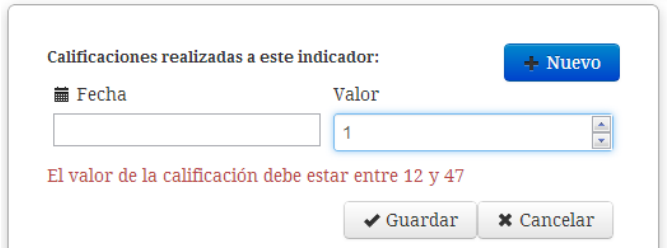

Figura 67. Mensajes de error para el valor de la calificación

Este rango es el que se le dio al indicador en el momento de ingresar los puntajes, y está detallado al lado derecho de la gráfica de estadísticas

Figura 68. Tabla de detalle del tipo de puntajes para el indicador

No en todos los indicadores el rango va a ser el mismo rango, esto va a depender del tipo de puntajes que se le de al indicador.

6.6.1.9.5 Modificación de una calificación

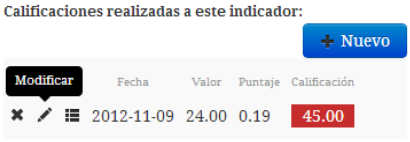

Figura 69. Modificación de un seguimiento

Al lado izquierdo de cada seguimiento se encuentra un icono en forma de lápiz cuya función es la de mostrar un formulario que permita la modificación de esa calificación en particular. Al dar click en esa opción se mostrará en el mismo lugar del seguimiento el siguiente formulario, en este ejemplo está con los bordes de color rojo:

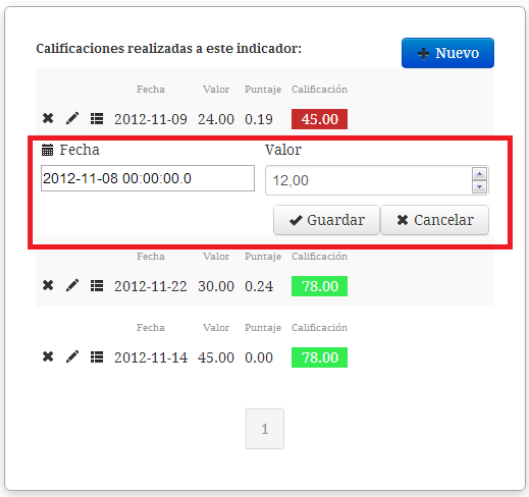

Figura 70. Formulario de modificación de un seguimiento

Desde acá podemos cambiar la fecha y el valor de la calificación para ese seguimiento en particular. Después de pulsar el botón guardar los cambios automáticamente se guardarán en el sistema y se actualizará tanto el gráfico de las estadísticas cómo el listado de los seguimientos.

6.6.1.9.6 Eliminación de un seguimiento

Figura 71. Eliminación de un indicador

Al lado izquierdo del seguimiento se encuentra la opción para eliminar el seguimiento, después de dar click en esta opción se mostrará el siguiente cuadro de confirmación:

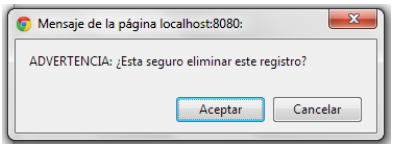

Figura 72. Cuadro de confirmación para le eliminación de un seguimiento

Al aceptar el registro seleccionado para la eliminación se habrá borrado del sistema y no se podrá deshacer los cambios. Después de la eliminación se mostrará el siguiente mensaje en caso de haberse eliminado:

Figura 73. Mensaje de confirmación de la eliminación de un seguimiento

6.6.1.9.7 Ingreso de evidencias por cada seguimiento

Figura 74. Evidencias de un seguimiento

Al lado izquierdo del seguimiento se encuentra la opción Ver evidencias, al dar click en esta opción se mostrará en la parte inferior de la calificación el siguiente formulario:

Figura 75. Formulario añadir evidencias para un seguimiento

Desde aquí se podrá añadir respaldos de las evidencias en formato digital. Después de dar un click en el botón evidencias se mostrará el siguiente formulario:

Figura 76. Formulario para el ingreso de un a nueva evidencia

Desde aquí podemos seleccionar un archivo y subirlo al sistema, al dar click en el botón seleccionar archivo se mostrará el siguiente cuadro de dialogo que permitirá seleccionar un tipo de archivo soportado por el sistema.

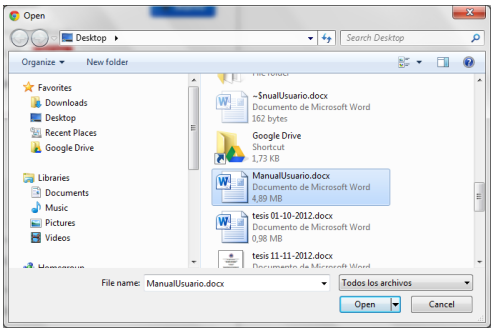

Figura 77. Cuadro de dialogo para la selección de una nueva evidencia

Los tipos de archivos soportados por el sistema son:

- ✓ Archivos de Excel (.xlsx, xls)
- ✓ Archivos de Word (.docx, .doc)
- ✓ Archivos Pdf
- ✓ Imágenes (.jpg, .png, .jpeg)

Si el archivo seleccionado no está en la lista anterior el sistema mostrará el siguiente mensaje de error:

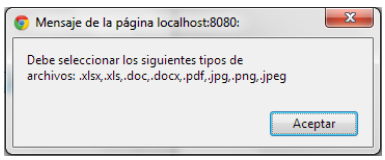

Figura 78. Tipos de archivos soportados por el sistema

En caso de haber seleccionado el archivo correcto el sistema preguntará si se desea subir otro archivo en caso afirmativo se deberá seguir los pasos anteriormente explicados sino se presionará el botón cancelar y se mostrará un listado de los archivos que se hayan subido al sistema.

Estas evidencias no se podrán ni eliminar ni modificar, si se desea modificar alguna se deberá subir un nuevo documento.

VII. GLOSARIO

- ✓ **ANSI**: (American National Standards Institute) Instituto Nacional Estadounidense de Estándares es una organización sin ánimo de lucro que supervisa el desarrollo de estándares para productos, servicios, procesos y sistemas en los Estados Unidos.
- ✓ **API**: (Application Programming Interface) Interfaz de programación de aplicaciones.
- ✓ **CEAACES**: Consejo de Evaluación, Acreditación y Aseguramiento de la Calidad de la Educación Superior.
- ✓ **CLDC**: (Connected Limited Device Configuration) Define el conjunto de la base de las interfaces de programación de aplicaciones y una máquina virtual para dispositivos con recursos limitados como teléfonos móviles, busca personas y principales asistentes digitales personales.
- ✓ **DB**: Base de Datos.
- ✓ **FICHEROS HBM**: Ficheros de mapeo que contienen metadatos que definen los mapeos objeto/relacional para las clases Java en aplicaciones Hibernate.
- ✓ **FRAMEWORK**: (estructura) Conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular, que sirve como referencia para enfrentar y resolver nuevos problemas de índole similar.
- ✓ **HIBERNATE**: es una herramienta para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.
- ✓ **HQL HIBERNATE QUERY LANGUAGE:** lenguaje similar a SQL con la diferencia que es completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación.
- ✓ **HTTP**: (Hypertext Transfer Protocol o HTTP) Protocolo de transferencia de hipertexto es el protocolo usado en cada transacción de la World Wide Web.
- ✓ **HTTPS**: (Hyper Text Transfer Protocol Secure) Protocolo seguro de transferencia de hipertexto, es un protocolo de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos de Hiper Texto, es decir, es la versión segura de HTTP. Describe cualquier interfaz web simple que utiliza XML y HTTP, sin las abstracciones adicionales de los protocolos basados

en patrones de intercambio de mensajes como el protocolo de servicios web SOAP.

- ✓ **IEEE**: (Institute of Electrical and Electronics Engineers) Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas.
- ✓ **J2EE**: (Java 2 Enterprise Edition) Edición empresarial del paquete Java creada y distribuida por Sun Microsystems. Comprende un conjunto de especificaciones y funcionalidades orientadas al desarrollo de aplicaciones empresariales.
- ✓ **J2SE**: (Java Platform, Standard Edition o Java SE) Colección de APIs del lenguaje de programación Java útiles para muchos programas de la Plataforma Java.
- ✓ **JAXP**: (Java Api for XML Processing) API de Java que sirve para la manipulación y el tratamiento de archivos XML.
- ✓ **JDBC**: JDBC es un API incluido dentro del lenguaje Java para el acceso a bases de datos. Consiste en un conjunto de clases e interfaces escritos en Java que ofrecen un completo API para la programación de bases de datos.
- ✓ **JSON**: (JavaScript Object Notation) es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.
- ✓ **JVM**.- (Java Virtual Machine) Es un máquina virtual ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el byteco de Java), el cual es generado por el compilador del lenguaje Java.
- ✓ **MIDP**: (Mobile Information Device Profile) Permite escribir aplicaciones descargables y servicios de red pueden conectar los dispositivos móviles.
- ✓ **ORM**: Mapeo Objeto-Relacional (Object-Relational Mapping). Los ORM son herramientas de software que permiten trabajar con los datos persistidos en una base de datos relacional como si fuera parte de una base de datos orientada a objetos. La función del ORM es transformar un registro en objeto y viceversa.
- ✓ **OSGi**: (Open Services Gateway Initiative) Iniciativa para el acceso a servicios abiertos.
- ✓ **POJO:** (Plain Old Java Objects). Archivos planos del java antiguo.
- ✓ **REST**: (Representational State Transfer) Transferencia de Estado Representacional es una técnica de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web.

- ✓ **RPC**: (Remote Procedure Call) Es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos.
- ✓ **Servlets**: Aplicaciones Java que corren en un entorno de servidor web. Esto es análogo a una aplicación Java que corre en un navegador. Los Java servlets se han vuelto muy populares como alternativas a los programas CGI. La diferencia entre ambos es que los applet de Java son persistentes. Esto significa que una vez que han sido iniciados, se mantienen en memoria y pueden satisfacer múltiples solicitudes. En contraste, los programas CGI desaparecen una vez que han satisfecho una solicitud.
- ✓ **SGML**: (Standard Generalized Markup Language - Lenguaje de Marcado de Anotaciones Generales). Es un metalenguaje de donde deriva el HTML y el XML.
- ✓ **SOAP**: (Simple Object Access Protocol) Protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.
- ✓ **SQL**: Lenguaje de definición de datos (LDD), un lenguaje de definiciones de vistas (LDV) y un lenguaje de manipulación de datos (LMD), que posee también capacidad para especificar restricciones y evolución de esquemas.
- ✓ **SRS** o **ERS**: (Software Requirement Specifications) Especificación de Requisitos Software, es una descripción completa del comportamiento del sistema que se va a desarrollar. Incluye un conjunto de casos de uso que describe todas las interacciones que tendrán los usuarios con el software. Los casos de uso también son conocidos como requisitos funcionales. Además de los casos de uso, la ERS también contiene requisitos no funcionales (o complementarios). Los requisitos no funcionales son requisitos que imponen restricciones en el diseño o la implementación (Como por ejemplo restricciones en el diseño o estándares de calidad).
- ✓ **Tomcat**: Contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de Java Server Pages (JSP) de Sun Microsystems.
- ✓ **UML**: (Unified Modeling Languaje) Lenguaje gráfico para construir, documentar, visualizar y especificar un sistema de software.
- ✓ **UNACH**: Universidad Nacional de Chimborazo.

- ✓ **UPnP**: (Universal Plug and Play) Es una arquitectura software abierta y distribuida que de forma independiente al fabricante, sistema operativo, lenguaje de programación, etc. permite el intercambio de información y datos a los dispositivos conectados a una red.
- ✓ **VO**: (valueObject). Objeto que contiene información de negocio estructurada en grupos de ítems de datos, también recibe el nombre de transfer object.
- ✓ **W3C**: (World Wide Web Consortium). Consorcio internacional que produce recomendaciones para la World Wide Web.
- ✓ **XML**: (Extensible Markup Language) Lenguaje de marcas extensible, es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos.

VIII. REFERENCIAS BIBLIOGRÁFICAS

- ✓ Alliance, O. (3 de Marzo de 2012). *OSGi Alliance*. Recuperado el 3 de Marzo de 2012, de OSGi Alliance: http://goo.gl/pCeid
- ✓ Alliance, T. O. (2011). *OSGi Service Platform Core Specification* . The OSGi Alliance.
- ✓ Bartlett, N. (2010). *OSGI in Practice.* United States of America: Christopher Brind.
- ✓ Bernard, E. (6 de Abril de 2011). *Hibernate Annotations*. Recuperado el 30 de Enero de 2012, de http://goo.gl/ZjEkx
- ✓ CloudSigma. (2011). *Computación En Nube*. Recuperado el 5 de Marzo de 2012, de http://goo.gl/gE4Ck
- ✓ Cogoluègnes, A., Templier, T., & Piper, A. (2011). *Spring Dynamics Modules in Action.* United States of America: Manning Publications Co.
- ✓ Corporation, O. (3 de Marzo de 2012). *Project Jigsaw*. Recuperado el 3 de Marzo de 2012, de Open JDK: http://goo.gl/5UJxJ
- ✓ Daniel Haischt, P. H. (2010). *Getting Started with the Feature Pack for OSGi Applications and JPA 2.0.* United States of America: IBM.
- ✓ Eclipse. (2010). *Guía de programación de Virgo*. Recuperado el 11 de 5 de 2012, de http://goo.gl/GgEyh
- ✓ Gracia, L. M. (7 de Junio de 2010). *OSGi: Primeros pasos y el inevitable Hello World*. Recuperado el 24 de Febrero de 2012, de OSGi: Primeros pasos y el inevitable Hello World: http://goo.gl/gCIoa
- ✓ Hall, R. S., Pauls, K., McCulloch, S., & Savage, D. (2011). *OSGi in Action. Creating modular applications in Java.* United States of America: Manning Publications Co.
- ✓ IBM. (8 de Septiembre de 2011). *La plataforma de servicios OSGi*. Recuperado el 2012 de Marzo de 25, de http://goo.gl/ST1ft

- ✓ Knoernschild, K. (23 de Febrero de 2010). *Osgi en la Empresa*. Recuperado el 1 de Marzo de 2012, de Agilidad, Modularidad y la paradoja de la Arquitectura.: http://goo.gl/MaLgd
- ✓ Knoernschild, K. (2012). *Java Application Architecture: Modularity Patterns with Examples Using OSGi.* United States of America: Prentice Hall.
- ✓ Martin, R. C. (Diciembre de 1996). *Granularidad*. Recuperado el 19 de Marzo de 2012, de http://goo.gl/dZZEh
- ✓ Martin, R. C. (Mayo de 1996). *Principio de Inversión de Dependencias*. Recuperado el 19 de Marzo de 2012, de http://goo.gl/SA8r9
- ✓ Martin, R. C. (Marzo de 1996). *Principios de la Orientación a Objetos*. Recuperado el 19 de Marzo de 2012, de http://goo.gl/oqExW
- ✓ Martines, D. (21 de Julio de 2010). *Daniel Martínez Blog*. Recuperado el 30 de Enero de 2012, de http://goo.gl/bMvGB
- ✓ Moreno, F. G. (01 de Enero de 2011). *Desarrollando una aplicación Spring Framework MVC v3 + JPA paso a paso*. Recuperado el 30 de Enero de 2012, de http://goo.gl/cMS5V
- ✓ Oracle. (1999). *JAR File Specification*. (Oracle) Recuperado el 26 de Marzo de 2012, de JAR File Specification: http://goo.gl/QYbm0
- ✓ Paez, N. (2010). *Utilización de programación orientada a aspectos en aplicaciones empresariales.* Buenos Aires: Anónimo.
- ✓ Palao, D. M. (1 de Febrero de 2010). *Desarrollando una aplicacion Spring Framework MVC paso a paso*. Recuperado el 30 de Enero de 2012, de http://goo.gl/XRFk4
- ✓ Parnas, D. (1972). *On the Criteria To Be Used in Decomposing Systems into Modules.* Communications of the ACM, vol.15.
- ✓ Ramos, A. (27 de Abril de 2010). *Modularización efectiva*. Recuperado el 1 de Marzo de 2012, de http://goo.gl/rFvBi

- ✓ Ramos, A. (23 de Febrero de 2010). *Modularización Efectiva en Java*. Recuperado el 19 de Febrero de 2012, de Modularización Efectiva en Java: http://goo.gl/yQIKw

- ✓ Risberg, T., Evans, R., & Tung, P. (1 de Enero de 2010). *Developing a Spring Framework MVC application step-by-step*. Recuperado el 30 de Enero de 2012, de http://goo.gl/kMWUs

- ✓ Rodríguez, J. H. (2010). *Simulación gráfica de servicios OSGi en entornos domóticos.* España.

- ✓ Völter, M. (10 de Marzo de 2005). *Software Architecture Patterns*. Recuperado el 3 de Marzo de 2012, de A pattern language for building sustainable software architectures: http://goo.gl/jYVHv

- ✓ Walls, C. (2010). *Modular Java, Creating flexible applications with OSGi and Spring.* Dallas, Texas: The Pragmatic Bookshelf.

- ✓ Wikilearning. (12 de Enero de 2010). *DEFINICIÓN Y TERMINOLOGÍA DE UN RDBMS*. Recuperado el 6 de Febrero de 2012, de http://goo.gl/CtKl6

IX. ANEXOS

9.1 CD de instalación

9.2 Manual de Usuario (Digital)

9.3 Manual Técnico (Digital)