Universidad de las Ciencias Informáticas
Facultad 3

Título: Arquitectura J2EE aplicada al módulo de Servicio Autónomo

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor(es): Adalberto Somarriba Montero

Yusbel Echemendia González

Tutor(es): Ing. Marbys Marante Valdivia

Ing. Luis Enrique Carrazana Carbonell

Ciudad de la Habana

Mayo de 2007

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2007.

Autores	Tutores
Adalberto Somarriba Montero	Ing. Marbys Marante Valdivia
_____	_____
Yusbel Echemendia González	Ing. Luis Enrique Carrazana Carbonell
_____	_____

AGRADECIMIENTOS

A la UCI,
por permitirnos alcanzar este anhelo.

A nuestros profesores,
indudables creadores de lo que hemos podido llegar a ser.

A nuestros tutores Marante y Carrazana,
por todo el apoyo y la confianza que nos brindaron.

A los compañeros del aula,
quienes han estado siempre en las buenas y en las malas
y de los que más hemos aprendido.

A Pedro,
por haber servido de ejemplo y enseñarnos a pensar.

A nuestros padres,
nadie más que ellos hacen realidad el hecho de
vernos hechos ingenieros.

A todos aquellos que no hemos mencionado,
pero que de sobra se deben dar por aludidos por haber
contribuido de un modo u otro en nuestra formación
y a la consumación de este sueño.

DEDICATORIA

A mi familia, especialmente a mima, a papi y a mi hermanita Yarenis, sin su amor y apoyo constante ni siquiera hubiese podido haber emprendido esta travesía.

A nuestra gloriosa Revolución, autora y consumadora de este sueño.

A nuestro querido e invencible Comandante en Jefe, guía y luz que me impulsa a continuar superándome y a ser mejor cada día.

Yusbel

A mi madre, jamás habría llegado hasta donde estoy si no hubiera sido por ti, aunque esto sobra decirlo. Te quiero mucho.

A mi padre, a pesar de que ya no estás, no sabes cuánto habría querido que me vieras graduado.

A mi hermana, por siempre estar lista para socorrerme cuando te necesité.

A mi sobrina, crece pronto, que ahora es tu turno.

A mis familiares, ustedes son todos muy especiales.

A Aileen, por esperarme y estar ahí en todo momento.

A todos los que siempre confiaron en mí y me dieron su apoyo incondicional, gracias.

Adalberto

RESUMEN

Es creciente el número de solicitudes de aplicaciones empresariales para la web basadas en Java, específicamente sobre Java 2 Entrerprise Edition (en lo adelante J2EE). Por otro lado, es muy grande el número de soluciones libres reutilizables orientadas a la web existente. Sin embargo es bien pobre el conocimiento por parte de los programadores de menos experiencia acerca de las posibles tecnologías y combinaciones de frameworks[1] capaces de brindar una arquitectura sólida sobre la cual desarrollar. Debido a esto y a que generalmente se cuenta con poco tiempo para el desarrollo de una aplicación es que se hace necesario contar con arquitecturas basadas en la plataforma J2EE viables, capaces de proveer la estructura base para crear aplicaciones web sólidas, confiables, robustas, escalables y portables, tanto en el marco de la Universidad de las Ciencias Informáticas[2] (UCI) como a nivel nacional.

Es por ello que, partiendo de un estudio de las tendencias actuales y tomando la experiencia de la implementación del módulo de Servicio Autónomo del Sistema Servicio Autónomo de Registros y Notarías (en lo adelante SAREN), desarrollado sobre J2EE utilizando soluciones completamente libres y de código abierto, que el presente Trabajo de Diploma pretende, mediante ejemplos prácticos y hechos, ofrecer una alternativa de arquitectura J2EE que brinde vías para reducir los riesgos y tiempo de desarrollo, permitiendo al mismo tiempo crear aplicaciones cada vez más complejas y seguras ensambladas íntegramente sobre soluciones de software libre.

PALABRAS CLAVES

Java, J2EE, arquitectura, framework, JSF, Struts, Spring, Hibernate, MVC, DAO, Acegi, SAREN, máquina virtual, software libre, XML.

[1] **Ver glosario**.
[2] **Universidad de las Ciencias Informáticas**. Es un centro universitario de la educación superior cubana, docente-productor para la industria del software en Cuba.

TABLA DE CONTENIDOS

AGRADECIMIENTOS I

DEDICATORIA II

RESUMEN III

INTRODUCCIÓN 1

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 6

1.1 Introducción 6
1.2 Gobierno electrónico 6
1.2.1 Gobierno electrónico en Venezuela 7
1.3 SAREN 7
1.3.1 Objetivos del SAREN 8
1.4 Aquitectura 9
1.4.1 Arquitectura de software 10
1.4.2 Patrones de arquitectura 13
1.4.3 Arquitectura de software empresarial 16
1.5 Plataformas para el desarrollo de software empresarial 17
1.5.1 La plataforma J2EE 17
1.5.2 La plataforma .NET 25
1.5.3 Comparación entre J2EE y .NET 26
1.6 Metodología de desarrollo 30
1.6.1 RUP y UML 30

CAPÍTULO 2: ELEMENTOS A CONSIDERAR PARA DESARROLLAR CON J2EE 32

2.1 Introducción 32
2.2 Aplicaciones en capas 32
2.3 Patrones de diseño para aplicaciones basadas en J2EE 34
2.4 Frameworks para J2EE 38
2.4.1 Panorama actual de los frameworks J2EE 40
2.4.2 Tecnología JSP-Servlet 40
2.4.3 Struts 41
2.4.4 JavaServer Faces 43
2.4.5 Spring 48
2.4.6 Acegi 51
2.4.7 Hibernate 52
2.5 Herramientas para el desarrollo de aplicaciones J2EE 54
2.5.1 Servidor de aplicaciones: Tomcat 54
2.5.2 IDE: Eclipse 55
2.5.3 Generador de reportes: iReport 55

2.6 Sistemas de ejemplo desarrollados sobre arquitecturas J2EE 56
2.6.1 Compiere 56
2.6.2 Open for Business OFBiz 56
2.6.3 JCatalog 57
2.6.4 Oness 58
2.7 Conclusiones 59

CAPÍTULO 3: ARQUITECTURA J2EE APLICADA EN EL MÓDULO SERVICIO AUTÓNOMO 60

3.1 Primera aproximación de la arquitectura de Servicio Autónomo 61
3.2 Arquitectura aprobada para Servicio Autónomo 65
3.3 Configuraciones necesarias 67
3.3.1 Configuración del Tomcat 67
3.3.2 Configuración de Struts 70
3.3.3 Configuración de Spring 75
3.3.5 Configuración de Hibernate 79
3.4 Integración entre los frameworks 83
3.4.1 Integración de Struts con Spring 83
3.4.2 Integración de Hibernate con Spring 86
3.4.3 Integración de Acegi con Spring 91
3.5 Ventajas y desventajas de la arquitectura propuesta 99
3.5.1 Ventajas 100
3.5.2 Desventajas 101
3.6 Análisis de los resultados 101

CONCLUSIONES 105

RECOMENDACIONES 107

BIBLIOGRAFÍA 108

GLOSARIO 110

INTRODUCCIÓN

Como parte de la Batalla de Ideas que libra el pueblo cubano, Cuba se encuentra en medio de un constante desarrollo de su industria del software, en el que la UCI representa uno de los principales exponentes. El enfoque del desarrollo de dicha industria en Cuba es hacia el software libre[3]. El número de soluciones empresariales basadas en software libre es inmensamente grande y continúa creciendo. Se hace imposible e inadmisible pasar por alto la presencia de dichas soluciones. En consecuencia, los responsables de la toma de decisiones están dando mayor crédito y considerando cada vez más al software libre como un recurso válido. Las soluciones de software libre pueden, y de hecho lo están haciendo, abrirse paso dentro del mundo de las aplicaciones empresariales y brindan al mismo tiempo todas las características que estos ambientes requieren. En efecto, como se verá en el desarrollo de este trabajo, se pueden tener soluciones y arquitecturas de software completamente empresariales utilizando única y exclusivamente software libre. La mayor parte de las posibles soluciones existentes para J2EE son libres y de código abierto. Sin embargo existe una inclinación por parte de los desarrolladores de sistemas a no utilizarlos de manera correcta o a integrarlos inapropiadamente, debido a que existe una amplia variedad de los mismos y no se cuenta con una arquitectura estandarizada.

Con la Alternativa Bolivariana para las Américas (ALBA) se abren y consolidan nuevos caminos hacia la unidad latinoamericana; la integración se evidencia en los innumerables convenios de colaboración y trabajo entre los pueblos latinoamericanos. Es sabido que en la República Bolivariana de Venezuela se lleva a cabo un proceso revolucionario hacia el socialismo del siglo XXI, entre sus metas se encuentra la modernización de las instituciones del sector público. Entre los ámbitos institucionales a modernizar está el sistema registral y notarial venezolano. El trabajo de modernización y automatización de los Registros y Notarías viene adelantándose en el país desde 1993, fecha en la cual se promulgó la Ley de Registro Público. El organismo encargado de dirigir y controlar los Registros y Notarías venezolanos es el Ministerio del Interior y Justicia (MIJ), el cual no cuenta con herramientas tecnológicas ni procedimentales que garanticen de forma efectiva la ejecución de funciones en los Registros y Notarías.

Con en el Decreto Nº 1.554 con fuerza de Ley de Registro Público y del Notariado publicado en Gaceta Oficial Nº 5.556 de fecha 13 de Noviembre de 2001(Venezuela 2001), se persigue la adaptación del

[3] **Ver glosario**.

ordenamiento jurídico a los cambios actuales, entre los que se encuentran la nuevas tecnologías informáticas para llegar a una automatización del sistema registral y notarial, así como unificar en un mismo texto normativo las disposiciones que regulen la actuación de los Registros Civiles y Públicos, de los Registros Mercantiles y de las Notarías Públicas.

También con la nueva ley de Registro Público y de Notarías publicada en Gaceta Oficial Nº 5.833 de fecha 22 de Diciembre de 2006, uno de los aspectos que se resalta es la creación del Servicio Autónomo de Registros y Notarías. Esto ya está contemplado en la actual ley pero no ha sido desarrollado en su totalidad. El Servicio Autónomo de Registros y Notarías depende jerárquicamente del MIJ, y es el órgano encargado de forma autónoma de la planificación, organización, coordinación, inspección, vigilancia, procedimiento y control sobre todas las oficinas de registros y notarías del país.

Para dar cumplimiento a la automatización del sistema registral y notarial se desarrolla la solución informática Servicio Autónomo de Registros y Notarías (SAREN), que es un sistema empresarial dividido en varios módulos o subsistemas cuyo objetivo rector es justamente contribuir a lo anteriormente planteado. Esto surge como parte de los convenios de colaboración e integración entre los pueblos de Latinoamérica. El módulo de Servicio Autónomo en específico, fue diseñado y desarrollado para regir a los restantes a todo lo largo y ancho del país y ha sido implementado sobre la arquitectura basada en software libre para J2EE que pretende proponer el presente trabajo de diploma.

Siguiendo la legislación venezolana, en el Decreto Nº 3.390 publicado en Gaceta Oficial Nº 38.095 de fecha 28 de Diciembre de 2004 (Venezuela 2004), mediante la cual se dispone que la Administración Pública Nacional empleará prioritariamente software libre desarrollado con estándares abiertos, en sus Sistemas, Proyectos y Servicios Informáticos. Al desarrollarse sobre la plataforma J2EE el módulo Servicio Autónomo se da cumplimiento a lo estipulado anteriormente. Cabe destacar que con la implantación de este sistema se hace un aporte a la fomentación y desarrollo del Gobierno Electrónico[4] en la República Bolivariana de Venezuela.

La plataforma J2EE es una de las alternativas más atrayentes para el desarrollo de aplicaciones empresariales debido a que ya es un producto maduro y con una amplia variedad de especificaciones,

[4] **Ver epígrafe 1.2**.

herramientas y técnicas. Sin embargo, el desarrollo de aplicaciones empresariales[5] con J2EE nunca ha sido fácil o rápido. Hacia ese preciso aspecto es que se han enfocado los miembros de la comunidad internacional de Java[6]: proveer un conjunto de poderosas tecnologías y herramientas con el principal objetivo de reducir el tiempo y la complejidad del proceso de desarrollo y mejorar el comportamiento de las aplicaciones. Para lograr esto, cuando se va a desarrollar una aplicación web sobre J2EE se puede escoger una arquitectura escalable conformada por un extenso número de soluciones libres y de código abierto.

El hecho de que una plataforma disponga de una arquitectura basada en soluciones libres aporta beneficios extras muy importantes a los arquitectos de software, ya que obtienen un amplio espectro de soluciones de bajo costo. Pero precisamente esto, el amplio margen de elección de soluciones, puede conllevar, como se ha expresado anteriormente, a una mala elección en dependencia de las necesidades reales. Es muy grande el número de soluciones libres reutilizables orientadas a la web existentes para crear arquitecturas empresariales dentro de la plataforma J2EE, sin embargo el conocimiento y la experiencia acerca de las posibles tecnologías y combinaciones de frameworks capaces de brindar una arquitectura sólida sobre la cual desarrollar es muy pobre y no existe un estudio detallado sobre el tema que muestre un ejemplo palpable y probado. Por todo lo planteado hasta el momento se identifica la existencia del siguiente **problema científico**: Se tiene como problema la carencia de una guía detallada sobre los modelos tangibles y eficientes de arquitecturas basados en J2EE para utilizar en la producción del módulo de Servicio Autónomo. Es por ello que el presente trabajo tiene como **objeto de estudio** los modelos de arquitectura de sistemas con soluciones libres para desarrollar software de gestión empresarial. El **campo de acción** del presente trabajo se centra en el modelo de arquitectura Struts+Spring+Hibernate. Se tiene como **objetivo general** brindar una guía técnica detallada sobre el modelo de arquitectura J2EE aplicado en el desarrollo del módulo Servicio Autónomo de la solución informática SAREN. Para ello se han trazado los siguientes **objetivos específicos**:

- Identificar los elementos esenciales que definen una arquitectura de software.
- Identificar las tendencias y tecnologías actuales del campo de la informática acerca del tema de las arquitecturas J2EE.

[5] **Ver en glosario**.
[6] **Ver glosario**.

- Identificar las soluciones empresariales más comunes y las más utilizadas por la comunidad de desarrolladores en J2EE.
- Identificar y seleccionar posibles mixturas de soluciones libres empresariales a través de los cuales se pueda definir una arquitectura J2EE robusta, escalable y de desarrollo rápido para un software empresarial; así como las herramientas libres más eficientes y comunes que contribuyen a la implementación de aplicaciones empresariales en el mundo actual.
- Proporcionar una guía para el desarrollo de aplicaciones Web sobre J2EE con soluciones libres.

Estos objetivos específicos pueden ser logrados a partir de las siguientes **tareas de investigación**:

- Realizar un estudio del arte sobre las tendencias y tecnologías actuales acerca de las arquitecturas de software sobre la plataforma J2EE.
- Estudiar la plataforma J2EE para el desarrollo de sistemas de gerencia.
- Estudiar herramientas para trabajar con la plataforma J2EE.
- Estudiar los frameworks más utilizados en la comunidad J2EE y su integración.
- Describir la aplicación del modelo de arquitectura Struts+Spring+Hibernate en el módulo Servicio Autónomo de la solución informática SAREN.

El **aporte práctico** del presente trabajo es que se logra proveer una guía detallada sobre un modelo de arquitectura basado en J2EE a partir de las tendencias actuales. Se presentan además los resultados prácticos luego de haber aplicado esta variante al desarrollo del sistema Servicio Autónomo, como parte de la solución SAREN implementada para el MIJ de la República Bolivariana de Venezuela. El propio sistema Servicio Autónomo representa un aporte ya que es algo novedoso en la República Bolivariana de Venezuela debido a que no existía anteriormente un sistema similar pues la institución Servicio Autónomo ahora es que se está gestando.

Desde el punto de vista metodológico, se emplearon los métodos empíricos de la observación y la experimentación para la investigación subyacente relacionada con el presente Trabajo de Diploma. La observación científica fue selectiva, dado que se enfoca hacia la moda de las arquitcturas J2EE; sistémica, dado que se realizó en diversos momentos por personas distintas un seguimiento de las tendencias arquitectónicas y a partir de ahí el modelo de arquitectura planteado inicialmente fue mejorado; y fue también objetiva, sustentada en la imparcialidad. Además, la observación científica realizada se

considera más bien externa, tomando como base las directrices que se encuentran en este ámbito a nivel mundial; pero también vale acotar que se realizó una observación interna, aunque en menor grado, relacionada con los intentos de crear arquitecturas web para J2EE anteriores dentro del marco de la universidad. Por otra parte, el método experimental influyó directamente relacionado con la sistematicidad de la observación, pues como se verá en el Capítulo 3 el modelo de arquitectura inicial no fue el que se toma definitivamente para desarrollar el sistema. Indudablemente la experimentación jugó un importante papel en este sentido y, no es menos cierto que estuvo muy relacionada con la experiencia del equipo de trabajo.

Este documento explica la satisfactoria experiencia del Servicio Autónomo en la adopción de varias de las últimas tecnologías más útiles disponibles en Java y ofrece una panorámica acerca de las posibles arquitecturas y tecnologías que se pueden emplear en la esfera de las aplicaciones corporativas sobre J2EE.

El presente trabajo se encuentra estructurado por tres capítulos y varios anexos, este incluye todo lo relacionado con el trabajo investigativo y práctico realizado:

1. **Capítulo 1: Fundamentación teórica.** En este apartado se pretende proporcionar los aspectos principales de lo que es una arquitectura de software, así como llegar a mostrar el estado del arte de las principales plataformas existentes para el desarrollo de aplicaciones empresariales, y las metodologías de desarrollo para guiar a las mismas.
2. **Capítulo 2: Elementos a considerar para desarrollar con J2EE.** Este capítulo se proyecta hacia la arquitectura de software empresarial sobre la plataforma J2EE. Se muestran las posibles herramientas, frameworks, y división en capas para el desarrollo con arquitecturas J2EE.
3. **Capítulo 3: Arquitectura J2EE aplicada en el módulo Servicio Autónomo.** Este capítulo se proyecta hacia la arquitectura de software empresarial sobre la plataforma J2EE empleada en el módulo de Servicio Autónomo del sistema SAREN. Se muestran las configuraciones necesarias para el funcionamiento del sistema y las ventajas y desventajas que brinda esta arquitectura, así como el análisis de los resultados ofrecidos por la misma.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se hace alusión a las cuestiones legales que amparan la creación del sistema SAREN en la República Bolivariana de Venezuela, y por ende del módulo gerencial Servicio Autónomo. Debido a que SAREN es un sistema de alta complejidad, se necesita una buena arquitectura de software sobre la cual sentar los cimientos de dicho sistema, por lo que se hace un estudio de la arquitectura de software, en especial de la arquitectura de software empresarial. Además se analizan las principales plataformas para el desarrollo de software empresarial y se llega a la conclusión de cual plataforma utilizar para el desarrollo del sistema.

1.2 Gobierno electrónico

La Organización para la Cooperación y el Desarrollo Económico (OCDE) ofrece la siguiente definición para Gobierno Electrónico: “la aplicación de tecnologías basadas en Internet para actividades comerciales y no comerciales en el seno de las Administraciones Públicas” (Serna 2002).

En “Factores estratégicos para desarrollar el gobierno electrónico en las Alcaldías de Venezuela” (G., DURANTE et al. 2004) se define que “el Gobierno Electrónico es el uso que hacen, de las modernas tecnologías de la información y las comunicaciones, los órganos de administración pública, en particular Internet para mejorar los servicios e información ofrecidos a los ciudadanos, incrementar la eficiencia y la eficacia de la gestión pública, proveer a las acciones del gobierno un marco de transparencia y crear mecanismos para facilitar la participación ciudadana en la toma de decisiones de la gestión pública”.

En “El gobierno electrónico en el Ministerio de Justicia de la República de Cuba” (Amoroso 2004) se plantea que “el gobierno electrónico consiste en la aplicación de las TIC[7] en la gestión hacia adentro (control de la entidad) y hacia fuera (servicios de cara al cliente) de los procesos que llevan a cabo los departamentos de administración pública. Es el resultado de un cambio radical en las relaciones entre gobierno-ciudadano (G2C), gobierno-negocios (G2B) y gobierno-gobierno (G2G)”.

[7] **Ver glosario**.

El desarrollo de estrategias integrales de gobierno electrónico requiere de una reflexión consciente de las dimensiones que supone y de su impacto en la sociedad a la que se dirige, abarcando desde su incidencia en la manera de informar y comunicarse, como en la manera de prestar servicios y de organizarse internamente.

1.2.1 Gobierno electrónico en Venezuela

Muchos países han iniciado la implementación de gobiernos electrónicos encaminados a mejorar la calidad de vida de sus ciudadanos, tomando la delantera Canadá según estudios realizados por la consultora Accenture[8], especializada en identificar áreas críticas con potencial para negocios de máximo impacto. Otros países han mostrado iniciativas, algunos de ellos por ejemplo son Colombia, Chile, entre otros. Venezuela se encuentra llevando a cabo proyectos que hacen una realidad el gobierno en línea donde se ha podido percibir que en materia jurídica se están realizando significativas acciones para desarrollar el Gobierno Electrónico, observándose hechos en cuanto a la implantación del mismo en los organismos del gobierno central, regional y municipal. El sistema de Servicio Autónomo respaldado por leyes encaminadas a la modernización y estandarización del Sistema Registral venezolano constituye un resultado tangible de lo expuesto anteriormente.

1.3 SAREN

El Servicio Autónomo de los Registros y Notarías (SAREN) surge a raíz de la Ley de Registro Público y del Notariado publicada en Gaceta Oficial el 13 de noviembre del 2001 (Venezuela 2001). Esta Ley, derogada por la Nueva Ley del Registro Público y del Notariado publicada en Gaceta Oficial el 22 de diciembre del 2006 (Venezuela 2006), en su artículo 14 planteaba: “Se crea la Dirección Nacional de Registros y del Notariado como servicio autónomo, sin personalidad jurídica, que depende jerárquicamente del Ministerio del Interior y Justicia (…)”. Por disposiciones del MIJ y dada la necesidad de instaurar un texto legal actualizado y moderno, funcionarios del Ministerio, se dieron a la tarea de recopilar la mayor cantidad de información necesaria y precisa con el objetivo de poner a tono y subsanar fallas que presenta la anterior Ley del Registro Público y que actualmente carece de reglamentación. Como resultado de esta investigación, surge la nueva Ley en virtud con el objeto de ”regular la

[8] **Accenture**. Es una compañía global de consultoría de gestión, servicios tecnológicos y búsqueda de recursos externos (outsourcing).

organización, el funcionamiento, la administración y las competencias de los registros principales, mercantiles, públicos y de las notarías" y contempla principios de modernización tales como la implementación de un solo texto legal para la función notarial y registral incluyendo el registro público, civil y mercantil, el sistema del folio real, firma digital, automatización de procesos, entre otros.

1.3.1 Objetivos del SAREN

Uno de los aspectos resaltantes de la nueva ley es la creación del Servicio Autónomo de Registros y Notarías que aunque se encontraba contemplado en la anterior no había sido desarrollado en su totalidad. La nueva Ley refiere a su creación en el Artículo 10 y plantea: "Se crea el Servicio Autónomo de Registros y Notarías, sin personalidad jurídica, que depende jerárquicamente del Ministerio del Interior y Justicia, y es el órgano encargado de forma autónoma de la planificación, organización, coordinación, inspección, vigilancia, procedimiento y control sobre todas las oficinas de registros y notarías del país. (…)".

Este organismo dependiente del MIJ, regirá de manera uniforme y centralizada todos los aspectos, efectos y alcances en materia registral y notarial así como también regirá el desarrollo de lo pautado en la nueva ley. Sus funciones abarcan además el nombramiento de los Registradores y Notarios, así como todo el personal necesario, según lo dispuesto en el Artículo 11: "El Servicio Autónomo (…) adoptará una política moderna de captación, estabilidad y desarrollo de su personal (…)", específicamente el MIJ será el encargado de la designación y remoción de los mismos. Además, debe encargarse de la reglamentación y del régimen de remuneración y salarios, respecto a ello la Nueva Ley refiere en su Artículo 13: "La remuneración de los registradores o registradoras y notarios o notarias y del resto de los funcionarios o funcionarias, adscritos al Servicio Autónomo de Registros y Notarías, será fijada mediante decreto dictado por el Presidente o Presidenta de la República de conformidad con la ley del Estatuto de la función pública". Es también deber de esta organización el efectivo funcionamiento de las oficinas, la integración y fuentes ordinarias de ingresos y el destino que se le dará a los mismos en el ejercicio de la actividad y el de los excedentes al final del ejercicio fiscal. En leyes anteriores se contemplaba la autonomía de cada Registro como servicios autónomos independientes, situación que ha sido eliminada por la ley sancionada. Cada Registro o Notaría dependerá directamente del Servicio Autónomo creado por la ley.

La modernización de los Registros y Notarías viene dada por la automatización-integración-estandarización de los procesos en cada una de las oficinas y la relación de dependencia que existe entre

estas y el Servicio Autónomo. La nueva ley advierte que “todos los soportes físicos del sistema registral y notarial actual se digitalizarán y se transferirán a las bases de datos correspondientes (…)” (Artículo 23). Coexistirán dos tipos de bases de datos según la ley: la base de datos nacional y las bases de datos regionales; la primera respaldando y consolidando todas las materias registrales y notariales correspondientes a los registros y notarías del país, para la segunda categoría “el Servicio Autónomo de Registros y Notarías determinará las entidades regionales donde se mantendrán las bases de datos que consolidarán y respaldarán la información de todas las materias correspondientes a los registros y notarías. Cada oficina de Registro y de Notaría mantendrá un sistema de información donde residirán los datos de su especialidad registral, notarial y los demás que señale el Reglamento de esta Ley”.

Se impone entonces la necesidad de modernización del sistema registral venezolano, donde la implementación de un Sistema para el Servicio Autónomo como núcleo integrador constituye un factor determinante para lograr resultados notorios del continuo desarrollo del gobierno electrónico en Venezuela. Para ello ocupa realizar un análisis de las tendencias y tecnologías más utilizadas actualmente a nivel mundial en el desarrollo de sistemas informáticos de gestión empresarial ya que se necesita una arquitectura sólida y eficiente para el desarrollo del sistema.

1.4 Aquitectura

Arquitectura es el arte de proyectar y producir y engloba, por tanto, no solo la capacidad de diseñar los espacios sino también la ciencia de construir los volúmenes necesarios. En su sentido más amplio, William Morris en *Los prospectos de la arquitectura en la civilización,* conferencia pronunciada en Londres el 10 de marzo de 1881 y recopilada en el libro *el Arte y el Socialismo* en 1947 (Morris 1999) dio la siguiente definición: «La arquitectura abarca la consideración de todo el ambiente físico que rodea la vida humana: no podemos sustraernos a ella mientras formemos parte en la civilización, porque la arquitectura es el conjunto de modificaciones y alteraciones introducidas en la superficie terrestre con objeto de satisfacer las necesidades humanas…»

Otra definición de arquitectura, en su sentido general, dice que «La arquitectura es el arte de construir. Se compone de dos partes, la teoría y la práctica. La teoría comprende: el arte propiamente dicho, las reglas sugeridas por el gusto, derivadas de la tradición, y la ciencia, que se funda sobre fórmulas constantes y absolutas. La práctica es la aplicación de la teoría a las necesidades; es la práctica la que pliega el arte y

la ciencia a la naturaleza de los materiales, al clima, a las costumbres de una época, a las necesidades de un período» (Duc 1868).

Resumiendo y teniendo en cuenta las definiciones anteriores es que se puede concluir que arquitectura es una ciencia intelectual y práctica dirigida a establecer racionalmente el buen uso y las proporciones de los artefactos y a conocer con la experiencia la naturaleza de los materiales que los componen. Se dice que la arquitectura descansa en tres principios fundamentales: la belleza, la firmeza y la utilidad; teniendo que existir un equilibrio visible entre estos tres elementos, sin sobrepasar ninguno a los otros. No tendría sentido alguno no aceptar estos principios.

1.4.1 Arquitectura de software

Ahora bien, si es muy importante la arquitectura desde el punto de vista de las edificaciones también lo es desde el punto de los sistemas informáticos. En los inicios de la era de la informática, la programación era considerada un arte, debido a las dificultades que entrañaba para la mayoría de las personas. En esos primeros tiempos, los desarrolladores se lanzaban directamente hacia el código con solo una idea general del negocio que se requería. En aquellos momentos los sistemas no eran ni una pequeña parte de lo complejos y extensos que son hoy en día, pero aún así muchos sistemas fracasaron. Con el desarrollo de la era digital y el creciente aumento de la información los sistemas fueron creciendo en complejidad e inevitablemente se hizo necesario el desarrollo de metodologías y técnicas o trucos para conseguir llevar a cabo las necesidades de los nuevos sistemas. En consecuencia, un conjunto de abstracciones coherentes y patrones que proporcionen el marco de referencia necesario para guiar la construcción del software para un sistema de información se puede llamar Arquitectura de Software. Entonces, queda que la arquitectura de software establece los fundamentos para que analistas y diseñadores de sistemas, programadores y todo el capital humano necesario para el desarrollo de un software trabajen en una línea común que permita alcanzar los objetivos y necesidades del sistema informático.

Una arquitectura de software se selecciona y diseña en base a algunos objetivos y restricciones. Los objetivos son aquellos prefijados para el sistema informático, pero no solamente los de tipo funcional, sino también aquellos otros objetivos como la mantenibilidad, auditabilidad, flexibilidad e interacción con otros sistemas y usuarios. Las restricciones son aquellas limitaciones derivadas de las tecnologías disponibles para implementar un sistema informático. Algunas arquitecturas son más factibles de ser implementadas con algunas tecnologías específicas mientras que ciertas otras tecnologías no son adecuadas para

determinadas arquitecturas. La arquitectura de software define, abstractamente, los componentes que llevan a cabo una tarea computarizada o digitalizada, sus interfaces y la comunicación entre ellos. Phillipe Kruchten explica que «La arquitectura de software tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como los requerimientos no funcionales, como la confiabilidad, portabilidad y disponibilidad» (Kruchten 1995).

En la vida real existen muchas definiciones de lo que podría ser la arquitectura de software, pero a ciencia cierta no parece que ninguna de ellas haya sido totalmente aceptada. En un sentido amplio se podría estar de acuerdo en que la arquitectura de software es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación y tiene el encargo de definir los módulos y responsabilidades que tendrán cada uno de esos módulos así como la interacción entre ellos. La definición oficial de arquitectura de software es la IEEE Std 1471-2000 (IEEE 2000), que reza así: «La arquitectura de software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán y los principios que sustentan su diseño y evolución». Según El Proceso Unificado de Desarrollo de Software (Jacobson, Booch et al. 2000), se puede definir como «el conjunto de decisiones significativas acerca de la organización de un sistema software, la selección de los elementos estructurales a partir de los cuales se compone el sistema y las interfaces entre ellos, junto con su comportamiento, tal y como se especifica en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y de comportamiento en subsistemas progresivamente mayores, y el estilo arquitectónico que guía esta organización (…)».

Haciendo un poco de historia y regresando a los orígenes, en los años de la década del 60 del siglo XX se comenzaba a acariciar el concepto de arquitectura de software en algunos círculos de investigación. En este caso está el ejemplo Edsger Wybe Dijkstra, científico de origen neerlandés que hizo grandes aportes al mundo de la computación. No obstante, toma popularidad en los años 1990, tras reconocerse la denominada crisis del software y como tema de interés de la incipiente disciplina de ingeniería de software. Esta llamada crisis del software estuvo dada por una baja calidad del software, tiempo y presupuesto excedido, confiabilidad cuestionable y altos requerimientos de personal para el desarrollo del software. El efecto producido era que el software no satisfacía los requerimientos ni las necesidades del cliente. Sobre esto influyó el aumento del poder computacional, la reducción en el costo del hardware, la

informatización de las empresas, el personal de desarrollo y mantenimiento no era el mismo, muchos cambios en el entorno tecnológico, económico y social. Esta crisis se manifestó a sí misma en varias formas: como proyectos gestionados con sobre presupuestos, con sobre tiempo y eran inmanejables, con un código difícil de mantener.

El objetivo primario de la arquitectura de software es el de aportar elementos que ayuden en la toma de decisiones y, al mismo tiempo, proporcionar conceptos y un lenguaje común que permitan la comunicación entre todos los que tienen parte en el proceso del desarrollo del software.

Toda arquitectura debe describir diversos aspectos del software. Por lo general, cada aspecto se describe de una manera más comprensible si se utilizan distintos modelos o vistas. Resulta importante destacar que cada uno de ellos constituye una descripción parcial de la misma arquitectura y es deseable que exista un cierto solapamiento entre ellos. Los modelos o vistas de una arquitectura pueden expresarse mediante uno o varios lenguajes, el más obvio es el lenguaje natural, pero existen otros lenguajes, los cuales son apropiados únicamente para el modelado. Afortunadamente existe cierto consenso en adoptar el Lenguaje Unificado de Modelado (UML, de sus siglas en inglés) como lenguaje único para todos los modelos. Sin embargo, un lenguaje de modelado generalista corre el peligro de no ser capaz de describir determinadas restricciones de un sistema informático o expresarlas incomprensiblemente. Es común que no sea necesario inventar, por decirlo de algún modo, una nueva arquitectura de software para cada sistema informático. Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso concreto. De esa forma, las arquitecturas más universalmente conocidas son:

- Monolítica: donde el software se estructura en grupos funcionales muy acoplados.
- Cliente-servidor: donde el software reparte su carga de cálculos en dos partes independientes pero sin reparto claro de funciones.
- Arquitectura por niveles o por capas: esta constituye una especialización de la arquitectura cliente-servidor, donde la carga se distribuye en partes con un reparto claro de las funciones y donde cada capa tiene relación con la siguiente.

En resumen, según “El Proceso Unificado de Desarrollo de Software” de Ivar Jacobson, Grady Booch y James Rumbaugh (Jacobson, Booch et al. 2000), la arquitectura de software es necesaria debido a que «un sistema grande y complejo requiere de una arquitectura para que los desarrolladores puedan tener

una visión común. Un sistema de software es difícil de abarcar visualmente porque no existe en un mundo de tres dimensiones». Es precisamente ahí donde encaja la necesidad de una arquitectura de software sólida y confiable.

1.4.2 Patrones de arquitectura

Los patrones de arquitectura expresan el esquema fundamental de organización para sistemas de software. Proveen un conjunto de subsistemas predefinidos; especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos. Los patrones de arquitectura representan el nivel más alto en el sistema de patrones propuesto en Pattern Oriented Software Architecture - Volume 1 (Buschmann, Meunier et al. 1996) (en adelante, POSA). Ayudan a especificar la estructura fundamental de una aplicación. Cada actividad de desarrollo es gobernada por esta estructura; por ejemplo, el diseño detallado de los subsistemas, la comunicación y colaboración entre diferentes partes del sistema. Cada patrón de arquitectura ayuda a conseguir una propiedad específica en el sistema global; por ejemplo, la adaptabilidad de la interfaz de usuario. Los patrones que dan soporte a características similares se agrupan en una misma categoría.

A continuación se muestra la categorización utilizada en dos de los sistemas de patrones de arquitectura más extendidos y celebrados: el de POSA y el de Pattern of Enterprise Application Architecture (en adelante, PEAA).

En el POSA, se divide a los patrones en las siguientes categorías:

- Estructura (From Mud to Structure)
- Sistemas Distribuidos (Distributed Systems)
- Sistemas Interactivos (Interactive Systems)
- Sistemas Adaptables (Adaptable Systems)

La Tabla 1.1 muestra la distribución de los patrones de POSA en las categorías enunciadas anteriormente:

Estructura	Sistemas Distribuidos	Sistemas Interactivos	Sistemas Adaptables
▪ Capas (Layers) ▪ Tuberías y Filtros	▪ (Broker)	▪ Modelo-Vista- Controlador (Model-View-Controller)	▪ (Microkernel) ▪ (Reflection)

(Pipes & Filtres) ▪ Pizarra (Blackboard)		▪ Presentación-Abstracción-Control (Presentation-Abstraction-Control)	

Tabla 1.1. Clasificación de patrones de arquitectura de POSA.

En PEAA, se describe una gran cantidad de patrones orientados a la arquitectura de aplicaciones empresariales, que se definen en las siguientes categorías:

- Patrones de Dominio Lógico (Domain Logic Patterns)
- Patrones de Mapeo Objeto / Relacional (Object / Relational Mapping Patterns)
- Patrones de Presentación Web (Web Presentation Patterns)
- Patrones de Distribución (Distribution Patterns)
- Patrones de Concurrencia fuera de Línea (Offline Concurrency Patterns)
- Patrones de Estado de Sesión (Session State Patterns)
- Patrones de Base (Base Patterns)

Las tablas Tabla 1.2 y Tabla 1.3 muestran la distribución de los patrones de PEAA en las categorías enunciadas anteriormente:

Dominio Lógico	**Mapeo Objeto / Relacional**	**Presentación Web**
▪ Transaction Script ▪ Domain Model ▪ Table Module ▪ Service Layer	*Patrones Arquitecturales de Fuentes de Datos* ▪ Table Data Gateway ▪ Row Data Gateway ▪ Active Record ▪ Data Mapper *Patrones de comportamiento Objeto / Relacionales* ▪ Unit of Work ▪ Identity Map	▪ Model View Controller ▪ Page Controller ▪ Front Controller ▪ Template View ▪ Transform View ▪ Two Step View ▪ Application Controller

	▪ Lazy Load *Patrones estructurales Objeto / Relacionales* ▪ Identity Field ▪ Foreign Key Mapping ▪ Association Table Mapping ▪ Dependent Mapping ▪ Embedded Value ▪ Serialized LOB ▪ Single Table Inheritance ▪ Class Table Inheritance ▪ Concrete Table Inheritance ▪ Inheritance Mappers *Patrones Objeto/Relacional para el mapeo de metadatos* ▪ Metadata Mapping Query Object ▪ Repository	

Tabla 1.2. Clasificación de patrones de arquitectura de aplicaciones empresariales de PEEA.

Distribución	**Concurrencia fuera de línea**	**Estado de Sesión**	**Base**
▪ Remote Facade ▪ Data Transfer Object	▪ Optimistic Offline Lock ▪ Pesimistic Offline Lock ▪ Coarse-Grained Lock ▪ Implicit Lock	▪ Client Session State ▪ Server Session State ▪ Database Session State	▪ Gateway ▪ Mapper ▪ Layer Supertype ▪ Separated Interface ▪ Registry

			▪ Value Object ▪ Money ▪ Special Case ▪ Plugin ▪ Service Stub ▪ Record Set

Tabla 1.3. Clasificación de patrones de arquitectura de aplicaciones empresariales de PEEA.

1.4.3 Arquitectura de software empresarial

Entonces, ¿qué es una arquitectura empresarial? Una posible definición abstracta de arquitectura empresarial podría ser el estudio de sistemas empresariales complejos desde el punto de vista de su estructura. La persona encargada de la arquitectura, normalmente el arquitecto de sistemas, ha de ser capaz de estudiar el problema en concreto y de escoger una serie de componentes a partir de los cuales modelar la arquitectura más adecuada para el problema en cuestión. El arquitecto, asimismo, debe ser capaz de establecer el modo de trabajo de dichos componentes, las herramientas utilizadas y las relaciones existentes entre los mismos. Quizás la labor de mayor complicación y responsabilidad para un arquitecto empresarial es la elección de la arquitectura sobre la cual se basará el sistema, puesto que una elección equivocada puede tener resultados catastróficos.

Una arquitectura de desarrollo de software empresarial ha de ofrecer una serie de servicios a los arquitectos y desarrolladores encaminados a facilitar la implementación de aplicaciones empresariales, al tiempo que brinda la mayor cantidad de funcionalidades a los usuarios. Normalmente tiene los siguientes requisitos:

- Escalabilidad: ha de ofrecer una buena escalabilidad tanto vertical como horizontal, de modo que se pueda aumentar la estructura tecnológica sin necesidad de realizar modificaciones.
- Mantenibilidad: ha de permitir la realización de modificaciones a los componentes existentes sin que se modifique el comportamiento del sistema.
- Fiabilidad.

- Disponibilidad: debe tener soporte de arquitecturas tolerantes a fallos y sistemas de redundancia y todo aquello que asegure que el sistema empresarial estará siempre disponible.
- Extensibilidad: ha de ser posible la añadidura de nuevos componentes y capacidades al sistema sin que se vean afectados el resto de los componentes.
- Manejabilidad: los sistemas han de ser fácilmente manejables y configurables.
- Seguridad: se han de tener buenos sistemas de seguridad, tanto a nivel de autenticación, como de autorización y de transporte de datos.
- Rendimiento: se han de ofrecer automáticamente soporte general de mecanismos que permitan aumentar el rendimiento de manera transparente al usuario.

Después de esto se puede llegar al punto de que la importancia de una arquitectura empresarial es que todos estos componentes se ofrecen casi automáticamente, logrando que el desarrollo del sistema sea más productivo.

1.5 Plataformas para el desarrollo de software empresarial

El desarrollo de aplicaciones empresariales, la colaboración tanto entre departamentos como entre empresas (programación distribuida) y el desarrollo de aplicaciones y servicios web han sufrido un auge muy importante durante los últimos años. Frente a esta nueva demanda surgen dos plataformas distintas para el desarrollo de este tipo de aplicaciones: J2EE de Sun Microsystems y .NET de Microsoft.

1.5.1 La plataforma J2EE

Antes de entrar al análisis de J2EE es necesario tener una breve idea acerca de lo que es Java. El lenguaje Java surge a principios de los años 90 en los laboratorios de Sun Microsystems[9]. A diferencia de los lenguajes convencionales, que generalmente están diseñados para ser compilados a código nativo, Java es compilado a un código intermedio o bytecode, el cual es interpretado por una máquina virtual de Java, en lo adelante máquina virtual. La máquina virtual hace posible que una aplicación que haya sido implementada en Java se ejecute en cualquier sistema operativo con soporte para la máquina virtual. La máquina virtual proporciona un entorno de ejecución que convierte el código neutro de Java al código nativo del ambiente en que está siendo ejecutada. Entonces, Java, como lenguaje de programación es

[9] **Sun Microsystems.** Empresa informática del Silicon Valley, fabricante de semiconductores y software, Java, Solaris, etc.

multipropósito, reúne todas las características de un ambiente orientado a objetos: es sencillo, cuenta con capacidad de generación de aplicaciones distribuidas, robusto, seguro, de arquitectura neutral, portable, multihilo, dinámico y de alto rendimiento. Pero esto no lo es todo, la API (Application Program Interface, interfaz de programas de aplicación) de Java es muy versátil, ya que está formada por un conjunto de paquetes de clases que le proporcionan una extensa funcionalidad. El núcleo de la API viene con cada una de las implementaciones de la máquina virtual: tipos de datos, clases y objetos, manejo de red, seguridad, componentes, etc. Estos componentes son llamados Java Beans, los cuales son código reusable que se pueden desarrollar fácilmente para crear aplicaciones sofisticadas. Se puede decir que con Java, Sun Microsystems introdujo en el mercado la primer plataforma de software universal diseñada desde y para el crecimiento de Internet y de las intranets corporativas. Esta tecnología permite escribir aplicaciones una sola vez y ejecutarlas en cualquier computadora, lo que, desde entonces ha revolucionado el mundo del desarrollo de software por representar un cambio de paradigma.

Java fue pensado originalmente para utilizarse en cualquier tipo de electrodoméstico pero la idea fracasó. Uno de los fundadores de Sun Microsystems rescató la idea para utilizarla en el ámbito de Internet y convirtieron a Java en un lenguaje potente, seguro y universal gracias a que lo puede utilizar todo el mundo y es gratuito. Uno de los primeros triunfos de Java fue que se integró en el navegador Netscape y permitía ejecutar programas dentro de una página web, hasta entonces impensable con el HTML.

Actualmente Java se utiliza en un amplio abanico de posibilidades y casi cualquier cosa que se puede hacer en cualquier lenguaje se puede hacer también en Java y muchas veces con grandes ventajas. Con Java se pueden programar páginas web dinámicas, con accesos a bases de datos, utilizando XML, con cualquier tipo de conexión de red entre cualquier sistema. En general, cualquier aplicación que se desee hacer con acceso a través de la web se puede hacer utilizando Java.

Debido a la necesidad del mercado de desarrollo de software de contar con medios y herramientas que permitan construir aplicaciones corporativas se diseñó la plataforma abierta y estándar de Java para este rubro, mejor conocida como J2EE (Java 2 Enterprise Edition, Java 2 edición empresarial). Se le denomina plataforma porque proporciona técnicas específicas que describen el lenguaje, pero, además, provee las herramientas para implementar productos de software basados en dichas especificaciones. Puede sonar muy complicado, pero no lo es.

La plataforma J2EE ha sido diseñada para aplicaciones distribuidas con base en componentes o unidades funcionales de software que interactúan entre sí para formar parte de una aplicación empresarial J2EE. Un componente de esta plataforma debe formar parte de una aplicación y ser desplegado en un contenedor, o sea, en la parte del servidor J2EE que le ofrece al componente ciertos servicios de bajo nivel y de sistema, tales como seguridad, manejo de concurrencia, persistencia y transacciones. Como se puede apreciar, J2EE no es solo una plataforma o una tecnología, sino un estándar de desarrollo, construcción y despliegue de aplicaciones.

J2EE ofrece muy buenas perspectivas para la implementación de software empresarial para aquellos sistemas informáticos que requieran basar su arquitectura en productos basados en software libre. J2EE ofrece, entre otras, las siguientes ventajas:

- Soporte para múltiples sistemas operativos: al ser una plataforma Java, es posible desarrollar arquitecturas basadas en J2EE usando cualquier sistema operativo donde pueda estarse ejecutando una máquina virtual de Java, teniendo la gran ventaja de una independencia total de la arquitectura de hardware.
- Organismo de control: J2EE está controlada por un organismo formado por más de 400 empresas. Entre esas empresas se encuentran muchas de las más importantes del mundo informático, tales como Sun Microsystems, IBM, Oracle, BEA, HP, AOL, etc.
- Competitividad: muchas empresas crean soluciones basadas en J2EE que ofrecen características tales como rendimiento y precio muy diferentes. De este modo, se ha desarrollado a un nivel exponencial la plataforma y los clientes tienen la posibilidad de escoger entre una gran cantidad de opciones.
- Madurez: creada en el año 1997, J2EE ya tiene varios años de vida y una amplia cantidad de proyectos importantes a sus espaldas.
- Soluciones libres: sobre la plataforma J2EE es posible crear arquitecturas basadas por completo en productos de software libre. No solo eso, sino que los arquitectos de software disponen de muchas soluciones libres para cada una de las partes de su arquitectura.

Existe mucha confusión, sobre todo entre la gente alejada del mundo de Java, sobre lo que es en realidad J2EE. La confusión más habitual es pensar que J2EE es un producto concreto que distribuye Sun Microsystems, como hace con su JDK, y que puede ser descargado desde su página web. Nada más lejos

de la realidad. No existe un J2EE concreto, por lo que no es posible ir a la página web de Sun Microsystems y descargar “el J2EE".

1.5.1.1 JSR

JSR es el acrónimo de Java Specification Request. Cuando una persona o entidad cree que es necesaria la presencia de una determinada tecnología dentro de las plataformas basadas en Java, lo que hace es crear un JSR y presentarlo para su aprobación. Dentro de este documento se relata por qué es necesaria dicha tecnología, por qué no se pueden abordar los problemas que soluciona con las tecnologías existentes.

Si dicha petición se aprueba entonces se crea una especificación, un documento en el cual se describe dicha tecnología, sus partes, las relaciones entre las mismas y los roles de las personas que usarán dicha tecnología. Además de la realización de este documento, el equipo encargado del desarrollo de la especificación ha de proporcionar un test de compatibilidad y una implememententación de referencia de los que se hablará posteriormente.

1.5.1.2 JCP

Java, siempre fue criticado por ser única y exclusivamente de Sun Microsystems. A raíz de todas esas críticas, Sun, el 8 de diciembre de 1998, decidió dar la posibilidad a todo el mundo de participar en la evolución de Java y de todas las plataformas que se basan en Java. Con esa intención se creó el JCP, que como se ha dicho es un organismo formado por alrededor de 500 empresas, asociaciones y particulares cuyo objetivo es asegurar la evolución de las plataformas basadas en Java.

Para entrar a formar parte del JCP las empresas e individuales han de pagar una cuota anual a Sun Microsystems. Cualquiera puede participar en el desarrollo de cualquier parte de la plataforma, previo pago de esa cuota, y por supuesto, si el comité de la especificación en la que quiere participar considera que esa persona o entidad tiene los conocimientos necesarios para aportar algún beneficio.

Una de las labores del JCP es la de controlar la evolución de las diferentes especificaciones que forman las plataformas basadas en Java. Este organismo es el encargado de decidir que especificaciones se aprueban y de controlar las fases por las que pasan.

1.5.1.3 J2EE

Después de ver los conceptos de JSR y JCP se puede definir lo que es J2EE. J2EE es una especificación, un JSR (concretamente el JSR-151), que define una plataforma de desarrollo empresarial, a la que se llama la plataforma J2EE. La plataforma J2EE está formada varios componentes:

- Un conjunto de especificaciones.
- Un test de compatibilidad, el J2EE Compatibility Test Suite (CTS).
- La implementación de referencia de J2EE.
- Un conjunto de guías de desarrollo y de prácticas aconsejadas denominadas J2EE BluePrints.

1.5.1.4 J2EE, una especificación de especificaciones

Como se ha dicho, la plataforma J2EE está definida por una especificación en formato electrónico. Esta especificación se encuentra bajo el JSR-151, y en ella se definen, de manera muy general, las pautas, reglas y servicios que han de seguir y ofrecer los diferentes servidores de aplicaciones que quieran implementar la plataforma J2EE. Además de eso, define también las normas generales que han de cumplir los desarrolladores que quieran crear aplicaciones empresariales compatibles con J2EE y los diferentes roles que éstos pueden tomar.

Los servicios que han de ofrecer los servidores de aplicaciones que implementen la plataforma J2EE están a su vez definidos por diferentes especificaciones. Estas especificaciones definen con mucho más detalle los diferentes componentes de los servidores de aplicaciones, como puedan ser un contenedor web, un servidor de mensajería, el sistema de seguridad, etc. De algún modo, se puede decir que la especificación J2EE engloba a un gran conjunto de especificaciones. Por poner un ejemplo. En la especificación de J2EE 1.5 se definen las siguientes especificaciones:

- JSR-109, (Servicios Web)
- JSR-101, (JAX-RPC, Java API for XML-based RPC)
- JSR-67, (JAXM, Java API for XML Messaging)
- JSR-93, (JAXR, Java API for XML Registries)
- JSR-77, (Configuración y control)
- JSR-88, (API de despliegue)
- JSR-115, (Interfaz de servicios de autorización)

- JSR-56, (JNLP, Ejecución remota de aplicaciones)
- JSR-112, (JCA 2.0, Arquitectura de conectores)
- JSR-152, (JSP 1.3, Java Server Pages)
- JSR-152, (Servlets 2.4)
- JSR-153, (EJB 2.1, Enterprise Java Beans)
- JSR-9XX, (JAXP 1.2, Soporte de esquemas XML)
- JSR-9XX, (JMS 1.1, API de mensajería)

Nótese que todas estas especificaciones tienen asociado un JSR, regido por un comité de empresas, asociaciones o individuos que se aseguran de crear dichas especificaciones y de que vayan evolucionando. Como se ha dicho anteriormente, cualquiera puede participar en la creación de estas especificaciones y por supuesto cualquiera puede descargarlas y leerlas cómodamente en su casa.

La gran importancia de toda esta enorme lista de especificaciones radica en que cuando se utiliza un servidor de aplicaciones que implementa la plataforma J2EE, los desarrolladores, obtienen de manera automática todos estos servicios. Es decir, se dispone de una gran caja de herramientas que se pueden aprovechar para realizar aplicaciones de una manera mucho más eficaz.

Por último, hay que señalar que cada una de estas especificaciones puede utilizarse perfectamente por separado. Por otra parte, cada una puede tener diferentes implementaciones que se pueden ofrecer como productos independientes, como por ejemplo Apache Tomcat (Servlets/JSP) o JORAM (JMS) y que por supuesto pueden tener la licencia que deseen, libre o propietaria.

1.5.1.5 El test de compatibilidad de J2EE

Es evidente que cualquier especificación regida por un JSR tiene que proporcionar un test de compatibilidad. Estos test ayudan a que los fabricantes que implementen la especificación puedan probar su compatibilidad con la misma. En el caso de J2EE este test se denomina J2EE CTS (Compatibility Test Suite).

El CTS es otra de las causas de confusión dentro del mundo de J2EE. Este test de compatibilidad consta de una serie de pruebas, más de 15000, cuyo objetivo es asegurar que un producto, típicamente un servidor de aplicaciones, está conforme con la especificación de J2EE. La finalidad de este test es clara:

asegurar que los productos tienen un mínimo de calidad, de modo que cumplan todas las normas de la especificación y con esto asegurar que no surjan implementaciones que desvirtúen a la plataforma.

1.5.1.6 La implementación de referencia

Además de ofrecer unos tests de compatibilidad, cualquier especificación ha de ofrecer también una implementación de referencia de la tecnología. Esta implementación sirve para que los desarrolladores tengan un producto con el que empezar a desarrollar y que al mismo tiempo puedan asegurarse de que estos desarrollos sean compatibles con la especificación.

La especificación de J2EE no es ninguna excepción y se dispone de una implementación de referencia, que se puede descargar desde la página de J2EE de Sun Microsystems. Esta implementación de referencia es la causa de que muchas personas piensen que J2EE es un producto que se puede descargar de la web de Sun Microsystems y que posee una licencia muy restrictiva.

1.5.1.7 J2EE BluePrints

Los J2EE BluePrints son un conjunto de documentos, que se pueden descargar desde la web de Sun Microsystems o encargar copias impresas de los mismos. Estos documentos definen patrones de diseño y prácticas recomendadas en la creación de aplicaciones utilizando J2EE. Dentro de ellos se puede encontrar como crear aplicaciones con JSP y Servlets de n-capas, que patrones de diseño utilizar al desarrollar con Enterprise Java Beans (EJB), etc.

1.5.1.8 El modelo de desarrollo J2EE

Para finalizar esta breve descripción de la plataforma J2EE es importante tener conocimiento de algunos conceptos básicos sobre el modelo de desarrollo de aplicaciones bajo esta plataforma. Abordar el modelo de desarrollo de J2EE podría invertir varios libros completos y aún podrían quedar varios aspectos sin tratar. La plataforma J2EE define un modelo de programación encaminado a la creación de aplicaciones basadas en n-capas. La lógica de la aplicación se divide en componentes de diferentes funciones que componen una aplicación J2EE y que están distribuidos en dependencia de la capa en el ambiente multi-capas J2EE al cual la aplicación pertenece. Aunque puede variar, típicamente una aplicación suele tener cinco capas diferentes:

- Capa cliente: representa la interfaz de usuario que maneja el cliente.

- Capa de presentación: representa el conjunto de componentes que generan la información que se mostrará en la interfaz de usuario del cliente. Normalmente se crea a través de componentes basados en Servlets y JSP.
- Capa de lógica de negocio: contiene los componentes de negocio reutilizables.
- Capa de integración: aquí se encuentran los componentes que permiten hacer más transparente el acceso a la capa de sistemas de información. Este es el lugar idóneo para implementar la lógica de objetos de acceso a datos (DAO, data access object).
- Capa de recursos: engloba los sistemas en los cuales la información se almacena físicamente como bases de datos relacionales, sistemas legacy, bases de datos orientadas a objetos, bancos de ficheros de datos, etc.

Las ventajas de un modelo como este son muy importantes. Al tener las capas separadas se tiene que existe poco acoplamiento entre las mismas, de modo que es mucho más simple hacer modificaciones en ellas sin que afecten a las demás. Todo esto redunda en la obtención de mejoras en cuanto a mantenibilidad, extensibilidad y reutilización de componentes. Otra ventaja que se obtiene es la de promover la heterogeneidad de los clientes, ya que añadir nuevos tipos de clientes se reduce a añadir nuevas capas de interfaz de usuario y presentación, sin necesidad de modificar el resto de las capas.

Como ya se ha dicho, el modelo de desarrollo con J2EE está basado en componentes reutilizables, con el objetivo de aumentar la reusabilidad de las aplicaciones. Estos componentes, además, gracias a las especificaciones, son intercambiables entre servidores de aplicaciones, por lo que la portabilidad de las aplicaciones es máxima.

1.5.1.9 La plataforma J2EE en el mundo corporativo

La plataforma J2EE resulta una propuesta atractiva, interesante y de vanguardia que responde, de manera natural, a la demanda actual para el desarrollo de software bajo el concepto de arquitectura en capas. Para una corporación que cuenta con una diversidad de equipos de cómputo, con sus respectivos sistemas operativos, una diversidad de ambientes de trabajo en las distintas áreas y una diversidad de entornos de desarrollos, los sistemas y soluciones informáticas con que se cuenta en cada lugar, la propuesta de J2EE para unificar el desarrollo de aplicaciones que puedan utilizarse en la empresa resulta

viable y con posibilidades de amplia aceptación, sobre todo por el incentivo principal de Java de «programar una vez y ejecutar en cualquier lugar».

1.5.2 La plataforma .NET

.NET es una plataforma de software que conecta información, sistemas, personas y dispositivos. La plataforma .NET conecta una grande variedad de tecnologías de uso personal y de negocios, de teléfonos celulares a servidores corporativos, permitiendo el acceso a información importante, donde y cuando se necesite.

Desarrollado con base en los estándares de Servicios Web XML, .NET permite que los sistemas y aplicaciones, ya sea nuevos o existentes, conecten sus datos y transacciones independientemente del sistema operativo, tipo de computadora o dispositivo móvil que se utilice, o del lenguaje de programación empleados para crearlo.

.NET es un "ingrediente" presente en toda la línea de productos Microsoft[10], ofreciendo la capacidad de desarrollar, implementar, administrar y utilizar soluciones conectadas a través de Servicios Web XML, de manera rápida, económica y segura. Estas soluciones permiten una integración más rápida y ágil entre las empresas y el acceso a información a cualquier hora, en cualquier lugar y a través de cualquier dispositivo.

La idea fundamental de Microsoft .NET es un cambio de enfoque en lo que es la informática, pasando de un mundo de aplicaciones, sitios Web y dispositivos aislados a una infinidad de computadoras, dispositivos, transacciones y servicios que se conectan directamente y trabajan en conjunto para ofrecer soluciones más amplias y ricas en contenido.

Las personas tendrán el control sobre cómo, cuándo y qué información desean. Las computadoras, sistemas y servicios estarán en capacidad de colaborar e interoperar mutuamente para beneficiar al usuario, mientras que las empresas podrán ofrecer sus productos y servicios a los clientes apropiados, en el momento correcto y de la forma precisa, combinando procesos de manera mucho más granular de lo que es posible hoy.

[10] **Microsoft.** Es una empresa informática de Estados Unidos, dueña y productora de los sistemas operativos Microsoft DOS y Microsoft Windows.

Los Servicios Web son la más innovadora tecnología para los negocios en la Web. Los Servicios Web XML utilizan tecnologías programables y reutilizables que aprovechan la flexibilidad de Internet. Con ellos es posible tener una infinidad de aplicaciones conectados en red, ya sea que se ejecuten en diferentes plataformas, proporcionando información a todos s sus clientes, socios de negocios y empleados. Los Servicios Web tienen como base un conjunto de estándares abiertos, incluyendo XML, SOAP, WSDL y UDDI, los cuales son controlados por el World Wide Web Consortium (W3C). Desde el punto de vista de un desarrollador, .NET facilita la escritura de sistemas capaces de conectar entre sí utilizando Microsoft Visual Studio .NET, .NET Framework y los XML Web Services.

1.5.3 Comparación entre J2EE y .NET

J2EE es una iniciativa de Sun Microsystems secundada por muchas más empresas como IBM, HP, BEA, ORACLE (en la actualidad la apoyan más de 400 empresas) y creada en 1996, mientras que .NET es la alternativa creada por el gigante Microsoft cuatro años más tarde. Para ello, ha creado una plataforma de desarrollo cerrada dentro de su paquete “Visual Studio .NET”, que incluye todo lo necesario para crear aplicaciones de este tipo. Sin embargo, J2EE es un conjunto de especificaciones definidas como estándar, que deben ser seguidas para desarrollar el producto, para ello es necesario adquirir una serie de recursos que, en su conjunto, permitirán desarrollar las aplicaciones: el JRE, el JDK (librerías), servidores web y de aplicaciones como IBM WebSphere, BEA Weblogic, Oracle9iAS, Sun ONE, JBoss, Apache Tomcat, Jetty, Resin, Apache Gerónimo, entornos de programación, etc. Por tanto, se muestra la primera diferencia entre estas dos plataformas es precisamente esa: .NET es un producto, mientras que J2EE es un conjunto de especificaciones que definen un estándar.

Ambas plataformas tienen un fin común pero metodologías sustancialmente diferentes a la hora de abordar problemas como la portabilidad, seguridad, etc. En el aspecto funcional ambas plataformas guardan varias similitudes; se programa en un lenguaje que luego se compila a un código intermedio (“Intermediate Language” en el caso de Microsoft .NET y “Bytecodes” en el caso de Java). Este código se ejecutará en un “entorno de ejecución” que transformará el lenguaje intermedio a código propio de la máquina en la que se corre la aplicación, Common Language Runtime (CLR) en Microsoft .NET y Java Runtime Environment (JRE) en J2EE.

Por otro lado, .NET cuenta con un ambiente de desarrollo llamado Visual Studio .NET, mientras que J2EE no es un producto de ninguna empresa, se trata en cambio de un estándar, una serie de reglas y pautas a

seguir, con lo que no cuenta con un entorno de desarrollo tipo “Visual Studio”. Como alternativa, son múltiples los productos que existen en el mercado ofreciendo entornos de desarrollo adecuados, tales como NetBeans de Sun Microsystems, Visual Café de WebGain, Visual Age for Java de IBM, IBM WebSphere, Eclipse, entre otros. Es bien sabido que la mayoría de estos entornos de desarrollo no ofrecen las facilidades de Visual Studio .NET.

En otro orden, una de las principales características de la plataforma .NET consiste en la posibilidad de programar los distintos componentes de una aplicación empleando distintos lenguajes (siempre que cumplan con los criterios de la Common Language Specification). Es posible programar en una gran cantidad de lenguajes como C# (su lenguaje estrella), Visual Basic, C++, Cobol, Delphi, etc. Pero .NET va más allá de soportar estos lenguajes; también ofrece plena interoperabilidad entre ellos, por lo que es posible construir un componente en un lenguaje, introducirlo en una aplicación escrita en otro distinto e incluso heredarlo y añadir nuevas características en un tercero. J2EE, por su parte, el único lenguaje que soporta es Java y es el que se tendrá que utilizar para desarrollo de todos los componentes. Existen sólo dos formas oficiales para acceder a la plataforma J2EE con otros lenguajes, la primera es a través de JNI (Java Native Interface) y la segunda es a través de la interoperabilidad que ofrece CORBA. Sobre esta gran diferencia entre las dos plataformas hay opiniones de todo tipo para elegir, algunas que parecen ser las más contundentes y precisas, son las siguientes:

A favor de la multiplicidad de lenguajes:

- Permite una migración más sencilla para antiguos programadores, reduciendo el tiempo de formación. Además, trabajar con un lenguaje conocido proporciona gran productividad individual.

En contra:

- La sencillez de mantenimiento se reduce. Si una aplicación está realizada en varios lenguajes se necesitan expertos en varios lenguajes para entenderla y mantenerla, aumentando los costes considerablemente. Además, trabajar con un lenguaje conocido proporciona gran productividad individual.
- La productividad del grupo decrece. Si los programadores utilizan lenguajes diferentes no pueden comunicar fácilmente sus conocimientos de unos a otros.

El rendimiento es uno de los temas más controvertidos a la hora de comparar estas dos plataformas. En aspectos fundamentales del entorno de ejecución, como son el rendimiento, la escalabilidad y la seguridad, J2EE sigue teniendo fama de estar muy por delante de .NET, y es por ello que en los entornos en los que éstos son los aspectos fundamentales (como por ejemplo los entornos transaccionales de las grandes multinacionales) suele ser la plataforma elegida. La experiencia y madurez de la plataforma juega a favor de J2EE, ya que salió al mercado varios años antes que .NET, y en todo este tiempo se han ido desarrollando multitud de productos y servicios al tiempo que se han ido corrigiendo errores y cubriendo las carencias y necesidades detectadas, por lo que hoy por hoy cuenta con una gama de productos altamente consolidados mientras que .NET tiene menos experiencia a sus espaldas.

Si bien se dice que J2EE presentaba cierta desventaja por el hecho de restringir la programación a un único lenguaje, Java, ahora se presenta la gran ventaja de esta plataforma respecto a su rival: la portabilidad, o la posibilidad de ejecutar las aplicaciones desarrolladas en cualquier sistema operativo y/o máquina del mercado, y he aquí entra a jugar su rol el ya mencionado «escribir una vez y correr en cualquier lugar». Los productos J2EE ofrecen mucha más portabilidad que .NET, que sólo está preparada para ejecutarse sobre plataformas Microsoft (Windows). También hay que señalar que, como era de prever, la plataforma de Microsoft está en vías de salvar esta circunstancia gracias a proyectos como MONO[11], un intento de crear un CLR para otras máquinas y sistemas.

La seguridad es, sin lugar a dudas, uno de los aspectos más importantes a la hora de evaluar las dos plataformas. J2EE y .NET proporcionan servicios de seguridad sencillos, aunque con enfoques diferentes. Ambas plataformas usan conceptos similares para manipular el acceso a los recursos por usuario y por código, basándose ambos en permisos. Además, se usa el concepto de perfiles en ambos. Mientras J2EE usa el concepto de “Perfiles Organizacionales” para delimitar responsabilidades a varios niveles del proceso de desarrollo y explotación (Product Provider, Application Component Provider, Application Assembler, Deployer y System’s Administrador, por defecto), .NET no define la jerarquía tan claramente. .NET proporciona un modelo sólido de seguridad mediante código tratado en el CLR, lo cual supone un peligro: la habilidad para ejecutar código no tratado confiere la posibilidad de traspasar la seguridad del CLR mediante llamadas directas a los APIs subyacentes del sistema operativo. Java, para ello, examina la procedencia de las clases mediante el Class-Loader, realizando una comprobación exhaustiva de éstas.

[11] **MONO.** Es un proyecto de código abierto para crear un grupo de herramientas libres, basadas en GNU/Linux y compatibles con .NET según lo especificado por el ECMA.

Aunque esto implica también que código firmado y confiado tiene acceso ilimitado a los recursos del sistema. Además, las llamadas de Java a código nativo (C/C++) mediante JNI confiere la posibilidad de traspasar la seguridad del JRE de una manera tan segura como se puede traspasar la seguridad de .NET ejecutando código no tratado en el CLR.

Uno de los más importantes retos para los distribuidores de Microsoft y J2EE al desarrollar sus respectivas plataformas es la manipulación segura de código obtenido de múltiples fuentes (fuera de la máquina local). Las funciones de verificación de código de la JVM están bastante maduras a estas alturas. Además, se ha aprendido de los errores cometidos en el pasado. El modelo CLR es similar, pero la implementación está relativamente sin probar.

Con todo esto, parece que ambas plataformas han llegado a un sistema de seguridad bastante aceptable, aunque citando a Vince Dovydaitis, ingeniero jefe de Foliage Software Systems Inc.: «J2EE ofrece una mejor solución para grandes sistemas que corren mediante aplicaciones críticas y múltiples plataformas remotas, mientras que .NET ofrece mejor respuesta para gestionar autorizaciones basadas en usuarios y roles».

A este punto es muy conveniente tener en cuenta la total posibilidad de crear una arquitectura para un software empresarial de altas prestaciones con soluciones completamente libres en J2EE, lo cual, sobre .NET es completamente imposible. Vale destacar la presencia del proyecto MONO, el cual es un proyecto de código abierto impulsado por Novell para crear un grupo de herramientas libres, basadas en GNU/Linux y compatibles con .NET. En otras palabras, MONO podría ser la estrategia de crear un .NET libre. Este proyecto tiene tantos defensores como detractores y solo el tiempo tiene la última palabra, aunque aún le falta mucho por andar y como punto significativo tomar la experiencia que como todo nuevo proyecto requiere para tener un producto sólido y probado.

Luego de un análisis en cuanto al licenciamiento de ambas plataformas y teniendo en cuenta la necesidad de tomar una plataforma de desarrollo que se ajustara a los términos legales venezolanos, dado que el módulo Servicio Autónomo está destinado para un organismo de la administración pública (Venezuela 2004), se toma la decisión de utilizar J2EE, teniendo a su vez la ventaja de que el grupo de desarrolladores contaba con conocimientos y experiencia.

1.6 Metodología de desarrollo

Se definió que la metodología para la modelación y desarrollo del sistema Servicio Autónomo fuese RUP (Rational Unified Process), y que como lenguaje de modelado se utilizara UML (Unified Modeling Languaje) o Lenguaje de Modelación Unificado. A continuación se mencionan algunos aspectos de la metodología de desarrollo y lenguaje de modelado. Para abundar sobre estos temas se puede referir al trabajo “Modelación del Sistema del Servicio Autónomo para la gestión y control de los Registros y Notarías en la República Bolivariana de Venezuela”(Vázquez and Bombalier 2007), en la cual se muestra la modelación de dicho sistema.

1.6.1 RUP y UML

El Proceso Unificado es un proceso de desarrollo de software. Un proceso de desarrollo de software es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software. Sin embargo, el Proceso Unificado es más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto. El refinamiento más conocido y documentado del Proceso Unificado es el Proceso Unificado de Rational o simplemente RUP.

RUP está basado en componentes, lo cual quiere decir que el sistema software en construcción está formado por componentes software interconectados a través de interfaces bien definidas. RUP utiliza el Lenguaje Unificado de Modelado (Unified Modeling Language, UML) para preparar todos los esquemas de un sistema software. De hecho, UML es una parte esencial de RUP.

No obstante, los verdaderos aspectos definitorios de RUP se resumen en tres frases claves, dirigido por casos de uso, centrado en la arquitectura, e iterativo e incremental. Esto es lo que hace único al Proceso Unificado.

RUP está compuesto por un conjunto de flujos de trabajo separados en dos grupos, los flujos de trabajo de ingeniería que son los siguientes, Modelamiento del negocio, Requerimientos, Análisis y diseño, Implementación, Prueba, y Despliegue, en el otro grupo están los flujos de trabajo de apoyo que son los

siguientes, Administración del proyecto, Configuración y control de cambios, y Entorno, y cada uno de estos flujos está presente en las cuatro fases, Inicio, Elaboración, Construcción y Transición.

UML es un lenguaje gráfico para especificar, construir, visualizar y documentar las partes o artefactos (información que se utiliza o produce mediante un proceso de software). Pueden ser artefactos: un modelo, una descripción que comprende el desarrollo de software que se basen en el enfoque Orientado a Objetos, utilizándose también en el diseño Web. UML usa procesos de otras metodologías, aprovechando la experiencia de sus creadores, eliminó los componentes que resultaban de poca utilidad práctica y añadió nuevos elementos.

UML es más expresivo, claro y uniforme que los anteriores definidos para el diseño Orientado a Objetos, que no garantiza el éxito de los proyectos pero si mejora sustancialmente el desarrollo de los mismos, al permitir una nueva y fuerte integración entre las herramientas, los procesos y los dominios.

UML es desde finales de 1997, un lenguaje de modelado orientado a objetos estándar, de acuerdo con el Object Management Group, siendo utilizado diariamente por grandes organizaciones como: Microsoft, Oracle, Rational.

CAPÍTULO 2: ELEMENTOS A CONSIDERAR PARA DESARROLLAR CON J2EE

2.1 Introducción

Una manera actual de enfrentar el desarrollo de sistemas informáticos es el uso de aplicaciones Web, las cuales basándose en la arquitectura cliente/servidor proveen una manera sencilla de interacción con poca o casi ninguna instalación de software adicional en el cliente, facilitando de esta forma el despliegue, mantenimiento y actualización de este tipo de sistemas. Su facilidad de administración centralizada las hace ideales tanto para su despliegue en Internet como en intranets corporativas. La facilidad de uso de los interfaces web y el hecho de que cada día más personas están acostumbradas a la navegación por internet hace que el tiempo de aprendizaje se reduzca considerablemente respecto de aplicaciones tradicionales standalone. El auge de multitud de soluciones o frameworks open source hace que su desarrollo sea sencillo y que un gran número de desarrolladores tengan experiencia con ellos.

2.2 Aplicaciones en capas

La estrategia tradicional de utilizar aplicaciones compactas causa gran cantidad de problemas de integración en sistemas software complejos como pueden ser los sistemas de gestión de una empresa o los sistemas de información integrados consistentes en más de una aplicación. Estas aplicaciones suelen encontrarse con importantes problemas de escalabilidad, disponibilidad, seguridad, integración, etc. Para solventar estos problemas se ha generalizado la división de las aplicaciones en capas que normalmente serán tres: una capa que servirá para guardar los datos (base de datos), una capa para centralizar la lógica de negocio (modelo) y por último una interfaz gráfica que facilite al usuario el uso del sistema.

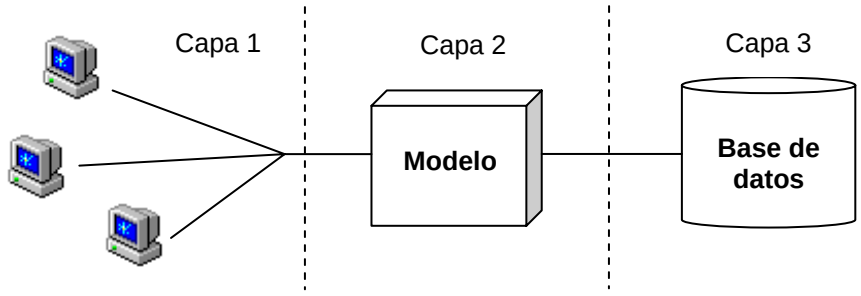

Figura 2.1. Arquitectura en tres capas.

Si se establece una separación entre la capa de interfaz gráfica (cliente), replicada en cada uno de los entornos de usuario, y la capa modelo, que quedaría centralizada en un servidor de aplicaciones, según la figura 2.1 se obtiene una potente arquitectura que otorga algunas ventajas:

- Centralización de los aspectos de seguridad y transaccionalidad, que serían responsabilidad del modelo.
- No replicación de lógica de negocio en los clientes: esto permite que las modificaciones y mejoras sean automáticamente aprovechadas por el conjunto de los usuarios, reduciendo los costes de mantenimiento.
- Mayor sencillez de los clientes.

Si se aplica esto a las aplicaciones web, debido a la obligatoria sencillez del software cliente que será un navegador web, se encuentra con una doble posibilidad:

- Crear un modelo de 4 capas, tal y como puede verse en la Figura 2.2, “Arquitectura web en cuatro capas”, separando cliente, servidor web, modelo y almacén de datos. Esto nos permite una mayor extensibilidad en caso de que existan también clientes no web en el sistema, que trabajarían directamente contra el servidor del modelo.

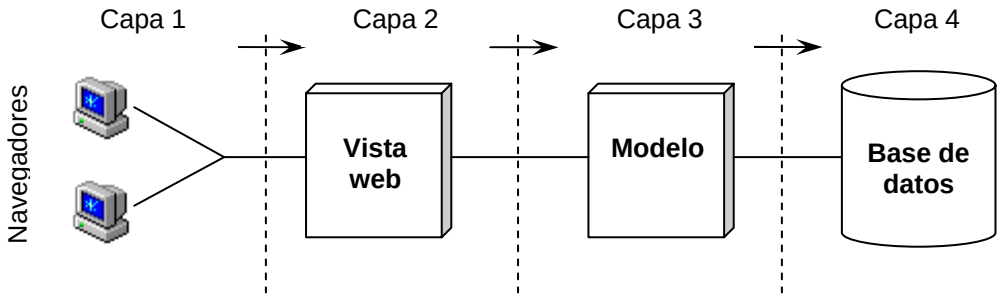

Figura 2.2. Arquitectura web en cuatro capas.

- Mantener el número de capas en 3, como se ve en la Figura 2.3, “Arquitectura web en 3 capas”, integrando interfaz web y modelo en un mismo servidor aunque conservando su independencia funcional. Ésta es la distribución en capas más común en las aplicaciones web.

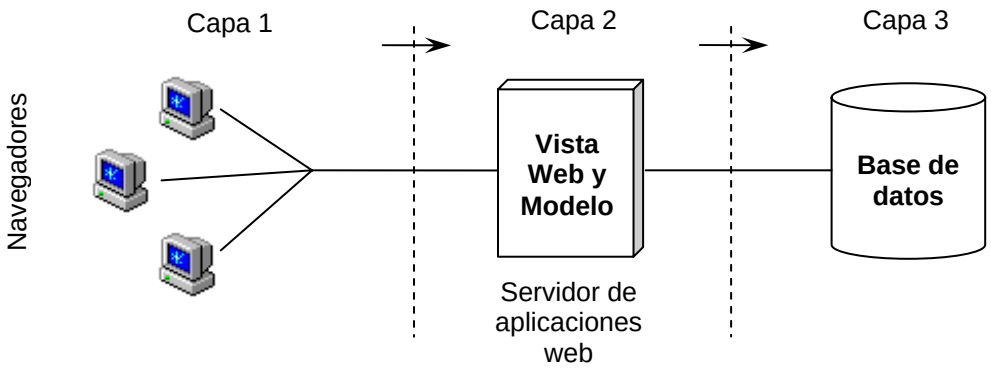

Figura 2.3. Arquitectura web en tres capas.

Para la realización del sistema Servicio Autónomo se decide tomar la variante de arquitectura web en tres capas.

2.3 Patrones de diseño para aplicaciones basadas en J2EE

Los analistas y desarrolladores van creando a diario habilidades para resolver problemas usuales. A mayor experiencia, mayor será el abanico de posibilidades para resolver los problemas que se presentan en el proceso de desarrollo de software. Los patrones de diseño tratan los problemas del diseño que se presentan y se repiten en situaciones particulares, con el fin de proveer el vehículo para ofrecer la solución. Por lo tanto, los patrones de diseño son soluciones exitosas a problemas comunes. Un patrón de diseño es una abstracción de una solución en un alto nivel. Hay patrones que abarcan las distintas etapas del desarrollo: desde el análisis hasta el diseño y desde la arquitectura hasta la implementación.

J2EE trajo consigo todo un nuevo catálogo de patrones de diseño. Desde que J2EE es una arquitectura por sí misma que involucra otras arquitecturas, merece su propio conjunto de patrones específicos para las diferentes aplicaciones empresariales. El libro “J2EE PATTERNS Best Practices and Design Strategies” (Alur, Crupi et al. 2003) (Patrones J2EE, Buenas prácticas y estrategias de diseño) explica quince patrones J2EE divididos entre las capas de negocio, integración y presentación, de los cuales, a continuación serán descritos los más usuales en el mundo de desarrollo de aplicaciones empresariales en J2EE de la actualidad.

- Model-View-Controller (modelo-vista-controlador): más conocido como MVC, es un patrón de diseño clásico utilizado con frecuencia en aplicaciones que necesiten la habilidad de mantener múltiples vistas de la información. El modelo es la representación específica de la lógica del negocio. La vista representa el modelo en un formato adecuado para interactuar con el usuario. El controlador responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y la vista. Esto se ve mejor en la siguiente figura:

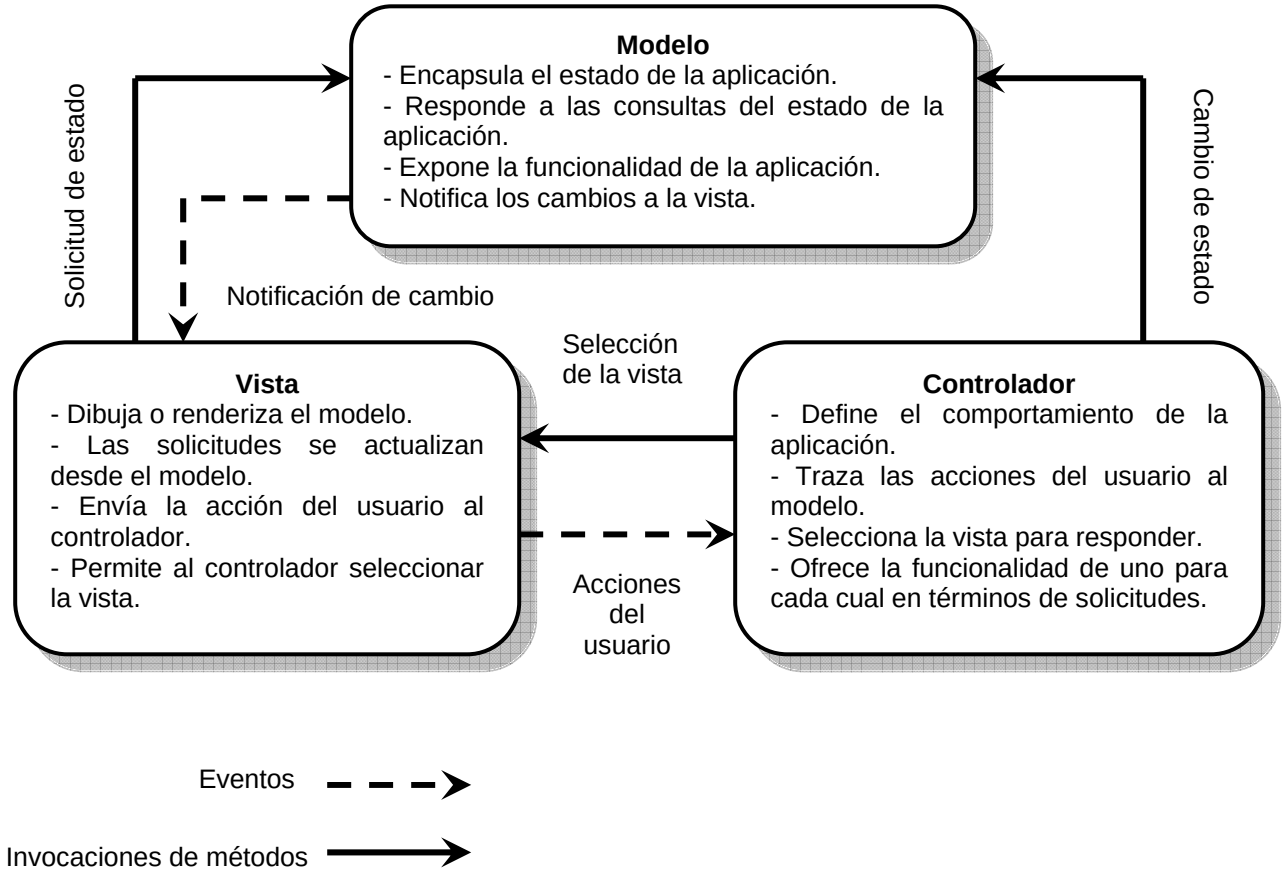

Figura 2.4. Esquema del patrón Modelo-Vista-Controlador.

La siguiente figura expresa como funciona esto en una aplicación web J2EE:

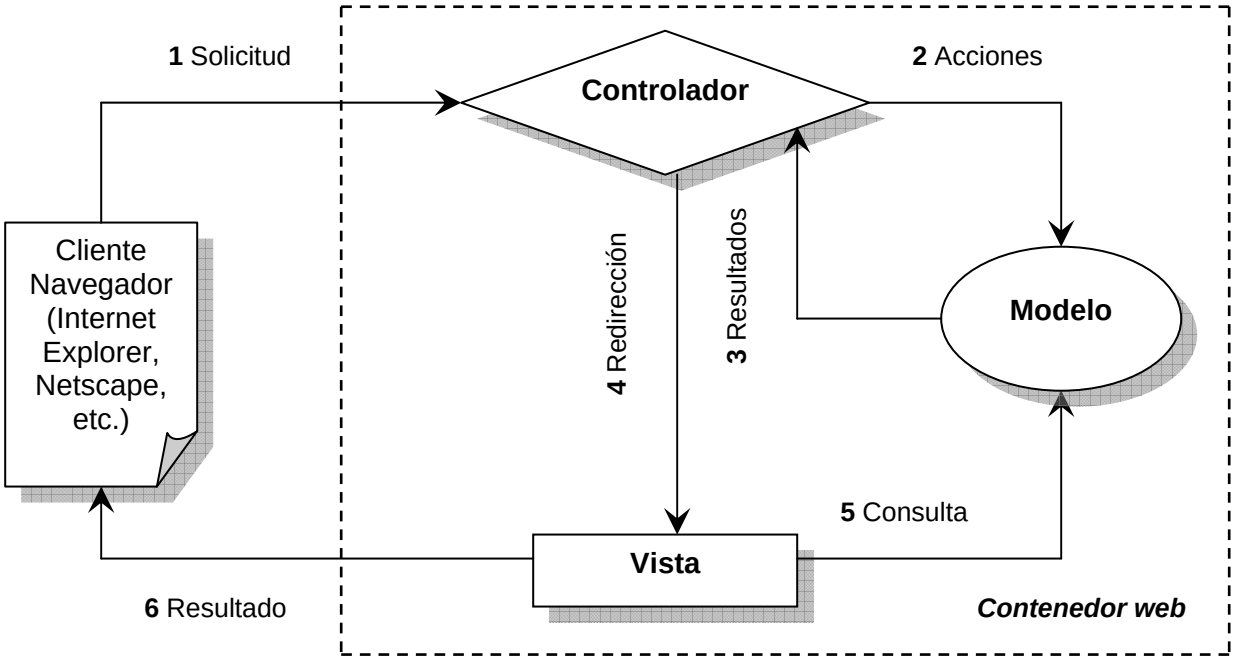

Figura 2.5. Patrón Modelo-Vista-Controlador en una aplicación web sobre J2EE.

El navegador genera una solicitud que es atendida por el controlador. El mismo se encarga de analizar la solicitud, seguir la configuración que se le ha programado y llamar al método que le corresponde en el modelo pasándole los parámetros enviados. Dicho método instanciará y/o utilizará los objetos de negocio para concretar la tarea. Según el resultado que se retorne, el controlador derivará la generación de interfaz a una o más páginas, las cuales podrán consultar los objetos del modelo a fin de concretar su tarea.

Patrones en la capa de presentación:

- Intercepting Filter (filtro que intercepta): se crean filtros para interceptar y procesar las peticiones entrantes y las respuestas salientes de un cliente web, permitiendo un pre y post-procesamiento de las mismas.
- Front Controller/Front Component (controlador frontal/componente frontal): un objeto que acepta todos los requerimientos de un cliente y los dirige a los manejadores apropiados. Si actúa en dos

sentidos el objeto Front Controller acepta los requerimientos y el objeto Dispatcher (entregador) los direcciona a los manejadores apropiados.

- View Helper (ayudante de vista): un objeto Helper que encapsula la lógica de acceso a datos en beneficio de los componentes de la presentación.
- Composite View (vista compuesta): un objeto Vista que está compuesto de otros objetos vista. Por ejemplo, una página JSP que incluye otras páginas JSP y HTML usando la directiva include o el action include.
- Service to Worker (servicio al trabajador): Es como el patrón de diseño MVC con el Controlador actuando como Front Controller pero con una cosa importante: aquí el Dispatcher (el cual es parte del Front Controller) usa View Helpers a gran escala y ayuda en el manejo de la vista.
- Dispatcher View (vista del entregador): Es como el patrón de diseño MVC con el controlador actuando como Front Controller pero con un aSunto importante: aquí el Dispatcher (el cual es parte del Front Controller) no usa View Helpers y realiza muy poco trabajo en el manejo de la vista. El manejo de la vista es manejado por los mismos componentes de la vista.

Patrones en la capa de negocio:

- Session Facade (fachada de sesión): Usar un bean de sesión como una fachada para encapsular la complejidad de las interacciones entre los objetos de negocio participantes en un flujo de trabajo. Este maneja los objetos de negocio y proporciona un servicio de acceso uniforme a los clientes.
- Business Delegate (delegado del negocio): Un objeto que reside en la capa de presentación y en beneficio de los otros componentes de la capa de presentación llama a métodos remotos en los objetos de la capa de negocios.
- Transfer Object (objeto de transferencia): Un objeto serializable para la transferencia de datos.
- Transfer Object Assembler (ensamblador de objetos de transferencia): Un objeto que reside en la capa de negocios y crea Transfer Objets cuando es requerido.
- Service Locator (localizador de servicios): consiste en emplear un objeto Locator para abstraer la utilización y ocultar las complejidades de la creación del contexto inicial. Varios clientes pueden reutilizar el objeto Locator para reducir la complejidad del código, proporcionando un punto de control.

- Value List Handler (manejador de lista de valores): se utiliza para controlar la búsqueda, hacer un caché con los resultados, y proporcionar los resultados al cliente en una hoja de resultados cuyo tamaño y desplazamiento cumpla con los requerimientos del cliente.
- Composite Entity (entidad compuesta): se utiliza para modelar, representar y manejar un conjunto de objetos persistentes relacionados en vez de representarlos como beans de entidad específicos individuales.

Patrones en la capa de integración:

- Data Access Object (objeto de acceso a datos): más conocido como DAO, consiste en utilizar un DAO para abstraer y encapsular todos los accesos a las fuentes de datos. El DAO maneja la conexión con la fuente de datos para obtener o almacenar datos.
- Service Activator (activador de servicios): se utiliza para recibir peticiones y mensajes asíncronos de los clientes. Cuando se recibe un mensaje, el Service Activator localiza e invoca a los métodos de los componentes de negocio necesarios para cumplir la petición de forma asíncrona.

2.4 Frameworks para J2EE

En el Cambridge Advanced Learner's Dictionary (Cambridge) se puede obtener la siguiente definición de framework: «una estructura de soporte en torno a la cual se puede construir algo». Por otro lado, un framework, según el punto de vista de Ralph Jonson, jefe del Departamento de Ciencias de la Computación en la Universidad de Illinois, es «un diseño reutilizable expresado como un conjunto de clases abstractas y la manera en que sus instancias se relacionan entre sí. Es un diseño reutilizable para toda o parte de una arquitectura de un sistema informático». Otra definición es: «Infraestructura de software que crea un entorno común para integrar aplicaciones e información compartida dentro de un dominio dado», SEMATECH 1998. Y de acuerdo con Fayed et Al «Es una aplicación semicompleta que contiene componentes estáticos y dinámicos que pueden ser personalizados para obtener aplicaciones específicas». Por otra parte, Javier Antoniucci, en "Manual Básico de Struts" (http://www.programacion.net/java/tutorial/joa_struts/1/) plantea que «un framework es la extensión de un lenguaje mediante una o más jerarquías de clases que implementan una funcionalidad y que, opcionalmente pueden ser extendidas».

Finalmente, y luego de leer las definiciones anteriores se puede concluir que un framework para el desarrollo de aplicaciones web es un conjunto de clases que cooperan y forman un diseño reutilizable formando una infraestructura que facilita y agiliza el desarrollo de aplicaciones web y sobre los cuales se pueden basar las arquitecturas de desarrollo de software empresarial.

Parafraseando el antiguo adagio de que para saber hacia dónde se va siempre es bueno conocer exactamente donde se está. Esto es aplicable también a J2EE. En los últimos años, paralelamente al desarrollo de Internet y al aumento del número de sus usuarios, el desarrollo de aplicaciones web ha sufrido un gran auge. El hecho de que una aplicación pueda ser utilizada desde cualquier lugar, sin necesidad de realizar ningún tipo de instalación en el ordenador cliente proporciona un valor añadido a este tipo de aplicaciones. Son muchas las empresas cuyo funcionamiento se centra en torno a compartir información por medio de aplicaciones web que funcionan en un ámbito corporativo. Este auge en el desarrollo de aplicaciones web ha venido acompañado del desarrollo de una gran cantidad de frameworks que intentan facilitar la implementación y eliminar las tareas repetitivas en el desarrollo de ese tipo de software.

La mayor parte de los problemas que se presentan en el desarrollo de aplicaciones web no existen en las aplicaciones de escritorio y se deben, fundamentalmente, a que el protocolo HTTP, en el cual se basa la comunicación cliente-servidor, es un protocolo sin estado. Por lo tanto, debe realizarse un trabajo extra en la parte del servidor para mantener una conversación con el cliente durante todo el tiempo que exista la sesión.

Entre los principales problemas que pretenden resolver los frameworks en el desarrollo de aplicaciones web se pueden encontrar los siguientes:

- Validación y procesamiento de formularios: la mayor parte de las entradas de información a las aplicaciones web se producen a partir de formularios web, cuyos datos hay que validar e informar al usuario si se ha encontrado algún error.
- Seguridad y control de acceso: una aplicación web puede ser accedida desde, potencialmente, cualquier lugar y por ende es necesario aumentar las medidas de seguridad para, entre otras cosas garantizar la integridad de la información.

- Separación entre desarrollador y diseñador: las aplicaciones web ofrecen la posibilidad de crear interfaces ricas en gráficos, por lo que es muy necesario que se trabaje en paralelo entre los diseñadores gráficos y los desarrolladores.

Además, cada framework puede especializarse en un tipo de aplicación web, y por tanto surgen distintos tipos, los que combinados forman la mayoría de las arquitecturas en el desarrollo de las actuales aplicaciones web.

2.4.1 Panorama actual de los frameworks J2EE

Teniendo en cuenta todo lo anteriormente planteado se puede proseguir hacia la descripción de los principales frameworks libres y de código abierto que existen en la actualidad, comenzando por los especializados en la capa de vista. Las aplicaciones web difieren de los sitios web convencionales en que las primeras pueden generar respuestas dinámicas, mientras que lo usual en los sitios web es que estén formados por contenido estático. Una aplicación web, por su parte, puede interactuar con motores de gestión de lógica de negocio, los que a la vez interactúan con sistemas de respaldo de información en físico para personalizar las respuestas.

2.4.2 Tecnología JSP-Servlet

Antes de comenzar a hablar de los frameworks es necesario tener una referencia a lo que es la tecnología de JSP-Servlets. En el anexo 1 se puede observar un modelo de objetos en UML mostrando lo que podría llamarse el framework Servlet.

Las JSP (JavaServer Pages) son una tecnología diseñada por Sun Microsystems para crear aplicaciones web. Son las páginas del lado del servidor empleadas en la capa de presentación de la plataforma J2EE, permitiendo incrustar código Java dentro del contenido estático de la página, lo que otorga dinamismo a la página al ser proyectada en el navegador del cliente. Además de esto, las JSP se pueden integrar directamente con clases Java, lo que permite separar por niveles las aplicaciones web, almacenando en clases aquellos comportamientos y funcionalidades que consumen mayor cantidad de recursos del lado del servidor. Su funcionamiento es a partir de etiquetas HTML convencionales y etiquetas especiales para el código Java. Los Servlets son objetos que se ejecutan dentro del contexto de un contenedor de servlets, normalmente los servidores o contenedores web. Los servlets tienen como principal función

obtener los datos entrados en los formularios web, procesar la información y generar páginas JSP de manera dinámica o enviar parámetros de respuesta a las JSP existentes como resultado de las solicitudes del cliente. Las páginas JSP, desde el punto de vista de la máquina virtual, son tenidas como serlvets.

La mayoría de los frameworks que emplean el Modelo 2, (modificación que hizo Sun Microsystems al MVC específicamente para J2EE) tienen como controlador frontal y núcleo un servlet genérico.

2.4.3 Struts

Struts es un subproyecto del proyecto Apache y es un framework libre y de código abierto para la creación de aplicaciones web en Java. Se puede decir que Struts ha sido el framework que ofreció un camino para el MVC en Java para desarrollos web. El núcleo de Struts es una capa flexible de control basada en las tecnologías estándares. Este framework soporta, como se menciona anteriormente, arquitecturas basadas en la metodología del Modelo 2, el cual es una variación específicamente para J2EE realizada por Sun Microsystems al clásico patrón MVC, el cual introdujo el patrón Front Controller, como se muestra en la figura 2.5. Citando una nota expuesta en el libro Mastering Jakarta Struts, de James Goodwill, se dice que existe un pequeño debate en la comunidad de desarrollo hacia el tipo de patrones de diseño a los que más Struts se acerca. De acuerdo con la documentación provista por los actuales desarrolladores del proyecto Struts, el patrón principal tras el cual se basa el framework es el Modelo-Vista-Controlador, pero algunos insisten en que a lo que más se acerca Struts es el patrón Front Controller, o controlador frontal, descrito por el programa J2EE Blueprints, de Sun Microsystems. Luego de haberlo estudiado y empleado, se puede concluir que la verdad es que en realidad Struts tiene mucho de haber sido inspirado en el Modelo 2, de Sun Microsystems. En la siguiente figura se puede observar la implementación de Struts del patrón Modelo-Vista-Controlador.

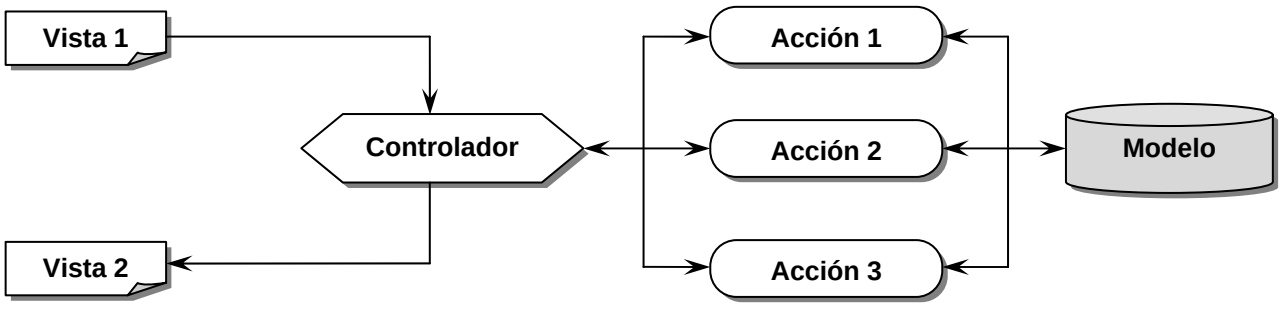

Figura 2.6. La implementación del MVC que hace Struts.

El framework provee tres componentes primordiales:

- Un manipulador de solicitudes provisto por la aplicación.
- Un manipulador de respuestas que transfiere el control a otro recurso que completa la respuesta.
- Una librería de etiquetas, o tags, que ayuda a los desarrolladores a crear aplicaciones interactivas basadas en formularios con páginas JSP del lado del servidor.

La arquitectura del framework y las tags que crean las páginas JSP son fácilmente comprensibles, en el anexo 2 se muestra un diagrama en UML de la arquitectura de Struts.

XML es el eXtensible Markup Lenguaje (lenguaje extensible de marcas) y se ha convertido en el lenguaje omnipresente en todas las aplicaciones web. Los ficheros XML están siendo comúnmente utilizados como archivos de configuración de las aplicaciones y Struts no es una excepción. Los dos ficheros XML de configuración para Struts son el WEB.XML, el cual es usado en la configuración de la aplicación web y el STRUTS-CONFIG.XML, el cual es utilizado para configurar todas las acciones que llevará a cabo la aplicación en Struts.

Al utilizar Struts primeramente los desarrolladores se encuentran con los beneficios del MVC. Sin embargo, Struts ofrece un significativo número de ventajas además de lo anteriormente dicho. He aquí un sumario de las mismas:

- Configuración centralizada basada en archivos: muchos de los valores de Struts están representados en XML y archivos de propiedades, lo que significa que muchos cambios puede ser realizados sin necesidad de modificar o recompilar código Java; es suficiente editar un simple archivo. Esto, además, permite a los desarrolladores enfocarse en sus tareas específicas.
- Librerías de etiquetas o tags: permite que se puedan crear interfaces JSP rápidamente y de menor peso para el servidor.
- Struts es libre y de código abierto: puede ser utilizado libremente y su código puede ser modificado en dependencia de las necesidades del usuario, además de que permite ver cómo fue programado.
- Todos los componentes y clases permiten ser heredados y son reemplazables.
- Validación de formularios: Struts contiene capacidades para chequear y validar la entrada de valores.

- Amplio tiempo de vida: dado a que Struts existe hace varios años, es posible y no difícil encontrar desarrolladores experimentados en dicho framework.

Además de tener muchas otras ventajas, Struts puede convertirse en algo bien complejo, lo que trae sus inconvenientes, algunos de los cuales son citados a continuación:

- Una gran curva de aprendizaje: es necesario tener amplio conocimiento acerca de las APIs de Java dedicadas a la web. Esta inconveniencia es específicamente significativa para proyectos pequeños con desarrolladores poco experimentados, en los que se puede emplear mucho más tiempo estudiando Struts que desarrollando el proyecto.
- Enfoque rígido: el lado opuesto del beneficio de que Struts tiene un apoyo consistente a la metodología del MVC es que Struts hace muy difícil (pero no imposible) al desarrollador utilizar otras metodologías.
- Poca transparencia: El código de las aplicaciones en Struts es difícil de comprender a simple vista y difícil de comparar y optimizar.

Además de esto, Struts no manipula eventos del lado del cliente, lo que provoca que tengan que ser implementados por el usuario con otro lenguaje.

Hasta hace algún tiempo existía poca documentación sobre Struts en Internet. Esto ya ha ido cambiando dado el tiempo de vida de dicho framework.

2.4.4 JavaServer Faces

JavaServer Faces, o simplemente JSF es el estándar presentado por Sun Microsystems para la capa de presentación web. Se rige, como una evolución natural, de los frameworks actuales hacia un sistema de componentes. Es un framework sencillo que aporta los componentes básicos de las páginas web además de permitir crear componentes más complejos, tales como menús, árboles, pestañas, etc. Hoy por hoy existe la disponibilidad de diferentes implementaciones de JSF, tanto comerciales como libres y de código abierto, así como librerías de componentes adicionales que amplían la funcionalidad del framework. La principal característica de JSF es la simplificación en el desarrollo de interfaces de usuario en J2EE.

Además de esto, JSF incluye:

- Un conjunto de APIs para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar la entrada de datos, definir un esquema de navegación de las páginas y dar soporte para internacionalización y accesibilidad.
- Un conjunto por defecto de componentes de interfaz de usuario.
- Dos librerías de etiquetas personalizadas para JavaServer Pages que permiten expresar una interfaz JavaSever Faces dentro de una página JSP.
- Un modelo de eventos del lado del servidor.
- Administración de estados.
- JavaBeans o Beans manipulados.
- Lenguaje unificado de expresiones, tanto para JavaServer Pages como para JavaServer Faces, las que al final compilan como JSPs.

La principal meta de JSF, como se ha mencionado, es la de agilizar el desarrollo de aplicaciones web a la vez que aumenta la productividad. Muchos son los que llaman a JSF como «el framework que ya estaba haciendo falta» ya que por vez primera ha sido sencillo crear complejas interfaces de usuario e integrarlas con fuentes de datos. Además de esto, existen muchos proyectos de software libre, como el proyecto MyFaces de la fundación Apache que ofrece una implementación abierta de JavaServer Faces, que, además, incluye un amplio conjunto de componentes adicionales. Oracle e IBM son otras dos compañías que, junto al creador Sun Microsystems, han optado por este framework como solución estándar. Como si esto fuese poco, la mayoría de los componentes de interfaz de usuario con que cuenta actualmente JSF han sido integrados con la novedosa tecnología AJAX. Es muy importante decir que el uso de librerías externas para JSF, como MyFaces, no impide el uso de otras librerías de componentes, así que su uso solo depende de la utilidad que ofrezcan a los programadores. JSF ha tomado muchas cosas de otros frameworks anteriores, como Struts, del que tomó la idea del controlador frontal, el sistema de validaciones, etc. En la siguiente imagen se muestra cómo funciona el patrón Modelo 2 del MVC en JSF.

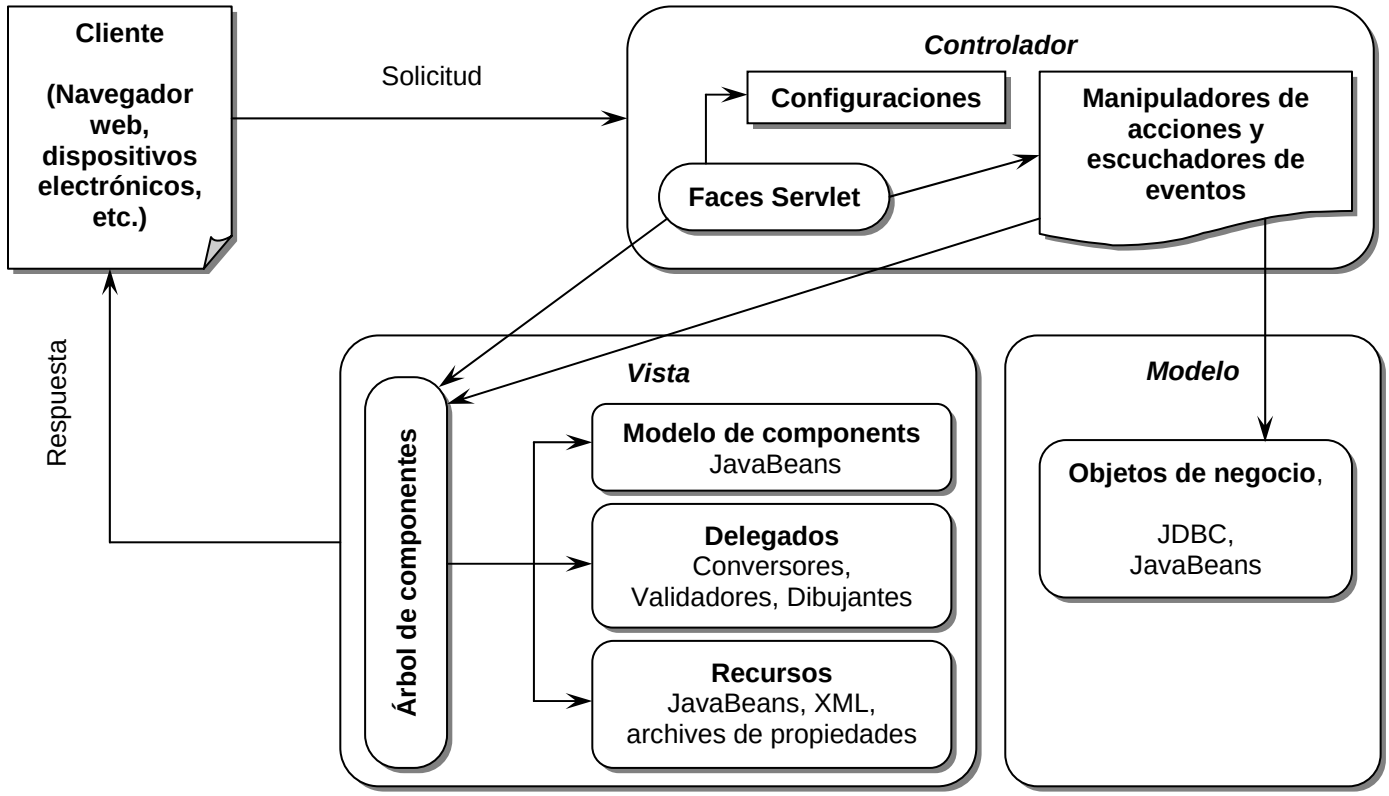

Figura 2.7. Implementación del Modelo 2 de JSF.

Resulta curioso que para algunos desarrolladores JSF tiende a ser tedioso, exponiendo que, desde la perspectiva del desarrollador es «demasiado JSP y XML». Entonces, ¿vale la pena JSF? No se puede ser negativo con algo desde sus comienzos. JSF es algo muy bien pensado, pero que es relativamente nuevo. Es un hecho que no se debe ser ciego, o hacer la vista gorda, en cuanto a las ventajas y limitaciones de una tecnología solo porque le guste a los desarrolladores, porque sea nueva o porque simplemente esté de moda. La principal ventaja es que se ha convertido en un estándar, que, como se ha expuesto anteriormente, es apoyado y soportado por las compañías más pujantes en el tema de las aplicaciones empresariales basadas en J2EE. A continuación se exponen algunas de las ventajas y desventajas de JSF.

Ventajas para el desarrollador:

- Básicamente el modelo de componentes de JSF hace muy simple el código.
- Puede usarse un lenguaje basado en expresiones en cualquier lugar y en cualquier momento.

- La habilidad para manipular el árbol de componentes de interfaz de usuario en Java antes de que la página se construya en el cliente hace el dinamismo de las páginas realmente simple.
- No se requiere de un entorno de desarrollo específico para construir páginas JSP basadas en la tecnología de JavaServer Pages, aunque el uso de un entorno de desarrollo apropiado simplificaría mucho las cosas.
- Manipulación de eventos del lado del servidor.

Desventajas:

- JSF es una tecnología muy nueva y aún no ha alcanzado el grado de madurez de otras tecnologías, lo que hace que los desarrolladores sientan recelos con respecto a ella.
- La documentación existente no es abundante y es bien difícil contactar personal capacitado en la tecnología.
- No tiene implementados mecanismos de seguridad.

A grandes rasgos, se puede establecer un cotejo entre Struts y JSF, así como las tecnologías más básicas en la gráfica del anexo 3. Haciendo una breve comparación entre JSF y Struts no es difícil concluir las ventajas y desventajas que existen entre ambos. Por un lado existen marcadas ventajas de JSF sobre Struts:

- Componentes habituales: JSF hace que sea relativamente sencillo combinar complejas interfaces gráficas de usuario en un componente simple y manejable. Struts no.
- Soporte para otras tecnologías de comunicación Máquina-Hombre: JSF no está limitado a HTML y HTTP, mientras que Struts sí.
- Acceso a los JavaBeans por su nombre: JSF permite asignarle nombre a los beans, entonces se puede hacer referencia a ellos por el nombre en los formularios web. Struts, por su parte, tiene un complejo sistema con varios niveles de oblicuidad a base de archivos XML en los que representa cual formulario es la entrada para una acción.
- Lenguaje de expresiones: el lenguaje de expresiones de JSF es más potente y conciso que las etiquetas <bean:write … /> de Struts.
- Controlador y definidor de JavaBeans simple: JSF no requiere que sus clases hereden de alguna clase en específico (por ejemplo Action) o usar algún método particular (por ejemplo execute o perform) y Struts sí.

- Archivo de configuración simple y estructura global: el archivo de configuración FACES-CONFIG.XML es mucho más fácil de usar y entender que el STRUTS-CONFIG.XML. En general, JSF es más simple.
- Potenciales herramientas de soporte más poderosas: la orientación alrededor de los controles de interfaces gráficas de usuario abren la posibilidad de uso simple en posibles entornos de desarrollo del tipo “arrastrar y soltar”. Struts no.

Pero no todo es sonrisas hacia JSF, Struts también tiene puntos por encima:

- Fuerte base y momento en la industria: sobre Struts programan una amplia cantidad de desarrolladores y cubre la mayor parte del momento entre programadores y directivos de la industria tecnológica. JSF no.
- En Struts el formulario y su manipulador tienen distintas URL, mientras que en JSF son idénticas. Muchos desarrolladores creen que esto entorpece el trabajo.
- Menor soporte en herramientas: Struts es soportado por una amplia gama de Entornos de Desarrollo. JSF no, todavía.
- Struts viene con poderosas facilidades para esquemas visuales. JSF no, pero esta funcionalidad puede ser extraída de Struts y utilizada por JSF, lo que conllevaría a la utilización de ambos frameworks en la capa de presentación.
- Solo método POST: JSF no soporta el método GET, de manera que resulta imposible marcar páginas resultantes.

Teniendo en mente lo anterior la balanza continúa inclinándose hacia JSF debido al incremento de usuarios, aunque algunas compañías continúan confiando más en Struts por ser un framework de mayor madurez. Es bien cierto que muchas empresas plantean que sería bueno y provechoso un framework que incluyera lo mejor de ambos, los vinculara en su interior o que al menos fuese el framework web definitivo para Java, pero la idea no ha fructificado, pues ambos frameworks continúan siendo desarrollados por separado, permitiendo, claro está, una integración entre ellos.

2.4.5 Spring

Desarrollado por Interface21[12], Spring es un conjunto de librerías ó módulos de entre los cuales se pueden escoger aquellos que faciliten la implementación de las aplicaciones. Es un framework libre y de código abierto. Entre sus posibilidades más potentes está su contenedor de Inversión del Control, o IoC, (Inversion of Control), también llamado Inyección de Dependencias, que es una técnica alternativa a las clásicas búsquedas de recursos y que consiste en que es el framework quien llama al código del desarrollador y no al contrario. Además, el contenedor inyecta dependencias dentro de las instancias utilizando métodos Java. Estas dependencias pueden ser objetos que colaboran entre sí o simples tipos de datos primitivos. Esto es más bien que los valores son empujados dentro de los objetos y no alados por los objetos, como tradicionalmente se hace.

Spring, además, permite configurar las clases en un archivo XML y definir en él las dependencias. De esta forma la aplicación se vuelve muy modular y a la vez no adquiere dependencias con Spring. Cuenta con plantillas de utilidades para Hibernate, Ibatis y JDBC, así como la integración con Struts, JSF y otros frameworks.

Es uno de los proyectos más sorprendentes en el panorama actual de Java, en el grado en que ayuda a que los diferentes componentes que forman una aplicación trabajen entre sí, pero no establece apenas dependencias consigo mismo. Esta es la primera característica de este framework. Sería posible retirarlo sin prácticamente cambiar líneas de código. Lo único que sería necesario es, lógicamente, añadir la funcionalidad que provee, ya sea con otro framework principal o con el código del propio desarrollador. A nivel de soporte de la comunidad, Spring es uno de los proyectos con mayor actividad, con desarrollos dentro y fuera del framework. De hecho, Acegi, el sistema de seguridad empleado por Spring, fue diseñado y realizado por colaboradores ajenos al framework.

Anteriormente se dijo que Spring era un conjunto de librerías o módulos. Pues bien, entrando un poco dentro de Spring se puede decir que es un framework compuesto por varios frameworks más pequeños, los cuales son:

- Contenedor de inversión de control: la configuración de los componentes de la aplicación y el ciclo de vida es manipulado básicamente por objetos Java comunes.

[12] **Interface21.** Es la empresa informática encargada del desarrollo del framework Spring.

- Programación orientada a aspectos: el trabajo con las funcionalidades que no pueden ser implementadas mediante la programación orientada a objetos sin hacer sacrificios de rendimiento y estabilidad.
- Framework de acceso a datos: trabaja con sistemas de administración objeto-relacionales que proveen soluciones para los desafíos técnicos reusables en multitudes de ambientes basados en Java.
- Manipulación de transacciones: armonización de varios APIs manipuladores de transacciones orquestados por objetos Java.
- Modelo-Vista-Controlador: framework basado en HTTP y Servlets que provee extensiones y personalizaciones.
- Framework de acceso remoto: soporte para aplicaciones basadas en el protocolo HTTP, tales como RMI, CORBA y Servicios Web o Web Services (SOAP).

Básicamente, la estructura de los módulos de Spring queda como sigue:

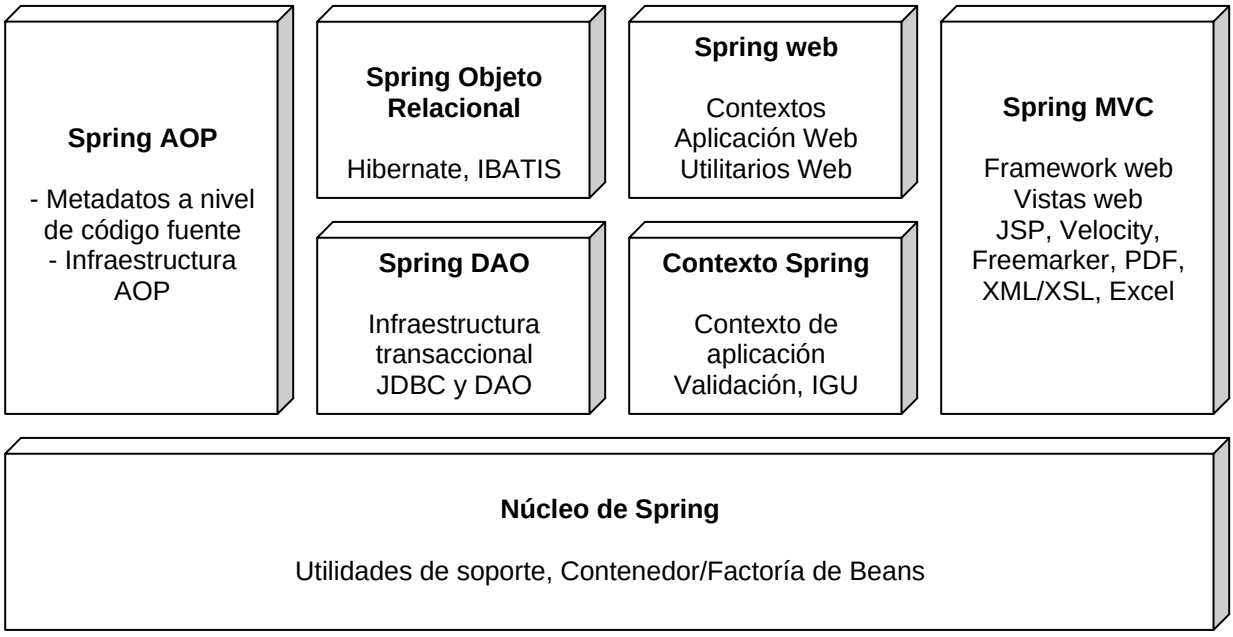

Figura 2.8. Módulos o sub-frameworks de Spring.

El núcleo contenedor provee las funcionalidades fundamentales de Spring. En este módulo se encuentra la factoría de Beans, lo que es el corazón de cualquier aplicación basada en Spring. Una factoría de

Beans es una implementación del patrón Factory que aplica inversión de control para separar la configuración de la aplicación y las especificaciones de dependencia del código de aplicación.

La factoría de Beans hace a Spring un contenedor, pero el módulo de contexto es lo que lo convierte en un framework. Este módulo extiende el concepto de factoría de Beans, agregando soporte para internacionalización, eventos del ciclo de vida y validaciones. En adición, suple algunos servicios empresariales tales como correo electrónico, acceso remoto, integración y programación de actividades. Además, incluye soporte para integración con frameworks de plantillas tales como Velocity y Freemarker.

Spring provee un amplio soporte para la programación orientada a aspectos, y para ello se encarga su módulo de AOP, el cual sirve como base para la implementación de los propios aspectos que los desarrolladores tengan que insertar en la aplicación. Para asegurar interoperabilidad entre Spring y otros frameworks con soporte para programación orientada a aspectos, el módulo AOP de Spring está basado en el API definido por la Alianza AOP, la cual es un proyecto de código abierto cuyo objetivo es promover la programación orientada a aspectos y la interoperabilidad entre las diferentes implementaciones orientadas a aspectos definiendo un conjunto común de interfaces y componentes. Asimismo, introduce la programación de meta datos para Spring. Con esto último se logra que, al agregar anotaciones al código fuente, Spring sea capaz de saber en qué lugar y como aplicar aspectos.

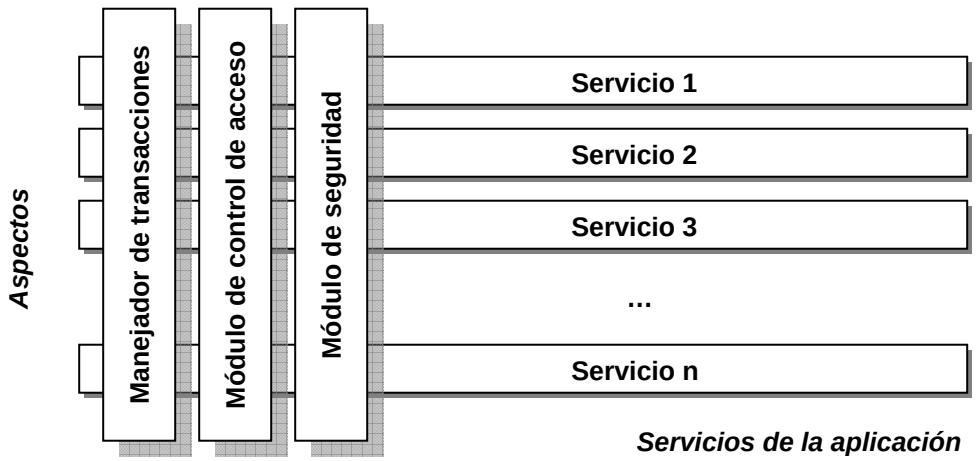

Figura 2.9. Modelo de la programación orientada a aspectos con Spring.

Al trabajar con JDBC (Java Database Connectivity) es común obtener un código textual que obtiene una conexión, crea una declaración, procesa el conjunto de resultados y entonces cierra la conexión. El

módulo de Spring para JDBC y DAO (Data Access Object) abstrae lo anteriormente dicho de manera que se pueda mantener el código de comunicación con las bases de datos simple y limpio, lo que previene de problemas que puedan resultar al cerrar los recursos de las bases de datos. Igualmente, este módulo utiliza el módulo AOP de Spring para proveer servicios de manipulación de transacciones para objetos en una aplicación Spring.

Para aquellos que prefieren utilizar el mapeo objeto relacional (ORM) sobre el JDBC directo, Spring provee un módulo objeto relacional. Por su parte, Spring no fue diseñado su propia solución objeto relacional, pero provee una extensa gama de posibilidades de integración con la mayoría de los frameworks que implementan el modelo objeto relacional, tales como Hibernate, JDO e IBATIS. Los manipuladores de transacciones de Spring proveen soporte para cada uno de los frameworks objeto relacional mencionados, así como soporte para JDBC.

El módulo de contexto web se fundamenta en el módulo de contexto de aplicación, proveyendo un contexto apropiado para las aplicaciones basadas en la web. En adición, este módulo contiene soporte para muchas tareas orientadas hacia la web, como manipulación transparente de solicitudes de subida de archivos y las ataduras de parámetros de solicitudes de manera programática hacia los objetos de negocio. Además contiene soporte de integración con Struts.

Spring tiene un bien formado y completo sub-framework Modelo-Vista-Controlador para la creación de aplicaciones web. Aunque Spring puede ser fácilmente integrado con otros frameworks basados en MVC tales como Struts y JSF, el módulo MVC de Spring utiliza mecanismos de inversión de control para proveer una separación limpia del controlador de lógica y los objetos de negocio. Esto además permite enlazar de manera declarativa los parámetros de solicitud y los objetos de negocio. Lo que es más, el módulo MVC de Spring puede tomar ventaja de cualquier otro de los servicios de Spring, ya sean internacionalización, validación, etc.

2.4.6 Acegi

El mencionado Acegi es un gestor de seguridad que está diseñado fundamentalmente para ser utilizado con Spring y en el que destaca su versatilidad. Acegi proporciona una capa que envuelve diversos estándares de seguridad presentes en Java y ofrece una forma unificada de configuración a través de un descriptor en XML. Cubre la capa web y la de negocio. A nivel web captura todas las peticiones mediante

la implementación de un filtro y a nivel de métodos mediante interceptación a través de programación orientada a aspectos. En ambos casos permite aplicar los criterios de seguridad que trae predefinidos o añadir nuevas opciones de forma sencilla implementando los interfaces diseñados para tal fin.

Acegi ha sido elegido como opción frente a los sistemas propietarios de los diferentes vendedores por su universalidad de uso, ya que no es necesario cambiar nada si se cambia de proveedor en los servidores, así como por su potencia, la cual engloba a todos los APIs de seguridad de Java. Acegi es muy cotizado debido a que, además de ser de código abierto y libre, ofrece una seguridad no invasiva. Esto es debido a que puede ser declarada la seguridad sin hacer cambios dentro del código fuente de la aplicación web como parte de la programación orientada a aspectos.

2.4.7 Hibernate

Usar JDBC es complejo y muy dependiente de la estructura de los datos. Sería más natural y mucho más sencillo trabajar directamente con objetos, pero es imposible con las bases de datos relacionales, y las bases de datos orientadas a objeto están todavía muy verdes. La mejor opción entonces consiste en utilizar un motor de persistencia, que es el componente software encargado de ser el traductor entre objetos y registros. Un motor de persistencia de código abierto es Hibernate, que permite manipular el acceso a datos desde el punto de vista de la programación orientada a objetos. Por tanto, Hibernate es un motor de persistencia objeto relacional. Hibernate usa el mecanismo de reflexión de Java, que permite a un objeto en ejecución examinarse y manipularse a sí mismo. Cuenta con una amplia documentación, tanto a nivel de libros publicados, como disponible gratuitamente en la web.

Las principales características de Hibernate se pueden resumir como sigue:

- Ocultamiento de objetos: la sesión a nivel de transacción que oculta los objetos persistentes.
- La ejecución de sentencias se realiza solo cuando son necesarias, de manera que si ocurre una excepción y es necesario abortar una transacción muchas declaraciones nunca habrían sido ejecutadas. Además, esto hace que los tiempos de acceso a las bases de datos sea el mínimo posible.
- La no actualización de objetos no modificados. Es muy común ver, en aplicaciones que utilizan código a mano JDBC, los estados persistentes de los objetos actualizados, por ejemplo, si se presiona el botón “guardar” pero no se han hecho modificaciones. Hibernate es capaz de saber en

todo momento si el estado de un objeto ha cambiado, quitando, de esta manera viajes innecesarios al servidor de bases de datos.

- En consecuencia con el anterior punto, Hibernate solo manipulará las colecciones de datos si estas realmente han cambiado.
- Hibernate puede enrollar dos actualizaciones aparentemente distintas en una del mismo objeto en la misma declaración de actualización.
- Hibernate implementa un muy eficiente algoritmo de búsqueda.
- Inicialización de colecciones perezosas.
- Inicialización de objetos perezosos.

Hibernate puede ser utilizado tanto en la web como en aplicaciones convencionales. Esto no solo brinda la inversión de tiempo en Hibernate de cara al futuro, sino que, de ser útil, hace a los desarrolladores totalmente independientes del mismo.

El siguiente es un diagrama muestra, a gran escala, la arquitectura base de Hibernate:

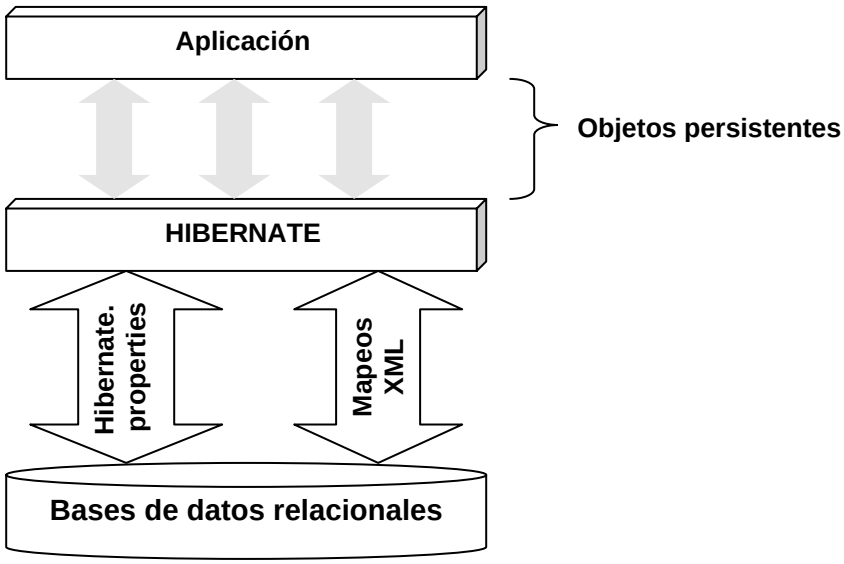

Figura 2.10. Arquitectura de Hibernate.

Hibernate se configura a partir de ficheros XML y ficheros properties o archivos de propiedades. Estos ficheros son el HIBERNATE.PROPERTIES y el HIBERNATE.CFG.XML. En estos archivos se indican los

parámetros de conexión a la base de datos como el nombre del equipo en que se encuentra la base de datos, la propia base de datos, nombre de usuario y contraseña, etc. Un parámetro interesante lo constituye el Dialecto de Hibernate. En este parámetro se indica el nombre de la clase que se encargará de comunicarse con la base de datos a través del lenguaje estructurado de consultas que sea capaz de comprender la base de datos. Es decir, cambiando la configuración de estos archivos es posible cambiar de base de datos, de tipo de base de datos, cambiar el nombre del equipo en que está ubicada la base de datos, etc. Todo esto sin necesidad de hacer cambios en la aplicación, de manera que migrar de una base de datos a otra constituye una labor que solo invertirá unos pocos minutos, mientras que si se usara JDBC se haría necesario incluso recompilar el proyecto con la consiguiente inversión de tiempo y posibles errores.

2.5 Herramientas para el desarrollo de aplicaciones J2EE

Para el desarrollo de aplicaciones J2EE se realizó un estudio de las posibles herramientas a utilizar. Teniéndose en cuenta la tendencia actual y las novedades de cada una de ellas.

2.5.1 Servidor de aplicaciones: Tomcat

Fueron valorados varios servidores de aplicaciones, entre ellos JBoss, Sun Application Server y Websphere Application Server. Todos ellos son servidores de aplicaciones enfocados, principalmente hacia el uso de EJBs, los que no están concebidos en la arquitectura y utilizarlos solo conllevaría a malgastar recursos del equipo sobre el que debería funcionar el servidor. Además de esto, los dos últimos pertenecen a compañías que cobran bien caro por el uso de los mismos; entonces, la respuesta consistía en utilizar Tomcat. Tomcat es un servidor web con soporte de servlets y JSPs. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache. Tomcat puede funcionar como servidor web por sí mismo. En sus inicios existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

2.5.2 IDE: Eclipse

Además del Eclipse se estudió la posibilidad de emplear NetBeans como entorno de desarrollo. NetBeans, a pesar de ser de código abierto y libre no tiene las facilidades brindadas por Eclipse para integrar elementos que faciliten el desarrollo de aplicaciones web. Por eso la elección fue Eclipse, debido a que es una plataforma de software de Código abierto independiente de una plataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar un Entorno integrado de desarrollo (del Inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se embarca como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones, como es el cliente BitTorrent Azureus. Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

2.5.3 Generador de reportes: iReport

Se analizaron algunos gestores de reportes como el BIRT (Business Intelligence and Reporting Tools) de la Eclipse Foundation, JFreeReports y Crystal Reports. Las dos primeras fueron rechazas por escasez de documentación y por ser productos muy inmaduros, mientras que la última es muy eficiente pero no es libre y las licencias actuales son sumamente caras para el uso de dicho software. Era necesario un generador de reportes libre, de código abierto, con un tiempo de explotación relativamente alto y con suficiente documentación. La solución más viable era la unión del JasperReports con el iReport. La herramienta iReport es un constructor/diseñador de informes visual, poderoso, intuitivo y fácil de usar para JasperReports escrito en Java. Este instrumento permite que los usuarios corrijan visualmente informes complejos con cartas, imágenes, subinformes. iReport está además integrado con JFreeChart, una de la biblioteca gráficas OpenSource más difundida para Java. Los datos para imprimir pueden ser recuperados por varios caminos incluso múltiples uniones JDBC, TableModels, JavaBeans, XML.

2.6 Sistemas de ejemplo desarrollados sobre arquitecturas J2EE

Fue necesario efectuar un estudio del estado del arte de las aplicaciones empresariales basadas en arquitecturas J2EE que emplearan las tendencias actuales de soluciones libres y de código abierto. Este estudio influyó notablemente en la elección de los componentes para la creación de la arquitectura del módulo de Servicio Autónomo del Sistema SAREN. A continuación se muestran algunos sistemas, los cuales presentan variantes arquitectónicas similares a la que se propone en el presente trabajo:

2.6.1 Compiere

El proyecto Compiere (http://www.compiere.org) ha desarrollado un sistema de gestión, llevado a cabo por la empresa ComPiere, Inc. Su modelo de negocio se basa en la liberación del código mientras se cobran los servicios de soporte que ofrecen mediante una red de asociados.

Sus principales características son, entre otras: amplia funcionalidad, número uno en descargas, el soporte no es gratuito, presenta una complejidad enorme, tanto como de cara al usuario como a los desarrolladores y proporciona clientes web como standalone. Esto requiere Oracle como sistema gestor de base de datos, utiliza procedimientos almacenados en la base de datos, con lo que su migración a otros sistemas es prácticamente inviable y utiliza EJB sobre el servidor de aplicaciones JBoss para la lógica de negocio.

2.6.2 Open for Business OFBiz

El proyecto Open for Business (http://www.ofbiz.org) es desarrollado principalmente por David E. Jones y Andy Zeneski. Presenta las siguientes características: una gran funcionalidad que abarca multitud de ámbitos de negocio, incluye facilidades para efectuar comercio electrónico, no es independiente del sistema gestor de base de datos, pero soporta los principales gestores de código abierto: MaxDB, PostgreSQL e Hypersonic SQL, es un proyecto maduro comenzado a mediados de 2001, tiene una interfaz de usuario no demasiado complicado con un diseño uniforme. Esto utiliza una gran cantidad de subproyectos de código abierto, emplea un amplio conjunto de estándares, utiliza un motor de persistencia realizado específicamente para el proyecto basado en metadatos en XML, aunque en sus planes está migrar a Hibernate en próximas versiones, utiliza gran cantidad de metadatos, lo que aunque aumenta su

flexibilidad hace que sea realmente complejo y su núcleo central no ha variado desde sus inicios, no ha aprovechado el gran número de soluciones que han surgido desde entonces.

2.6.3 JCatalog

Es un sistema de catálogo de productos online, la implementación de la misma ha sido descrita en forma de artículo por Derek Yang Shen, y el objetivo principal de este es mostrar como integrar los frameworks JSF, Spring, e Hibernate, además que discute las ventajas y desventajas de las tecnologías utilizadas en JCatalog y demuestra aproximaciones para diseñar algunos de los aspectos clave de la aplicación. La arquitectura empleada en esta aplicación es una arquitectura multi-capa no distribuida, en la cual el framework JSF se utiliza para implementar la capa de presentación, el framework Spring es utilizado en la capa de lógica-de-negocio para manejar los objetos de negocio, y proporcionar control de transacciones de forma declarativa y control de recursos, el framework de mapeo objeto ralacional (O/R) Hibernate se utiliza en la capa de integración y es donde se implementa de forma muy cómoda y eficiente el patrón de diseño DAO. A continuación se muestra un diagrama que ilustra dicha variante arquitectónica:

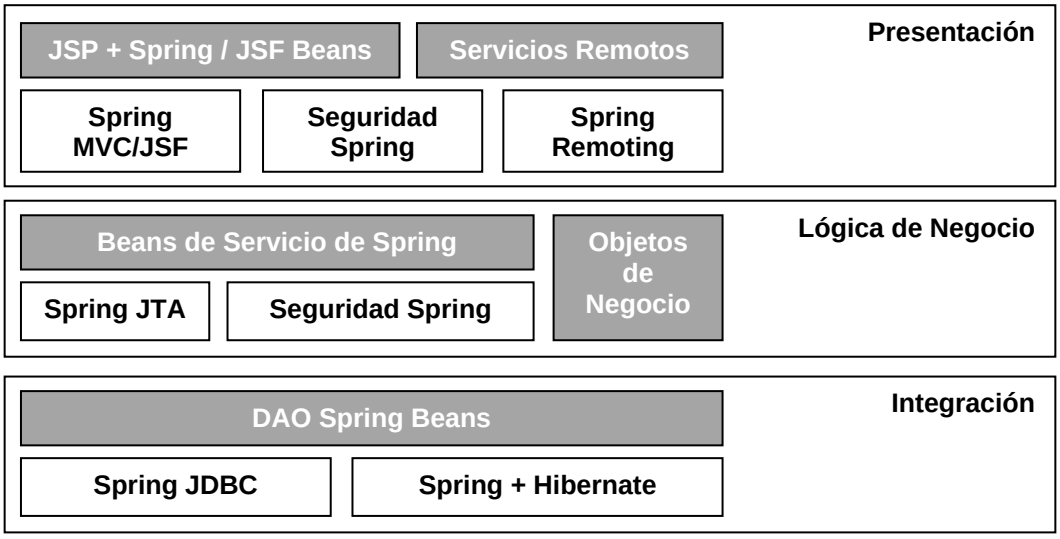

Figura 2.11. Arquitectura JSF, Spring, Hibernate.

2.6.4 Oness

Es un sistema de gestión de una empresa dedicada a la venta al mayor dentro del comercio textil. El desarrollo de dicho sistema fue el proyecto de fin de carrera de Ingeniería Informática del compañero Carlos Sánchez González en la Universidad de Coruña. En este trabajo se muestra como integrar los frameworks Struts, Spring, e Hibernate, además que discute las ventajas y desventajas de las tecnologías utilizadas y demuestra aproximaciones para diseñar algunos de los aspectos clave de la aplicación. La arquitectura empleada en esta aplicación es una arquitectura multi-capa no distribuida, en la cual el framework Struts se utiliza para implementar la capa de presentación, el framework Spring es utilizado en la capa de lógica-de-negocio para manejar los objetos de negocio, y proporcionar control de transacciones de forma declarativa y control de recursos, el framework de mapeo objeto ralacional (O/R) Hibernate se utiliza en la capa de integración y es donde se implementa de forma muy cómoda y eficiente el patrón de diseño DAO. A continuación se muestra un diagrama que ilustra dicha variante arquitectónica:

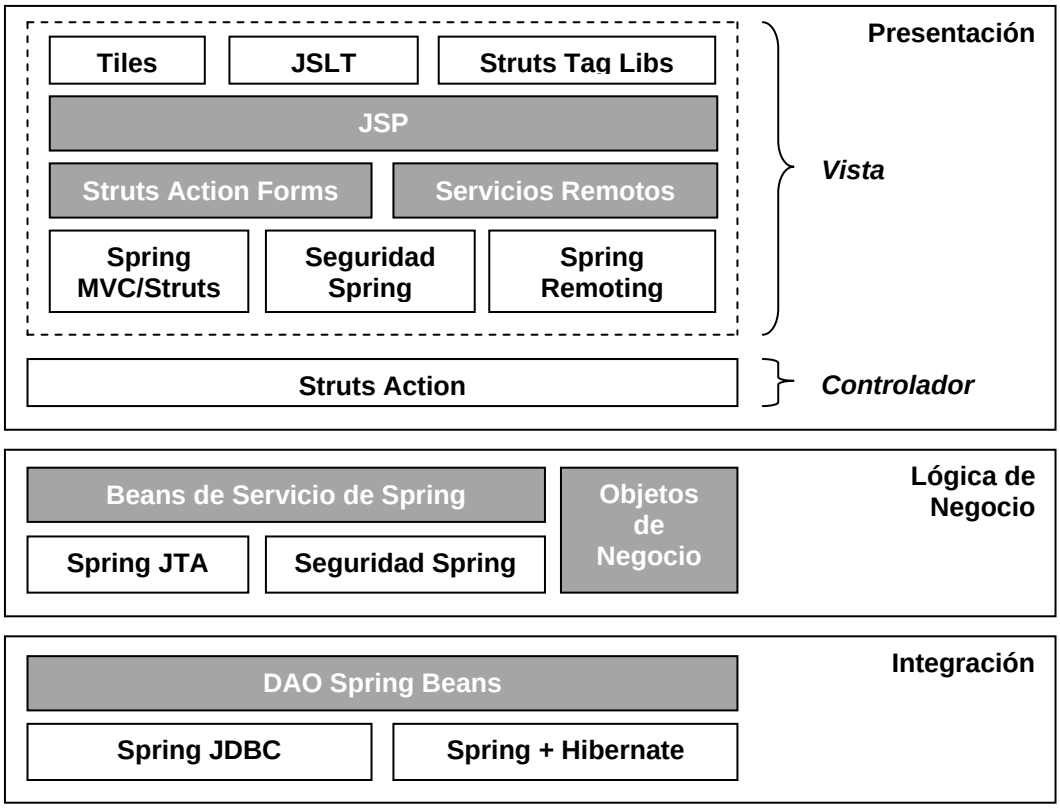

Figura 2.12. Arquitectura Struts, Spring, Hibernate.

2.7 Conclusiones

A manera de conclusión, se puede plantear que J2EE es una plataforma empresarial ideal para el desarrollo de software libre. La cantidad y riqueza de las soluciones libres que se pueden utilizar para crear arquitecturas empresariales dentro de la plataforma J2EE ha quedado de manifiesto en el estudio que se ha hecho para la realización de este trabajo de diploma. No sólo son muchos los productos sino que además son productos de calidad contrastada y que están siendo adoptados por numerosas soluciones comerciales.

En el estudio que se hizo de sistemas similares se confirma que utilizando J2EE es posible crear una infraestructura empresarial únicamente con productos libres. Además, esta arquitectura tiene todos los requisitos que se le pedían a cualquier plataforma empresarial, escalabilidad, fiabilidad, disponibilidad, extensibilidad, y se ha conseguido utilizando productos libres.

Se está abriendo un mercado muy amplio y hay grandes oportunidades. La solución que se pretende dar con el presente trabajo será una razón más para apostar por J2EE como la plataforma ideal para el desarrollo de aplicaciones empresariales. Esta solución no tiene grandes costes de licencias por lo que el margen de beneficios que ofrece es muy grande. Queda claramente demostrado que las arquitecturas basadas en software libre son las que ofrecen la mejor relación calidad/precio.

CAPÍTULO 3: ARQUITECTURA J2EE APLICADA EN EL MÓDULO SERVICIO AUTÓNOMO

Las dimensiones de los sistemas software actuales son extremadamente grandes y normalmente el tiempo requerido para la entrega de los mismos no es mucho. Antaño una sola persona, o un equipo de personas bien reducido en número, era capaz de crear un programa para dar la solución informática necesaria. Hasta hace poco, se asumía que la usabilidad era una propiedad exclusiva de la presentación de la información. Comúnmente se creía que, encapsulando la capa de presentación y separándola del resto, se podía desarrollar una aplicación. En realidad este modelo ha fallado muy a menudo, por lo que es muy importante ir más lejos y no solo basta tener en cuenta la presentación y la funcionalidad, sobre todo en sistemas complejos, como pueden ser los entornos distribuidos, los transaccionales y aquéllos en los que puede haber miles de usuarios conectados al mismo tiempo. Se hace muy sustancial tener en cuenta la usabilidad desde el mismo inicio del diseño del sistema, es decir, desde lo que se denomina momento de la Arquitectura de Software. Es bien sabido por parte de los ingenieros de software que mientras más tarde se detectan los problemas más cuesta corregirlos. Muchas veces, al diseñar una interfaz de usuario, se desean crear interacciones y diálogos que el ambiente tecnológico no lo permite, a lo que la respuesta de los técnicos no se hace esperar y, en medio de un vocabulario lleno de siglas y conceptos técnicos, plantean que en su plataforma eso no es posible. Son cosas como que el usuario pueda visualizar el progreso de sus peticiones, que pueda disponer de entornos en varios idiomas, cancelar una operación que lleva mucho tiempo en espera, utilizar y reutilizar información entrada previamente y muchas otras cosas. Si se analizan estos entornos de interacción, se llega a la conclusión de que la causa principal por la que no se puedan implementar es debido a que no se tuvo en cuenta al usuario en el inicio del diseño del sistema, es decir, en la Arquitectura de Software. En otras palabras, se tomó una arquitectura de software arbitraria que en ningún momento facilitaba el trabajo de los desarrolladores, o que al menos brindaba una fuerte base, y por ende la aplicación tendría un desempeño y pobre performance, rayando sobre lo mediocre, para llamarlo de alguna manera.

El mundo del desarrollo de software se encuentra en constante movimiento y desarrollo. Parafraseando a RUP, y tomando sus ideas centrales sobre el proceso de desarrollo de software, es atinado apuntar que el entorno informático y de las tecnologías de la información hace tiempo que se ha venido a convertirse en un proceso iterativo e incremental, ya que cada día cambian y aparecen nuevas versiones de todos aquellos programas y tecnologías a los que los desarrolladores están habituados y que generalmente

utilizan para trabajar. Es por ello que es bien común encontrar ligeros cambios, e incluso grandes cambios en algunas oportunidades, en la arquitectura de un proyecto software cuando no se tiene un pleno conocimiento acerca de los componentes de la misma. En múltiples ocasiones estos cambios están dados por actualizaciones y mejoras del momento de software o por necesidades que vayan surgiendo a medida que se vayan refinando los requisitos tanto funcionales como no funcionales del negocio que pretenda automatizar el proyecto de software en cuestión. Sin lugar a dudas resulta imprescindible tener bien definidos todos los requerimientos antes de concertar una arquitectura de software, debido a todas las implicaciones, principalmente tiempo, que trae consigo un cambio en la arquitectura, pero también es muy cierto la existencia de momentos y proyectos en los que un cambio podría resultar beneficioso tanto para desarrolladores, y equipo del proyecto de software en general, como para usuarios finales.

3.1 Primera aproximación de la arquitectura de Servicio Autónomo

El módulo de Servicio Autónomo del sistema SAREN fue concebido desde sus inicios para ser ejecutado en un ambiente web a solicitud de los clientes, además por las ventajas que brindan las aplicaciones de este tipo, proveen una manera sencilla de interacción con poca o casi ninguna instalación de software adicional en el cliente, facilitando de esta forma el despliegue, mantenimiento y actualización de este tipo de sistemas. El auge de multitud de soluciones o frameworks open source hace que su desarrollo sea sencillo y que un gran número de desarrolladores tengan experiencia con ellos. A continuación se representa la forma en que se manejará la información en el sistema Servicio Autónomo, la cual consiste en un servidor web que contiene la aplicación de Servicio Autónomo que es accedida desde distintas estaciones de trabajo a través de un navegador web, la información mostrada y gestionada por el sistema se encuentra almacenada en un servidor de base de datos central denominado Centro de Datos, y este centro de dato se nutre constantemente de la información que es generada y procesada en la oficinas de los registros y notarías de la República Bolivariana de Venezuela.

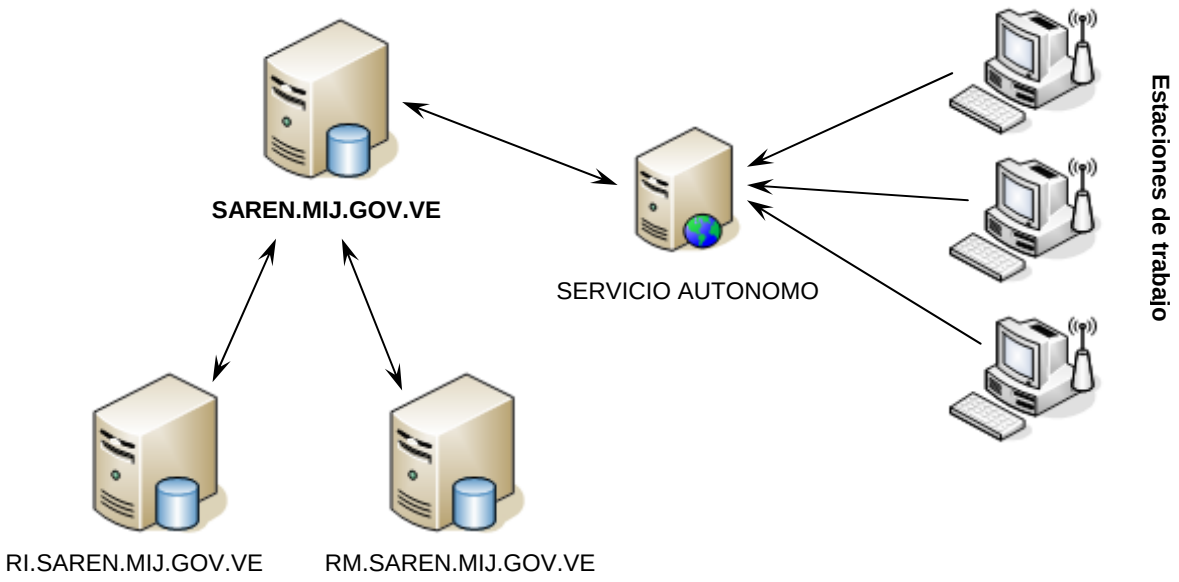

Figura 3.1. Esquema resumen de relaciones entre el Servicio Autónomo y las oficinas.

Por lo anteriormente dicho era necesaria la elección de una arquitectura a fin con los propósitos para los cuales fue pensado el Servicio Autónomo y que fuera capaz de brindar las funcionalidades requeridas en el tiempo convenido para su puesta en marcha. Entonces, teniendo en cuenta lo anterior es que se decide tomar como arquitectura basada en frameworks libres sobre la cual fundar dicho módulo una que tuviese a Struts en la capa de presentación, Hibernate en la capa de acceso a datos y en la capa de lógica del negocio estarían las clases controladoras de Struts y clases simples o JavaBeans de Java. La resolución de utilizar frameworks traería como resultado la implementación, por defecto, de muchos patrones de diseño, así como soluciones probadas a problemas comunes. Se elige al framework Struts como encargado de la lógica de presentación y la navegación teniendo en cuenta las características del mismo, descritas en el capítulo anterior, principalmente a que es uno de los frameworks web más utilizados en la creación de aplicaciones sobre J2EE y que mayor tiempo de explotación tiene, lo que evidencia una marcada madurez, y por ende es uno de los más documentados y que trae a sus espaldas una amplia gama de proyectos exitosos, además de que la experiencia de los desarrolladores en el tema Struts era mucho mayor que en otros frameworks destinados a la capa de presentación, como por ejemplo JSF. Hibernate, por su parte, a pesar de ser relativamente nuevo, se eligió para manipular el acceso a datos y la transaccionabilidad dado a que permite tratar los datos en un código aparte al gestor de bases de datos

empleado y que es muy trasparente y portable, lo que permite que la aplicación sea independiente, aparte del sistema operativo, del gestor de bases de datos, lo que resultaba de vital importancia hasta que se tuviera plena conciencia y conocimiento del gestor que finalmente fuera utilizado, teniendo en cuenta que la mayor parte de las empresas y compañías productoras de software de este tipo presentan dificultades a la hora de comerciar con Cuba esta especie de programas. Ese mismo código al cual se hacía referencia anteriormente permite el acceso a datos de manera orientada a objetos lo cual constituye otro punto a favor de incluir a Hibernate en la arquitectura. Además de esto, Hibernate es muy poderoso y seguro y está siendo ampliamente utilizado por la comunidad de desarrollo J2EE a nivel mundial, incluso grandes compañías como la propia Sun Microsystems e IBM consienten en que Hibernate es un framework muy potente y muy recomendado para ser utilizado. El siguiente es un esbozo de cómo quedaría esa arquitectura:

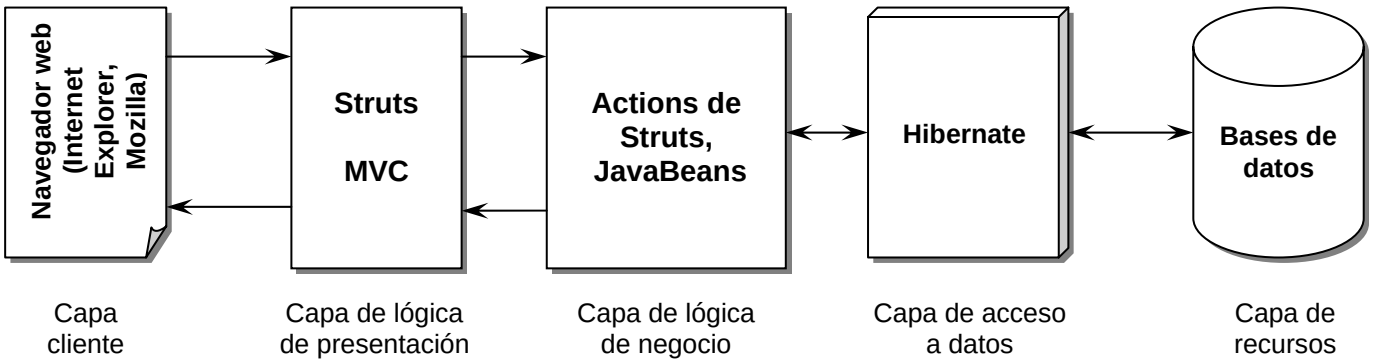

Figura 3.2. Primera aproximación de arquitectura de Servicio Autónomo.

El manejo de las transacciones hasta el momento se hacía utilizando Hibernate, este manejo requería ser programado de forma explícita, que es una forma invasiva que ensucia el código básico del sistema y lo vuelve complejo, además de que se repite el mismo código en varios componentes de la aplicación. También puede ser fuente de errores, sino se tiene un elevado conocimiento en cuanto a este manejo. Se necesita de un mecanismo donde este aspecto pueda ser manejado de forma transparente al código programado por los desarrolladores. Es por ello que se recurre a utilizar Spring ya que provee una completa integración con Hibernate en términos de soporte a implementaciones DAO y estrategias con transacciones. Especial soporte a Hibernate añadiendo convenientes características de inversión de control (IoC), y solucionando muchos de los comunes problemas de integración de Hibernate. Todo ello cumpliendo con las transacciones genéricas de Spring y la jerarquía de excepciones DAO. Funcionalidad

de programación orientada a aspectos (AOP), totalmente integrada en la gestión de configuración de Spring. Se puede aplicar AOP a cualquier objeto gestionado por Spring, añadiendo aspectos como gestión de transacciones declarativa. Con Spring se puede tener gestión de transacciones declarativa sin enterprise java beans (EJB), incluso sin java transaction API (JTA), si se utiliza una única base de datos en un contenedor web sin soporte JTA.

Además de lo anterior vale la pena mencionar algunos otros aspectos. Por ejemplo, la seguridad de la aplicación, el control de sesiones, los flujos de trabajo, los niveles de acceso y visibilidad, por solo citar algunos, estarían dados por clases simples de Java ó JavaBeans de una manera muy invasiva, dado a que era necesario que cada uno de estos tópicos estuviese implementado directamente en cada uno de los submódulos que comprenden el sistema. Esto, evidentemente atentaría en contra del tiempo de desarrollo y estaría jugando en contra de la mantenibilidad del sistema, teniendo en cuenta que aumentaría la complejidad del mismo.

Esta primera aproximación de arquitectura carecía de un mecanismo para el manejo de la seguridad de forma sencilla y eficiente, es por ello que se toma la decisión de utilizar el framework Acegi, que con sus características y prestaciones resuelve gran parte de dichos problemas. Acegi Security System for Spring (sistema de seguridad de Acegi para Spring) proporciona la funcionalidad necesaria para adoptar mecanismos de seguridad en aplicaciones Java utilizando características de programación orientada a aspectos, de forma transparente para el desarrollador, sin necesidad de desarrollar código, utilizando para ello el soporte prestado por el framework Spring, pero siendo posible utilizarlo en aplicaciones no desarrolladas con Spring. Acegi es uno de los mejores frameworks de seguridad existentes en Java, potente y flexible a la vez que sencillo de configurar, sin necesidad de modificar código ya existente y portable entre distintos contenedores de aplicaciones sin necesidad de cambios. Su integración con Spring hace que sea el recomendado para añadir funcionalidades de seguridad a las aplicaciones que utilizan ese magnífico framework, cuyo número crece día a día, si bien puede ser utilizado en cualquier tipo de aplicación sin ningún problema.

3.2 Arquitectura aprobada para Servicio Autónomo

El principal objetivo de esta arquitectura es separar, de la manera más limpia posible, las distintas capas de desarrollo, con especial atención a permitir un modelo de dominio limpio y a la facilidad de mantenimiento y evolución de las aplicaciones. También se tiene como un elemento importante la facilidad del despliegue. Spring es una de las novedades en el mundo de los frameworks para Java, introduciendo múltiples conceptos e ideas muy relevantes sin precedentes hasta el momento, lo que constituye algo verdaderamente importante para el mundo de la informática, aún para los más escépticos y detractores de la tecnología Java. Spring fue desarrollado en conjunto con uno de los padres de Struts, Rod Johnson, lo que hace que el framework, de antemano, tenga un alto nivel de confiabilidad. A pesar de ser algo relativamente nuevo y como se ha mencionado en el apartado anterior, Spring tomaría el control de la capa de negocios y sería el encargado de manejar la integración entre los otros dos frameworks de la aplicación. Esto podría parecer algo redundante o algo complejo y cabría hacerse la pregunta de si al emplear otro framework en la arquitectura ¿no complicaría las cosas? Regresando al concepto de framework de que es un conjunto de clases cooperando entre sí para facilitar el desarrollo de aplicaciones proveyendo una solución reutilizable, es muy sencillo llegar a la conclusión de que el uso de Spring no solamente facilitaría en gran medida la implementación del sistema, sino que además ofrecería una serie de características y funcionalidades que proporcionarían un alto grado de robustez y portabilidad a la arquitectura y por lo tanto a cualquier sistema software que fuese implementado basado en ella. Por otra parte y, sin olvidar que una aplicación actual que no se encuentre modularizada, o dividida en capas, está indudablemente condenada a tener un muy bajo rendimiento y todo lo que esto trae consigo y por ende el fracaso, la introducción de Spring permitiría una mayor modularización, lo que por consiguiente permitiría una mayor claridad entre las diferentes capas que conforman la arquitectura, de modo que se pudiera cambiar cualquiera de las capas por otra sin afectar el resto de las mismas. Volviendo al tema de los frameworks, es válido recordar que estos implementan por defecto muchos patrones de diseño; de hecho, los frameworks en algún momento han sido llamados “bibliotecas implícitas de patrones de diseño”, lo que daría a la arquitectura un mayor peso en cuestiones de rendimiento, usabilidad, etc. Además de todo lo expresado en este párrafo, es muy conveniente señalar algunas de las ventajas de Spring que resultarían de una gran utilidad en cualquier sistema informático que haya sido implementado con este framework. Primero está el asunto de Acegi, un framework que brinda una alta seguridad y control de roles y sesiones a las aplicaciones desarrolladas en Spring de manera no intrusiva, es decir, sin necesidad de intervenir en

el código fuente de la aplicación, sino que todo lo concerniente a los aspectos mencionados estaría de forma declarativa, o lo que es lo mismo, en archivos de configuración. Esta es una de las principales ventajas que trajo consigo la programación orientada a aspectos integrada con Spring. Esto último de la manera declarativa no intrusiva de manejar los entes tiene la gran ventaja de que no es necesario hacer cambios al código de la aplicación en caso de que se requiera hacer algún cambio en los niveles de seguridad, por solo citar un ejemplo. Y segundo los desarrolladores de Spring reconocieron que era prácticamente imposible desarrollar un framework objeto relacional que pudiese superar las prestaciones de Hibernate en el acceso a datos y por ello equiparon a Spring con módulos que permitiesen una fácil integración entre ambos frameworks, de manera que constituyeran una sólida unidad en ese aspecto, teniendo en mente que ambos frameworks continuarían operando en capas separadas, pero con una alta interrelación entre ellos. En este sentido muchos son los entendidos del tema que se han manifestado acerca de la gran lucidez de los creadores de Spring de reutilizar y unificar recursos que permiten realizar aplicaciones firmes y consistentes. En resumen, con la llegada de Spring no solamente se facilitaría el trabajo de integración entre los frameworks que componen la arquitectura y el proceso de desarrollo, sino que además implicaría sustanciales ahorros de tiempo en cuanto a requerimientos no funcionales a la par de que brindaría confort, seguridad, ligereza y una extensa lista de atributos que, en conjunto solo podrían traer beneficios a corto y a largo plazo. Es, pues, el siguiente boceto la manera en que se podría representar la nueva arquitectura luego de la inserción del framework Spring:

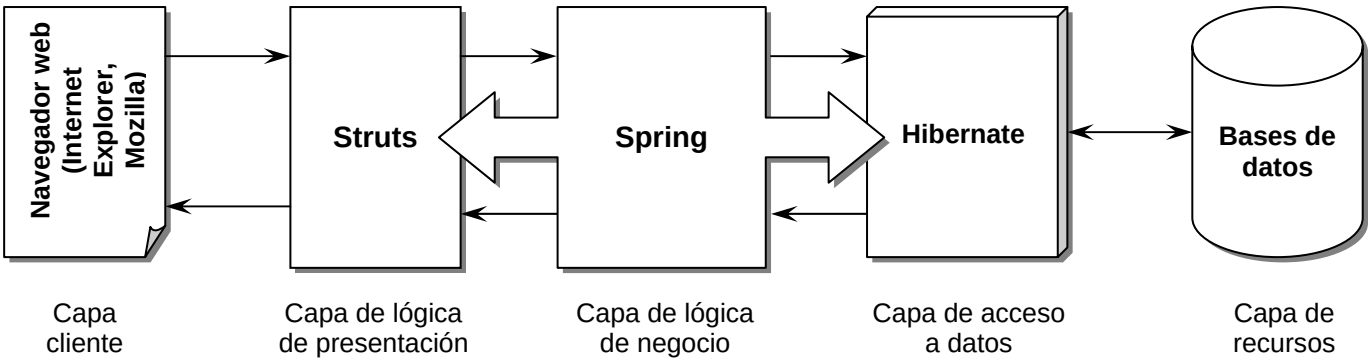

Figura 3.3. Arquitectura de Servicio Autónomo.

Resulta bien interesante y bien importante enfatizar en la importancia del desarrollo de aplicaciones de software basadas en arquitecturas de múltiples capas. Esto se evidenció en que se incrustó el framework

de Spring y sin embargo las restantes capas no sufrieron cambios, o en la práctica si existió algún cambio fue mínimo y he aquí otra de las principales características que se persigue en cualquier sistema informático, mayormente en los de gran escala, y es la escalabilidad. Lo expuesto con anterioridad en este mismo párrafo fue posible dado que, aunque hayan sido creados en momentos distintos y por diferentes empresas e instituciones, todos estos frameworks han sido diseñados e implementados sobre Java, lo que, a fin de cuentas, expresa que todo esto al final es Java y sin lugar a equivocaciones es una muestra más de los beneficios que puede traer para el desarrollo de un software el uso y la reutilización de código. Por otra parte, el mismo hecho de que al final todo esto es ciento por ciento Java brinda una total independencia de la plataforma en que se ejecuten las aplicaciones basadas en arquitecturas como esta, ofreciendo de esta manera una total portabilidad de la aplicación. Struts continuaría manipulando todo lo referente a la manipulación de interfaces, la navegación y parte del flujo de trabajo; mientras que Hibernate continuaría estando a cargo del acceso a datos.

3.3 Configuraciones necesarias

3.3.1 Configuración del Tomcat

Para el correcto funcionamiento de la aplicación es necesario efectuar varias configuraciones en el servidor web, en este caso Tomcat. Dichas configuraciones se realizan en el fichero SERVER.XML que se puede encontrar en la ruta \conf\server.xml en el directorio de instalación del propio Tomcat. Para activar el soporte HTTPS del Tomcat, que será utilizado por la aplicación para el transporte seguro de los datos es necesario añadir un certificado ssl en el directorio del servidor y hacer las respectivas modificaciones al fichero SERVER.XML. Es necesario partir de que se cuenta con una certificado SSL de nombre keystore. A continuación se muestran los elementos que se deben añadir en el SERVER.XML.

```
...
    <Connector acceptCount="100" connectionTimeout="20000" debug="0"
      disableUploadTimeout="true" enableLookups="false"
      maxSpareThreads="75" maxThreads="150" minSpareThreads="25"
      port="80" redirectPort="443"/>

    <Connector acceptCount="100" clientAuth="false" debug="0"
      disableUploadTimeout="true" enableLookups="false"
      keystoreFile="${catalina.home}/keystore"
      keystorePass="contrasenna" maxSpareThreads="75"
      maxThreads="150" minSpareThreads="25" port="443" scheme="https"
      secure="true" sslProtocol="TLS"/>
```

```
    <Connector debug="0" enableLookups="false" port="8009"
      protocol="AJP/1.3" redirectPort="443"/>
...
```

La etiqueta Connector representa un componente conector que le especifica al servidor web un conjunto de características acerca del comportamiento y funcionalidades que este tendrá disponible. Entre los elementos que se pueden especificar a estos conectores encontramos, acceptCount es el número máximo de conexiones que pueden ser procesadas por el servidor, connectionTimeout es el número máximo de milisegundos que este conector esperará para presentar una respuesta después de aceptar una conexión, debug representa el nivel de detalles de los mensajes generados por este componente en el proceso de debugging, disableUploadTimeout este permite que los servlets que son ejecutados en el servidor para hacer upload de datos puedan demorar un largo tiempo realizando dicha tarea, enableLookups se utiliza para obtener el nombre del cliente remoto que está realizando alguna petición al servidor, maxSpareThreads el máximo número de hilos que se podrán encontrar como reserva para utilizarlos en caso de que se realice una petición que necesite ser procesada, maxThreads es el número máximo de hilos que pueden ser creados por un conector y por tanto el máximo número de peticiones que pueden ser manejadas simultáneamente, minSpareThreads es el número de hilos que serán creados cuando un conector es iniciado, port es el puerto de la PC servidora que se utilizará para la funcionalidad que brinda el conector, redirectPort es el puerto de la PC servidora que se utilizará en caso que se requiera algún mecanismo de seguridad por ejemplo el uso de SSL, clientAuth es para cuando se utiliza el mecanismo SSL especificar si se requiere una certificado válido por parte del cliente remoto para entonces aceptar o rechazar la conexión con el mismo, keystoreFile es para especificar la ruta de el fichero que contiene el certificado que debe ser cargado por el servidor, keystorePass es la contraseña utilizada para acceder al certificado utilizado por el servidor que se especifica en el keystoreFile, scheme es para especificar el nombre del protocolo que se utiliza, secure se utiliza para especificar si serán validadas las peticiones haciendo uso de SSL, sslProtocol es para especificar la versión del protocolo SSL que se utiliza.

Tomcat permite manejar las reservas de conexiones (connections pool) posibilitando el uso eficiente de tecnologías como Hibernate. Usando Tomcat las fuentes de datos (DataSources) se localizan mediante JNDI (Java Naming and Directory Interface de sus siglas en inglés), y los atributos de los mismos se encuentran en el fichero de configuración del servidor (SERVER.XML). Véase ejemplo de código de configuración de las fuentes de datos en el Tomcat.

```
...
 <Context debug="0" docBase="${catalina.home}/aplicacion"
          path="/ServicioAutonomo" reloadable="true">
  <Resource auth="Container" name="jdbc/saren" type="javax.sql.DataSource"/>
  <ResourceParams name="jdbc/saren">
    <parameter>
      <name>url</name>
      <value>jdbc:oracle:thin:@10.7.10.100:1521:basedatos</value>
    </parameter>
    <parameter>
      <name>maxIdle</name>
      <value>10</value>
    </parameter>
    <parameter>
      <name>maxActive</name>
      <value>20</value>
    </parameter>
    <parameter>
      <name>driverClassName</name>
      <value>oracle.jdbc.driver.OracleDriver</value>
    </parameter>
    <parameter>
      <name>maxWait</name>
      <value>-1</value>
    </parameter>
    <parameter>
      <name>removeAbandoned</name>
      <value>true</value>
    </parameter>
    <parameter>
      <name>username</name>
      <value>nombreusuario</value>
    </parameter>
    <parameter>
      <name>factory</name>
      <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </parameter>
    <parameter>
      <name>logAbandoned</name>
      <value>true</value>
    </parameter>
    <parameter>
      <name>defaultAutoCommit</name>
      <value>false</value>
    </parameter>
    <parameter>
      <name>removeAbandonedTimeout</name>
      <value>60</value>
    </parameter>
    <parameter>
      <name>password</name>
      <value>contrasenna</value>
```

```
    </parameter>
  </ResourceParams>
 </Context>
...
```

La etiqueta Context junto a un conjunto de sub-etiquetas representa una aplicación web así como un conjunto de recursos que son utilizados por la misma. A continuación se explica el significado de cada uno de los atributos y elementos que componen dichas etiquetas, en caso de que no se muestra la explicación de algún elemento es porque ya ha sido explicado con anterioridad. El elemento docBase es para especificar la ruta del directorio donde está contenida la aplicación web.

3.3.2 Configuración de Struts

El framework Struts mantiene unidas varias tecnologías relacionadas de modo que los desarrolladores web pueden crear aplicaciones basadas en estándares de manera que son más sencillas de construir, extender y mantener. Es innegable el hecho de que Struts es el framework en que al menos primero piensan los desarrolladores tanto novatos como experimentados alrededor del mundo.

En el momento en que se diseñó y pensó la actual arquitectura del módulo de Servicio Autónomo del Sistema SAREN, después de estas y otras muchas consideraciones, se optó por emplear Struts en la capa de lógica de presentación, debido a que, en primer lugar y no es menos cierto, que la experiencia de los desarrolladores y arquitectos era mayor en este framework que en otros. Entonces, y ya entrando concretamente en el tema de este epígrafe ¿de qué manera se ajusta Struts? ¿De qué forma se ensambla?

El framework de Struts puede ser obtenido gratuitamente desde el mismo sitio del proyecto Apache a través de la dirección web http://jakarta.apache.org/struts/index.html. Además, es posible obtener los archivos fuentes de Struts en esa misma ubicación. En el capítulo de la fundamentación teórica se mencionaba que Struts está empaquetado de la misma manera que los creadores de Java proponen el empaquetamiento para las aplicaciones web J2EE tradicionales. Por otra parte, y como es bien sabido, los frameworks están diseñados para ser configurados mediante archivos XML de configuración y ficheros de propiedades. Las aplicaciones web en Java tienen un archivo de configuración llamado el WEB.XML, el cual es leído por el contenedor de la aplicación una vez que esta es lanzada y es el que va a portar toda la descripción y los recursos disponibles de que hace uso la aplicación, y precisamente en este archivo es en

el cual se le indica a la aplicación que se va a emplear Struts. Por otro lado, el propio framework, como ya fue mencionado en el apartado dedicado a Struts del capítulo anterior, también tiene un archivo de configuración, el llamado STRUTS-CONFIG.XML. Estos archivos de configuración son más que importantes desde el punto de vista de los conceptos arquitectónicos de escalabilidad y mantenibilidad, dado que proporcionan la facilidad de hacer cambios, tanto a la aplicación en general, como a una parte de la misma, en este caso Struts y ambos se encuentran localizados en la ruta \WebContent\WEB-INF\ de la aplicación, tal y como lo descrito por Sun Microsystems para la organización de aplicaciones web. En otras palabras, con esos ficheros de configuración se puede modificar el funcionamiento de la aplicación o simplemente de una fracción de esta sin tener que recurrir a otra cosa. Más adelante se mostrará como los otros frameworks empleados en la arquitectura hacen uso de este importante recurso que constituyen los archivos XML para la configuración de sus funcionalidades. Anteriormente se mencionó que es muy normal que los frameworks estén basados en el Modelo 2 de Sun Microsystems. Este Modelo 2 es básicamente un servlet. Es necesario mantener en mente la idea que Struts sería utilizado para la capa de presentación. En el siguiente fragmento se muestra cómo es que hace referencia al framework y al fichero de configuración del mismo:

```
<servlet>
  ...
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  <init-param>
   <param-name>config</param-name>
   <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
  ...
</servlet>
```

La etiqueta servlet, junto a un conjunto de sub-etiquetas, engloba la localización y definición de cierto y determinado Servlet utilizado en la aplicación. La sub-etiqueta servlet-name referencia al nombre de dicho Servlet, en este caso “action”, relacionado con el núcleo de Struts y servlet-class referencia a la clase del mismo. Por su parte init-param se utiliza para la inicialización de parámetros por defecto, en este caso param-name indica el nombre del parámetro y param-value su localización física. De esta manera, el contenedor que desplegará la aplicación sabrá que debe utilizar el framework de Struts. Además, es necesario indicarle las librerías de etiquetas (taglibs) que serán empleadas:

```
<taglib>
  <taglib-uri>/WEB-INF/struts-bean</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
 </taglib>
 <taglib>
  <taglib-uri>/WEB-INF/struts-logic</taglib-uri>
  <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
 </taglib>
 <taglib>
  <taglib-uri>/WEB-INF/struts-html</taglib-uri>
  <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
 </taglib>
<taglib>
```

Las etiquetas taglib encierran la definición de las bibliotecas de etiquetas y su localización. Para lo primero se utiliza la sub-etiqueta taglib-uri y para lo segundo taglib-location. Sin embargo esto no es suficiente. Es necesario además indicar en el archivo que contiene el CLASSPATH de la aplicación, ó camino de clases, en qué ubicación se encuentran las librerías del framework. Una vez hecho esto y ya está el framework insertado y listo para ser utilizado.

Como el resto de las soluciones libres que conforman la arquitectura, Struts juega un importante papel dentro de la misma. Struts actúa como el mediador entre el usuario y el sistema. Es decir, el contenido que se muestra al usuario descrito como HTML es dinámico y eso lo hace el framework. Es el encargado de manipular y procesar las solicitudes realizadas por el usuario en la capa de cliente y llevarlas al servidor (request) donde se procesa la lógica del negocio. De igual manera y una vez que el servidor tenga una respuesta para el usuario, es el encargado de tomar esa respuesta (response), y mostrársela al usuario convertida en HTML. En otras palabras y en el sentido más abstracto, Struts es la vía de comunicación entre el hombre y la máquina y entre la máquina y el hombre.

Además, Struts, dentro de la arquitectura, es el framework, o mejor dicho, el recurso que tiene a su cargo la navegación. Eso significa que Struts es quien dice de dónde viene y hacia dónde se va. En muchos de los llamados sitios web la navegación se realiza mediante código HTML, o lo que es lo mismo, se lleva a cabo en el lado del cliente con la consiguiente posibilidad de que puedan existir problemas de seguridad y para un sistema donde la seguridad es un factor vital esto no es ni siquiera pensado por los arquitectos y diseñadores, por el contrario, la navegación se realiza en el lado del servidor. Con esto se garantiza que al menos el sistema vaya a la página que tiene que ir en el momento en que debe hacerlo y nunca antes y

es precisamente una de las cosas en que interviene Struts. Por ejemplo, antes de poder utilizar la aplicación de Servicio Autónomo, el usuario debe ingresar su nombre de usuario y su contraseña; si la autenticación es satisfactoria la navegación irá hacia el interior del sistema, pero si por el contrario, la autenticación resulta fallida la navegación no permitirá ir hacia el interior del sistema, sino que regresará a la parte en que se solicita el nombre de usuario y la contraseña y además mostrará un mensaje de error explicando lo sucedido. Este es un ejemplo muy sencillo acerca de cómo funciona la navegación, pero existen muchos flujos de trabajo en los que se requiere que el sistema vaya de una página a otra e intercambie entre distintos lugares en los que esto se convierte en algo muy importante.

Como se ha mencionado anteriormente, Struts tiene su propio archivo de configuración, el STRUTS-CONFIG.XML, y es exactamente en este archivo donde se le indica al framework lo referente a la navegación. En este sentido es bueno aclarar que no importa en qué lugar se esté siempre se puede ir a varios lugares, según sea la navegación definida. Esto está dado, evidentemente, por la solicitud del usuario en ese momento y la respuesta que sea capaz de dar el sistema. El siguiente es un fragmento del archivo STRUTS-CONFIG.XML en el que se muestra este proceso:

```
<action-mappings>
  ...
  <action input="/jsp/AdminUsuarios/CambiarContrasenna.jsp"
       name="contrasenna" parameter="method" path="/cambiarContrasenna"
       scope="session" type="CambiarContrasennaAction" validate="true">
     <forward name="satisfactorio" path="/jsp/Plantilla/Inicio.jsp"/>
     <forward name="fallido" path="/jsp/AdminUsuarios/CambiarContrasenna.jsp"/>
  </action>
  ...
</action-mappings>
```

En este caso las etiquetas action-mappings enclaustran las acciones, que indican a dónde ir y cuándo. Para eso se usan las sub-etiquetas de acción (action), en cuya definición se especifica, entre otras cosas, en el input la página desde donde se accede a la acción, el name o nombre de referencia a la clase formulario, el parameter o nombre del parámetro utilizado para el transporte de información adicional, el path o el camino que se especifica en el formulario de la página, el scope o alcance ya sea de sesión (session) o de solicitud (request), el type o la clase Action en sí y el validate o validador para si es necesario validar (true) ó no (false) el formulario. Dentro de la etiqueta action se encuentran las etiquetas forward en las que se especifica en el path a qué página ir en dependencia del valor retornado por el

action en el name. Aquí se está en la parte que permite el cambio de contraseña y si este proceso devuelve un estado satisfactorio la navegación se mueve hacia el camino /jsp/AdminUsuarios/Inicio.jsp. Si por el contrario, el proceso devuelve un estado fallido se mueve hacia esa misma página /jsp/AdimUsuarios/CambiarContrasenna.jsp con el correspondiente mensaje de error.

Muy relacionado con la parte de manipulación de solicitudes y redibujado en código HTML las respuestas está todo lo referente a los formularios. Toda la información dentro de campos de texto, casillas de verificación y en cualquier componente web que se manipula en el lado del cliente está primeramente dentro de formularios. Struts enlaza los formularios web con clases formulario (ActionForm) en el lado del servidor, los que a su vez se enlazan con clases acción (Action), lo que le da su funcionamiento. Estas clases son simples clases Java ó JavaBeans que extienden las funcionalidades de las clases de Struts ActionForm y Action y que por tanto se les llama de igual manera, y que además implementan métodos especiales. Es muy necesario contemplar al framework como un todo único, debido a que es en las clases Action donde se procesan las solicitudes del usuario y se invocan los métodos de la capa de lógica de negocios y en consecuencia de las respuestas dadas por estos es que se retorna el estado que guiará al framework cuando vaya a ejecutar la navegación. Ambos tipos de clases se definen en el mismo archivo STRUTS-CONFIG.XML. El fragmento anterior de este archivo muestra la manera de definirle a Struts que en la página /jsp/AdminUsuarios/CambiarContrasenna.jsp existe un formulario llamado /cambiarContrasenna que se relaciona con la clase Action CambiarContrasennaAction y esta a su vez se encuentra relacionada con la clase formulario contrasenna definida en otro lugar del mismo archivo STRUTS-CONFIG.XML. Esta última definición a la que se hace alusión sería como sigue:

```
<form-beans>
  ...
  <form-bean name="contrasenna" type="CambiarContrasennaForm"/>
  ...
</form-beans>
```

Los form-beans del STRUTS-CONFIG.XML son las etiquetas encargadas de encapsular las declaraciones de las clases formulario que respaldan los elementos de los formularios en las páginas web y que a la vez están relacionadas con las clases action. Para lograr esto se utilizan las etiquetas form-bean, dentro de las cuales se define el name o nombre que recibirá en el XML la clase del formulario definida en type. Así,

pues, se le indica a Struts que la clase CambiarContrasennaForm será utilizada en la definición de una clase Action como contrasenna.

Luego de lo anteriormente dicho es muy sencillo llegar a la conclusión de que Struts juega un importante papel dentro de la arquitectura. Igualmente no es difícil llegar a la conclusión el acertado uso de los archivos de configuración XML en los frameworks, en este caso el archivo STRUTS-CONFIG.XML, el cual se puede incluso pensar como el mismo corazón de Struts, ya que en él se define todo el comportamiento del framework y por ende de la aplicación.

3.3.3 Configuración de Spring

En correspondencia con lo que se dice en el propio sitio web de Spring (http://www.springframework.org), «Spring es un framework para J2EE distribuido en capas basado en el código publicado por Rod Johnson en Expert One-on-One J2EE Design and Development (http://www.wrox.com/books/1861007841.htm)». Según su núcleo, provee los medios para manejar los objetos de negocio y sus dependencias. Por ejemplo, mediante la inversión del control (IoC) permite especificar los objetos de acceso a datos (DAO) que dependan de una fuente de datos (DataSource) en específico. Permite a los desarrolladores, además, realizar un código a base de interfaces y definitivamente simplificar el proceso de implementación de código fuente con el uso de ficheros XML. Spring contiene muchas clases que soportan otros frameworks, como Struts e Hibernate, para facilitar los procesos de integración.

Seguir los patrones de diseño J2EE puede ser en ocasiones engorroso e innecesario y de hecho, en muchas oportunidades se convierten en antipatrones. Spring sigue los patrones de diseño, pero todo mucho más simplificado. Por ejemplo, en lugar de implementar el patrón ServiceLocator para buscar las sesiones de Hibernate, se puede configurar un SessionFactory en Spring. Esto permite seguir las mejores prácticas de los expertos en J2EE en lugar de tener que estar tratando de entender el último patrón. Si se siguen los foros en línea, tales como el de TheServerSide.com ó el de JavaLobby.org, es muy presumible encontrar que Spring es muchas veces mencionado. Esto incluso tiene mayor tracción en la comunidad de blogs de Java en donde muchos desarrolladores describen sus experiencias y alaban su facilidad de uso.

¿Por qué Spring? Después de todo el paisaje de Java está repleto de frameworks, entonces ¿qué es lo que hace a Spring diferente? Para ponerlo más simple, Spring hace que el desarrollo de aplicaciones empresariales más sencillo. Pero Spring no solo resuelve los problemas a los desarrolladores, sino que pone en ejecución las buenas prácticas de programación orientada a objetos, como la codificación con interfaces, lo que reduce el acoplamiento y permite realizar pruebas con mayor facilidad. Además, como se dice en el argot del mundo informático “un buen diseño es más importante que la tecnología subyacente” y es eso de lo que precisamente hablamos desde el punto de vista de la arquitectura cuando se habla de insertar a Spring, una considerable mejora en el diseño.

En el momento en que se diseñó y pensó la actual arquitectura del módulo de Servicio Autónomo del Sistema SAREN, después de estas y otras muchas consideraciones, se optó por emplear Spring en la capa de lógica de negocio, debido a que, cuenta con las características ideales para el manejo de dicha capa, por el excelente manejo de recursos que ayuda grandemente en el rendimiento y optimización de la aplicación, por todo el conjunto de facilidades que le brinda a los desarrolladores, por la organización que proporciona de forma efectiva a nuestros objetos en la capa central, y por la excelente integración que proporciona con los demás frameworks utilizados en la aplicación.

El framework de Spring puede ser obtenido gratuitamente desde el mismo sitio del proyecto Spring Framework a través de la dirección web http://www.springframwork.org. Además, es posible obtener los archivos fuentes de Spring en esa misma ubicación. En el capítulo de la fundamentación teórica se mencionaba que Spring está diseñado para ser configurado mediante archivos XML de configuración y ficheros de propiedades. Para la configuración de Spring se cuenta con los archivos llamados contextos de aplicación (aplicationContext.XML), los cuales son leídos por el contenedor de inversión de control y son los que van a portar toda la descripción y los recursos disponibles, entre ellos los objetos de la capa central y las dependencias entre los mismos, ficheros de propiedades, entre otros. Estos archivos pueden estar localizados cualquier parte de la aplicación, pero como la aplicación es Web, siguiendo los estándares pues estos son ubicados en la ruta \WebContent\WEB-INF\ de la aplicación, tal y como lo descrito por Sun Microsystems para la organización de aplicaciones web. Con esos ficheros de configuración se puede modificar el funcionamiento de la aplicación o simplemente de una fracción de esta sin tener que recurrir a otra cosa.

Es necesario mantener en mente la idea que Spring sería utilizado para la capa de lógica de negocio, de una aplicación Web. En el siguiente fragmento se muestra cómo es que hace referencia a los ficheros de configuración del mismo, esto se encuentra contenido en el fichero \WebContent\WEB-INF\web.xml:

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
   ...
   /WEB-INF/applicationContext.xml
   ...
 </context-param>
```

En este segundo fragmento se configura un listener (o escucha) que procesa el anterior de forma automática cada vez que el contenedor de aplicaciones inicia la aplicación web:

```
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
   class>
</listener>
```

En este tercer fragmento se configura un listener que carga de forma automática las configuraciones necesarias para el traceo (logging) de la aplicación cada vez que el contenedor de aplicaciones inicia la aplicación web:

```
<listener>
  <listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>
</listener>
```

Esto no es suficiente. Al igual que Struts, es necesario indicar en el archivo que contiene el CLASSPATH de la aplicación, la ubicación se encuentran las librerías del framework y luego ya el framework está listo para ser utilizado.

Como el resto de las soluciones libres que conforman la arquitectura, Spring juega un importante papel dentro de la misma. Es el encargado de inicializar todos los objetos de la capa central, haciendo esto de forma eficiente, de manera que el sistema solo utilizará los recursos necesarios.

Spring tiene sus propios archivos de configuración, los APLICATIONCONTEXT.XML, y es exactamente en estos archivos donde se le indica al framework los objetos que deben ser inicializados y las dependencias

que existen entre estos. El siguiente es un fragmento del archivo APLICATIONCONTEXT-DATASOURCE-JNDI.XML:

```
<beans>
  ...
  <bean id="sarenDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName">
      <value>java:comp/env/jdbc/saren</value>
    </property>
  </bean>
</beans>
```

El fragmento anterior muestra la manera de declarar un objeto cuyo identificador es sarenDataSource el cual es de la clase JndiObjectFactoryBean, y que tiene una propiedad o atributo llamado jndiName cuyo valor es java:comp/env/jdbc/saren. A continuación se muestra un ejemplo de cómo se declara un objeto el cual tiene como atributo a otro objeto:

```
<beans>
   ...
   <bean id="saBaseDAO"
      class="com.saren.sa.dao.SABaseDAO">
      <property name="sessionFactory">
         <ref bean="sarenSessionFactory" />
      </property>
   </bean>
   ...
 </beans>
```

El fragmento anterior muestra la manera de declarar un objeto cuyo identificador es saBaseDAO el cual es de la clase SABaseDAO, y que tiene una propiedad o atributo llamado sessionFactory cuyo valor es otro objeto que fue declarado y que tiene como identificador sarenSessionFactory.

Por el momento se ha visto la forma de declarar los objetos que deben ser inicializados por Spring y como declarar además las dependencias entre ellos, es el caso de que un objeto tenga como propiedad a otro objeto. Entrando ya en cuestiones de configuración de recursos que serán utilizados por Spring es que se muestra el siguiente fragmento:

```
<beans>
   ...
   <bean id="propertyConfigurer"
      class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
      <property name="locations">
```

```
        <list>
           ...
           <value>classpath:db.properties</value>
           ...
        </list>
     </property>
  </bean>
  ...
</beans>
```

Así, pues, se le indica a Spring que los ficheros de propiedades o en este caso que el fichero DB.PROPERTIES se encuentra en la ruta classpath:db.properties. Estos ficheros contienen un conjunto de claves valores que pueden ser utilizados en otras configuraciones de Spring. En un apartado posterior donde se muestra la integración Hibernate con Spring se hará alusión a estos ficheros de propiedades.

3.3.5 Configuración de Hibernate

Es indisputable que las bases de datos relacionales se encuentran en el núcleo de las actuales aplicaciones empresariales. Mientras que los lenguajes de programación modernos, incluyendo a Java, proveen una vista intuitiva y orientada a objetos de las entidades, los datos empresariales subyacentes de dichas entidades es pesadamente relacional por naturaleza. Adicionalmente, la mayor fuerza del modelo relacional es que, por diseño, es intrínsecamente agnóstico a la manipulación programática y a la vista del nivel de aplicación de los datos a que sirve.

Muchos han sido los intentos para establecer un puente relacional y una tecnología orientada a objetos, o para reemplazar uno con el otro, pero la brecha entre ambos es uno de los hechos más duros de la computación empresarial de hoy en día. Ese es el reto, proveer un puente entre los datos relacionales y los objetos de Java, que Hibernate toma con sus mapeos objeto relacionales. Hibernate toma ese desafío de una manera muy pragmática, directa y realística.

Solo porque es posible mover ramitas por el suelo con la nariz no quiere decir que sea necesariamente la mejor manera de recoger leña. Java permite conectar directamente a cualquier base de datos mediante el puente JDBC/ODBC. Esta es la manera más sencilla para los desarrolladores de acceder a datos, pero no siempre lo más simple es lo más conveniente, debido a que el programador debe insertar el código del gestor que esté utilizando dentro de su aplicación y cada gestor de bases de datos tiene su propio código.

Como se puede ver, esto sería muy poco mantenible y portable. En este punto es donde entra a jugar su papel Hibernate, proveer un mismo código de acceso a datos para cualquier gestor y una alta portabilidad.

Primeramente es necesario indicar en el CLASSPATH el lugar en el que se encuentran las clases de Hibernate. Para lograr un verdadero mapeo objeto relacional es necesario que cada entidad, o tabla, de la base de datos tenga como equivalente una clase JavaBean en la aplicación que tenga los mismos atributos, cada uno con el mismo tipo de dato, que la propia entidad. Este tipo de clase, en este caso, es también conocido como POJO (Plain Old Java Object). Estas clases cuentan con atributos y métodos para acceder o modificar los atributos o los llamados setters y getters. A final de cuentas esto no es otra cosa que la implementación del ValueObject (objeto de valor), VO en lo adelante, que será quien cargue la información, la clase persistente. Para asociar estas clases con su correspondiente tabla en la base de datos Hibernate utiliza los ficheros HBM.XML. Es decir, si la clase se llama USUARIO.JAVA, el correspondiente archivo HBM.XML sería USUARIO.HBM.XML. En dicho archivo se declaran las propiedades del VO y sus correspondientes nombres de columna en la base de datos, asociación de tipos de datos, referencias, relaciones de cardinalidad con otras tablas, etc., como se observa a continuación:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
      <class name="Usuario" table="USUARIO" schema="OFICINA">
           <id name="idusuario" type="integer">
               <column name="IDUSUARIO" precision="9" scale="0" />
               <generator class="increment" />
            </id>
            <many-to-one name="oficina"
                  class="Oficina" fetch="select" lazy="false">
                  <column name="IDOFICINA" precision="9" scale="0" />
            </many-to-one>
            ...
            <property name="nombreusuario" type="string">
                  <column name="NOMBREUSUARIO" length="20" />
            </property>
            ...
      </class>
</hibernate-mapping>
```

Las primeras tres líneas constituyen el encabezamiento estándar de un archivo de XML normal. La etiqueta hibernate-mapping engloba las clases utilizadas en el mapeo de Hibernate, para ello se utilizan

las etiquetas class que indican las clases. Estas etiquetas tienen además una serie de parámetros como son el name o nombre de la clase, table o el nombre de la entidad en la base de datos y el esquema (schema) de la base de datos, estas dos últimas en letras mayúsculas. Dentro de las etiquetas class se define la manera en que se mapean los atributos de la clase con los atributos de la entidad. Primero se comienza con el identificador o id, al que se le especifica el nombre (name) y el tipo de datos (type). Además se especifica si existen relaciones entre tablas, y por tanto entre clases: en este ejemplo many-to-one o lo que es lo mismo muchos a uno, que indica la cantidad de elementos a ambos lados de la relación en el orden indicado. A esta etiqueta se le especifica la clase relacionada (class), el nombre (name) con el que será identificada, la manera en que se traen los datos de la relación (fetch), en este caso por selección (select) y si se hace una captura de datos perezosa o lazy (true) o no (false). Luego de especificar los identificadores y las relaciones se procede a declarar los atributos y para esto se utiliza property (propiedad), la cual tiene los mismos parámetros que la etiqueta id. Cada una de las etiquetas recientemente mencionadas incluyen otra sub-etiqueta que es la llamada column (columna), a la que se le especifica el name o nombre de la columna de la tabla en la base de datos en letras mayúsculas y en dependencia del tipo de datos la longitud y/o la presición. En el caso anterior se enlaza la clase Usuario.java con la tabla USUARIO del esquema OFICINA de la base de datos. Además, se le indica que la llave principal, o el identificador que hace único a cada elemento contenido en la tabla USUARIO, está siendo representado por el atributo idusuario de tipo Integer (número entero). Se especifica que esta tabla tiene una relación muchos a uno con la tabla OFICINA, lo que indica que una oficina puede tener muchos usuarios, pero que un usuario solo puede existir en una y solo una oficina. También se especifican el resto de las propiedades o atributos de la clase Usuario, por ejemplo el nombreusuario de tipo String (cadena de caracteres) que se mapea contra el campo NOMBREUSUARIO de la tabla USUARIO y que permite una longitud máxima de 20 caracteres.

Hasta el presente momento se ha obviado un dato muy importante y es la manera en que se configura el propio Hibernate, de manera que pueda acceder a la base de datos. Para esto hay varias maneras y la que definitivamente se utilizó en la arquitectura puede ser apreciada en detalles en la sección de la integración entre Spring e Hibernate. Para configurar a Hibernate se puede usar un fichero HIBERNATE.PROPERTIES, el cual debe estar en el camino de la aplicación.

```
hibernate.dialect org.hibernate.dialect.Oracle
hibernate.connection.driver_class jdbc.oracledriver
```

```
hibernate.connection.url jdbc:oracle:// /
hibernate.connection.username nombre_de_usuario
hibernate.connection.password contrasenna
```

De otra manera, se puede configurar a través del archivo HIBERNATE.CFG.XML, también incluido en el camino de la aplicación:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-configuration>
   <session-factory>
      <property name=”connection.driver_class”>
         jdbc.oracledriver
      </property>
      <property name=”connection.url”>
         jdbc:oracle:// /
      </property>
      <property name=”connection.username”>
         nombre_de_usuario
      </property>
      <property name=”connection.password”>
         contrasenna
      </property>
      <property name=”dialect”>
         org.hibernate.dialect.Oracle
      </property>
      ...
   </session-factory>
</hibernate-configuration>
```

Las etiquetas hibernate-configuration o configuración de Hibernate engloban a la sesión-factory o factoría de sesiones, la que tiene las etiquetas property para especificar las propiedades. Como se puede apreciar, en ambos casos, archivo XML y archivo de propiedades, se hace referencia al driver (manejador) encargado de enlazar a Java con el determinado gestor de bases de datos, la URL de conexión, la que indica la identificación del equipo en el cual se encuentra el servidor que contiene el gestor de bases de datos que coincida con el driver, el nombre de usuario y la contraseña que permitan acceder al mismo y el dialecto.

El dialecto es la gran obra. Para cada gestor de bases de datos, o al menos para la gran mayoría de los gestores más utilizados en aplicaciones empresariales, Hibernate posee un dialecto, que es lo que se le especifica en este caso, el lenguaje que debe emplear para comunicarse con la base de datos. Cada sistema de bases de datos posee su propio lenguaje estructurado de consultas (SQL), algunos son muy

parecidos, pero a la corta o a la larga el lenguaje es diferente. Si el desarrollador implementa completamente su lógica de acceso a datos de la manera descrita al inicio de este apartado mediante el puente JDBC/ODBC para un gestor en específico y luego los requerimientos cambian, debe, por consiguiente cambiar todo el SQL de ese gestor empotrado en la aplicación al código del nuevo gestor. Sin lugar a dudas esto provoca que la aplicación sea dependiente del gestor, además de la resultante modificación y re-compilación del código fuente de la aplicación en sí. Hibernate provee un acceso a datos objetual, es decir, mediante objetos se pueden realizar operaciones de inserción, modificación, eliminación y consultas de información a la base de datos. Por si esto fuera poco, Hibernate introduce un nuevo lenguaje, el Hibernate Query Languaje (lenguaje de consultas de Hibernate) o HQL. Este lenguaje es también orientado a objetos y es único y el mismo para cada tipo de gestor de bases de datos.

En resumen, Hibernate proporciona una infraestructura capaz de migrar entre bases de datos con solo modificar unos cuantos parámetros en un simple archivo de configuración y sin la mínima necesidad de entrar al código a realizar modificaciones, además de un acceso a datos orientado a objetos, con todas las pertinentes ventajas que esto trae consigo. Además, Hibernate proporciona un alto grado de independencia en cuanto al acceso a datos para las aplicaciones que lo empleen y con esto hace que sean más portables. Por otro lado, el uso de este framework provee un alto rendimiento en cuanto a lo relacionado con la transaccionabilidad, debido a que ofrece una fácil manipulación de transacciones.

3.4 Integración entre los frameworks

Es bien sabido que en la unión está la fuerza, entonces, al poner a trabajar al unísono los frameworks anteriormente mencionados se crea una sólida unidad, la cual deriva en una arquitectura de software potente, limpia, libre y probada. En el caso de la presente arquitectura toda la integración gira en torno a Spring debido a las facilidades que ofrece en ese sentido y porque permite integrarse directamente tanto con Struts como con Hibernate.

3.4.1 Integración de Struts con Spring

A pesar de que Spring tiene la capacidad de manipular la capa de lógica de presentación se decidió utilizar Struts con ese fin. Como se ha dicho con anterioridad, Spring tiene a su cargo la lógica de negocios y la integración entre los frameworks. Debido a esto, y como Struts maneja la capa de lógica de presentación, que finalmente se dibuja en la capa cliente como código HTML, es necesario tener un

mecanismo que enlace a Struts con Spring. Es por ello que resulta muy importante la integración de Struts con Spring, la que se logra partiendo de un archivo de configuración XML que se localiza dentro de la ruta \WEBCONTENT\WEB-INF y que se llama APPLICATIONCONTEXT-ACTION.XML. Entonces, la idea es que Struts va a procesar las solicitudes del cliente y llamar a los métodos de negocio en Spring y de igual manera mostrar los resultados al usuario. El siguiente es un ejemplo del contenido de dicho archivo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "WEB-INF/dtd/spring-beans.dtd">
<beans>
   ...
   <bean name="/cambiarContrasenna" class="CambiarContrasennaAction">
      <property name="adminUsuariosBO">
            <ref bean="adminUsuariosBO" />
      </property>
   </bean>
   ...
</beans>
```

Sensible

Las etiquetas beans engloban la configuración de las clases de negocio y las clases action de Struts. Para lograr esto a las sub-etiquetas bean se le especifica el name o nombre que recibe el action en el formulario web y además se le especifica la ubicación de la clase action de Struts. Dentro, en las propiedades se especifica el nombre (name) con que Spring reconoce las clases de negocio y además a la clase de negocio (bean) a que relaciona. Esto le indica a Spring que existe una clase localizada en CambiarContrasennaAction y que va a ser referenciada como la acción de un formulario en /cambiarContrasenna. Dicha clase va a tener acceso a la propiedad adminUsuariosBO, la cual hace alusión a los métodos de negocio en la clase de negocio referenciada por adminUsuariosBO. Pero Spring necesita saber en qué lugar encontrar las clases de negocio a que se hace referencia en el APPLICATIONCONTEXT-ACTION.XML. Para ello se utiliza otro fichero de configuración XML, el APPLICATIONCONTEXT-BO.XML, situado en \WEBCONTENT\WEB-INF. A continuación se muestra la estructura de dicho archivo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "WEB-INF/dtd/spring-beans.dtd">
<beans>
   <bean id="adminUsuariosBO"  class="AdminUsuariosBO">
      ...
      <property name="usuarioDAO"><ref bean="usuarioDAO" /></property>
      ...
```

```
   </bean>
</beans>
```

En este caso las etiquetas funcionan de igual manera que en el XML anterior, a excepción de que en este caso bean utiliza un id en lugar de name y que se utiliza para referenciar los métodos del negocio. Como se puede apreciar, adminUsuariosBO referencia a la clase de negocio AdminUsuariosBO, la que tiene, entre otras, la propiedad nombrada usuarioDAO, que a su vez hace referencia a la clase de acceso a datos usuarioDAO. No es difícil darse cuenta que el framework de Spring vuelve a necesitar saber en qué lugar se encuentran las clases de acceso a datos referenciadas en este último archivo de configuración. Finalmente, esto se le especifica en el archivo de configuración APPLICATIONCONTEXT-DAO.XML, también localizado en la carpeta \WEBCONTENT\WEB-INF. Lo que sigue es un ejemplo de dicho archivo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "WEB-INF/dtd/spring-beans.dtd">
<beans>
   ...
   <bean id="usuarioDAO" class="UsuarioDAO" parent="saBaseDAO">   </bean>
   ...
</beans>
```

Aquí todo permanece de igual manera solo que con esta configuración se le especifica con exactitud la localización de la clase que contiene los métodos de acceso a datos a que hace referencia la clase con los métodos de negocio. Obsérvese que el nombre de los archivos de configuración XML terminan conforme a las clases que referencian y además indican en que se especializan dichas clases: Action para las clases de acción de Struts que intervendrán en las acciones del usuario y las respuestas, BO (Business Objects u objetos de negocio) para las clases dedicadas específicamente al negocio, DAO (Data Access Objects u objetos de acceso a datos) para aquellas clases especializadas en el acceso a datos. Estos dos últimos tipos de clases le pertenecen a Spring.

Ahora bien, ya Spring sabe que va a manipular solicitudes de Struts, pero también es necesario decirle a Struts donde están los métodos del negocio a través de los cuales se va a comunicar con Spring. Para lograr esto, dentro de las clases Action de Struts se le especifican atributos privados que tienen como tipo de dato las interfaces implementadas por las clases de negocio de Spring. Además, para dichos atributos es necesario especificar los métodos de acceso y modificación, o los llamados setters y getters, como se muestra:

```
private IAdminUsuariosBO adminUsuariosBO;
public IAdminUsuariosBO getAdminUsuariosBO() {
      return adminUsuariosBO;
}
public void setAdminUsuariosBO(IAdminUsuariosBO adminUsuariosBO) {
      this.adminUsuariosBO = adminUsuariosBO;
}
```

Estas sencillas líneas de código declaran un atributo privado en la clase action de Struts que referencia a una de las interfaces de negocio definidas sobre Spring y además los métodos get… para acceder dicho atributo y set… para el cambio de valor del mismo. Resulta interesante el hecho de que no se especifican constructores para estos atributos para obtener instancias de las clases de negocio, sino que mediante el mecanismo de inversión del control se accede a los métodos de negocio de las clases de negocio de Spring. De esta manera quedan integrados Struts y Spring y vuelve a resurgir la importancia de los archivos de configuración XML, reiterando que con ellos se logra que la aplicación sea más portable y escalable, siendo esto un importante factor para la arquitectura.

3.4.2 Integración de Hibernate con Spring

¿Y cómo se integra Spring con Hibernate? Es necesario que el negocio tenga acceso a la información contenida en la base de datos, por eso es necesario tener un mecanismo que enlace a Hibernate con Spring. Resulta muy importante la integración de Hibernate con Spring, la que se logra partiendo de varios archivos de configuración XML que se localizan dentro de la ruta \WEBCONTENT\WEB-INF y que se llaman APPLICATIONCONTEXT-DATASOURCE-JNDI.XML, APPLICATIONCONTEXT-COMMON-BUSINESS.XML, APPLICATIONCONTEXT-DAO.XML, y APPLICATIONCONTEXT-BO.XML. Entonces, la idea es que las clases de negocio manejadas por Spring hacen uso de los métodos de acceso a datos en Hibernate. A continuación se muestra ejemplos del contenido de cada uno de estos archivos.

Para el mejor entendimiento de las configuraciones que se muestran a continuación, es válido recordar que se propuso usar Tomcat como servidor Web. Tomcat permite manejar las reservas de conexiones (connections pool) posibilitando el uso eficiente de tecnologías como Hibernate. Usando Tomcat las fuentes de datos (DataSources) se localizarían mediante JNDI, y los atributos de los mismos se encontrarían en el fichero de configuración del servidor (SERVER.XML). Véase ejemplo de código de configuración de las fuentes de datos en el Tomcat.

```
<?xml version="1.0" encoding="UTF-8"?>
 ...
 <Context debug="0" docBase="${catalina.home}/aplicacion"
          path="/ServicioAutonomo" reloadable="true">
  <Resource auth="Container" name="jdbc/saren" type="javax.sql.DataSource"/>
  <ResourceParams name="jdbc/saren">
    <parameter>
      <name>url</name>
      <value>jdbc:oracle:thin:@10.7.10.100:1521:basedatos</value>
    </parameter>
    <parameter>
      <name>maxIdle</name>
      <value>10</value>
    </parameter>
    <parameter>
      <name>maxActive</name>
      <value>20</value>
    </parameter>
    <parameter>
      <name>driverClassName</name>
      <value>oracle.jdbc.driver.OracleDriver</value>
    </parameter>
    <parameter>
      <name>maxWait</name>
      <value>-1</value>
    </parameter>
    <parameter>
      <name>removeAbandoned</name>
      <value>true</value>
    </parameter>
    <parameter>
      <name>username</name>
      <value>nombreusuario</value>
    </parameter>
    <parameter>
      <name>factory</name>
      <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </parameter>
    <parameter>
      <name>logAbandoned</name>
      <value>true</value>
    </parameter>
    <parameter>
      <name>defaultAutoCommit</name>
      <value>false</value>
    </parameter>
    <parameter>
      <name>removeAbandonedTimeout</name>
      <value>60</value>
    </parameter>
```

```
    <parameter>
      <name>password</name>
      <value>contrasenna</value>
    </parameter>
  </ResourceParams>
 </Context>
```

Hasta el presente se tiene configurada una fuente de datos con nombre “jdbc/saren”, la cual es obtenida posteriormente por Spring y utilizada por Hibernate. La forma de obtener esta fuente de datos es mostrada en el contexto de aplicación de Spring que está en el fichero APPLICATIONCONTEXT-DATASOURCE-JNDI.XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="sarenDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName">
      <value>java:comp/env/jdbc/saren</value>
    </property>
  </bean>
</beans>
```

En el fragmento anterior se declara un objeto cuyo identificador es sarenDataSource el cual es de la clase JndiObjectFactoryBean, y que tiene una propiedad o atributo llamado jndiName cuyo valor es “java:comp/env/jdbc/saren”. En cuanto al valor asignado a la propiedad jndiName simpre está compuesto por la cadena “java:comp/env/” más el nombre de la fuente de datos declarada en el fichero del Tomcat, en este caso es “jdbc/saren”, por tanto el valor del atributo jndiName es “java:comp/env/jdbc/saren”.

La configuración general de Hibernate, Session Factory y el gestor de transacciones, se establece en el contexto de aplicación de Spring que está en el fichero APPLICATIONCONTEXT-COMMON-BUSINESS.XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "WEB-INF/dtd/spring-beans.dtd">
<beans>
  <bean id="sarenSessionFactory"
  class="org.springframework.orm.hibernate3.LocalSessionFactoryBean" lazy-init="true">
    <property name="dataSource">
      <ref bean="sarenDataSource" />
    </property>
    <property name="mappingDirectoryLocations">
```

```
      <list>
        <value>classpath:/com/saren/sa/vo </value>
      </list>
    </property>
    <property name="hibernateProperties">
      <props>
        <prop key="hibernate.dialect">${hibernate.dialect}</prop>
        <prop key="hibernate.cglib.use_reflection_optimizer">
            ${hibernate.cglib.use_reflection_optimizer}
        </prop>
        <prop key="show_sql">${show_sql}</prop>
        </props>
      </property>
  </bean>
  ...
</beans>
```

En el fragmento anterior se indica la manera en que a través de Spring se obtiene la Session Factory que es utilizada por Hibernate. Para esto se declara un objeto cuyo identificador es sarenSessionFactory, este tiene una propiedad llamada dataSource cuyo valor es la referencia al objeto sarenDataSource declarado con anterioridad en el fichero APPLICATIONCONTEXT-DATASOURCE-JNDI.XML. La Session Factory requiere además de la configuración de las propiedades mappingDirectoryLocations y hibernateProperties respectivamente. La propiedad mappingDirectoryLocations se refiere a la ubicación de los ficheros de mapeo utilizados por Hibernate, en este caso están ubicados en la ruta “classpath:/com/saren/sa/vo”. La propiedad hibernateProperties se refiere a su vez a un conjunto de propiedades de Hibernate, “hibernate.dialect”, "hibernate.cglib.use_reflection_optimizer", y "show_sql". La propiedad “hibernate.dialect” se refiere al dialecto SQL a utilizar, puesto que es diferente para cada gestor de base de datos. La propiedad "hibernate.cglib.use_reflection_optimizer" es para declarar si se utilizará la funcionalidad de optimización de reflection brindada por el framework cglib. La propiedad "show_sql" indica si se muestran o no en la consola del servidor las sentencias sql generadas por Hibernate. Los valores de estas últimas propiedades están ubicados en el fichero DB.PROPERTIES. A continuación se muestran los mismos.

```
hibernate.dialect=org.hibernate.dialect.Oracle9Dialect
hibernate.cglib.use_reflection_optimizer=true
show_sql=false
```

La manera de indicarle a Spring la ubicación del fichero de configuración DB.PROPERTIES es como se muestra a continuación.

```
<beans>
  ...
   <bean id="propertyConfigurer"
      class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
      <property name="locations">
         <list>
            ...
            <value>classpath:db.properties</value>
            ...
         </list>
      </property>
   </bean>
  ...
 </beans>
```

El segmento anterior se encuentra ubicado en el contexto de aplicación de Spring que está en el fichero APPLICATIONCONTEXT-COMMON-BUSINESS.XML. A continuación se muestra el segmento en el que se indica el gestor de transacciones que se utiliza.

```
<bean id="transactionManager"
  class="org.springframework.orm.hibernate3.HibernateTransactionManager">
  <property name="sessionFactory">
    <ref bean="sarenSessionFactory" />
  </property>
</bean>
```

Esto le indica a Spring que el gestor de transacciones que se utiliza es el relacionado a la Session Factory de Hibernate, HibernateTransactionManager. El identificador del gestor es transactionManager, este tiene una propiedad nombrada sessionFactory que tiene como valor la referencia del objeto sarenSessionFactory declarado anteriormente. La referencia al objeto sarenSessionFactory es utilizada por los objetos de acceso a datos que son declarados en el contexto de aplicación de Spring ubicados en el fichero APPLICATIONCONTEXT-DAO.XML, a continuación se muestra un fragmento del mismo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "WEB-INF/dtd/spring-beans.dtd">
<beans>
   <bean id="saBaseDAO" class="SABaseDAO" lazy-init="true">
     <property name="sessionFactory">
       <ref bean="sarenSessionFactory" />
     </property>
   </bean>
   <bean id="usuarioDAO" class="UsuarioDAO" parent="saBaseDAO"></bean>
</beans>
```

En el segmento anterior se muestra la declaración de un objeto con identificador saBaseDAO el cual tiene una propiedad llamada sessionFactory y el valor de esta es la referencia al objeto sarenSessionFactory declarado anteriormente. También se muestra el objeto declarado de identificador usuarioDAO el cual hereda todo del objeto saBaseDAO.

Por último se muestra como las clases de negocio manejadas por Spring utilizan las clases de acceso a datos. Las respectivas configuraciones se realizan en el contexto de aplicación de Spring ubicados en el fichero APPLICATIONCONTEXT-BO.XML. A continuación de muestra un segmento de dicho fichero.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "WEB-INF/dtd/spring-beans.dtd">
<beans>
   <bean id="adminUsuariosBO"  class="AdminUsuariosBO">
      ...
      <property name="usuarioDAO"><ref bean="usuarioDAO" /></property>
      ...
   </bean>
</beans>
```

Como se puede apreciar, adminUsuariosBO referencia a la clase de negocio AdminUsuariosBO, la que tiene, entre otras, la propiedad nombrada usuarioDAO, que a su vez hace referencia a la clase de acceso a datos UsuarioDAO.

Con esta configuración se le especifica con exactitud la localización de la clase que contiene los métodos de acceso a datos a que hace referencia la clase con los métodos de negocio. De esta manera quedan integrados Hibernate y Spring y vuelve a resurgir la importancia de los archivos de configuración XML, reiterando que con ellos se logra que la aplicación sea más portable y escalable, siendo esto un importante factor para la arquitectura.

3.4.3 Integración de Acegi con Spring

Para gestionar la autenticación y autorización del sistema se utilizará Acegi, que permite integrar de manera sencilla y no intrusiva toda una serie de características de seguridad en una aplicación que utilice Spring. Acegi es un framework de seguridad de software libre diseñado para Spring. Creado por Ben Alex, el framework tiene una extensa lista de características, excelente cobertura de pruebas, facilidad de uso, y cercana colaboración con Spring. Acegi nos ofrece aspectos de seguridad muy avanzados, tales como la

funcionalidad AOP basada en Spring para proteger los objetos de negocio y la funcionalidad de las listas de control de acceso (ACL) para la seguridad de las instancias de los objetos. A la hora de desarrollar una aplicación empresarial con Spring hay cuatro conceptos de seguridad que típicamente deben ser considerados: Autenticación, seguridad en las peticiones Web, seguridad de la capa de servicio (por ejemplo los métodos que implementan la lógica de negocio), y seguridad en el dominio de la instancia del objeto (por ejemplo, diferentes dominios de objetos poseen diferentes permisos.), estos son resueltos elegantemente mediante el uso de Acegi.

En el momento en que se diseñó y pensó la actual arquitectura del módulo de Servicio Autónomo del Sistema SAREN, después de estas y otras muchas consideraciones, se optó por emplear Acegi para el manejo de la seguridad del sistema, debido a que, cuenta con características excelentes para el control de este importante aspecto. Por la excelente integración que proporciona con Spring, colaborando de forma eficaz con el contenedor de IoC de Spring y a la vez tomando los beneficios de este en cuanto a rendimiento y optimización. Además que Acegi proporciona a los desarrolladores una gran facilidad en cuanto a su configuración.

El framework de Acegi puede ser obtenido gratuitamente desde el sitio del proyecto Acegi Security System for Spring a través de la dirección web http://www.acegisecurity.org. Además, es posible obtener los archivos fuentes de Acegi en esa misma ubicación. En el capítulo de la fundamentación teórica se mencionaba que Acegi está diseñado para ser configurado mediante archivos XML de configuración. Para la configuración de Acegi se cuenta con los archivos llamados contextos de aplicación (aplicationContext.XML) de Spring, los cuales son leídos por el contenedor de inversión de control al arrancar o recargar la aplicación. Estos archivos son ubicados en la ruta \WebContent\WEB-INF\ de la aplicación. Además, Acegi proporciona implementaciones con las que la información de usuario puede estar en memoria, útil en entornos de prueba, en una base de datos accesible vía JBDC, o en un servicio JAAS. Para obtener una total independencia del sistema gestor de base de datos será necesario realizar una implementación que utilice Hibernate para acceder a la información.

En cuanto a la configuración de Acegi ésta se realiza dentro del contexto de aplicación de Spring, en un fichero APPLICATIONCONTEXT-ACEGI-SECURITY.XML que puede ser incluido o excluido en la configuración, habilitando o no las características de autenticación y autorización, útil para la realización

de tests. El DAO implementado con Hibernate y que es utilizado por Acegi para la gestión de los usuarios es definido en el fichero APPLICATIONCONTEXT-DAO.XML como se muestra a continuación.

```
<beans>
   ...
   <bean id="usuarioAuthenticationDAO" class="UsuarioSADAO" parent="saBaseDAO"></bean>
   ...
</beans>
```

Entrando en las configuraciones de Acegi en el fichero APPLICATIONCONTEXT-ACEGI-SECURITY.XML, se tiene primeramente que Acegi proporciona un cifrador de contraseñas utilizando el algoritmo MD5 el cual asegura que la contraseña viaje de forma cifrada entre el Servidor Web y el Servidor de Base de Datos. A continuación se muestra como se define este cifrador.

```
<beans>
   ...
   <bean id="passwordEncoder"
            class="org.acegisecurity.providers.encoding.Md5PasswordEncoder" />
</beans>
```

Acegi proporciona un mecanismo para guardar la información de autenticación de usuario en caché basada en EHCache, de forma que pueda no tiene que estar accediendo constantemente a la Base de Datos para chequear los permisos de un usuario que ya se ha autenticado en el sistema. A continuación se muestra la definición del mismo.

```
<beans>
   ...
   <bean id="cacheManager"
      class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean" />
   <bean id="userCacheBackend"
      class="org.springframework.cache.ehcache.EhCacheFactoryBean">
      <property name="cacheManager">
        <ref local="cacheManager" />
      </property>
      <property name="cacheName">
        <value>userCache</value>
      </property>
   </bean>
   <bean id="userCache"
      class="org.acegisecurity.providers.dao.cache.EhCacheBasedUserCache">
      <property name="cache">
        <ref local="userCacheBackend" />
      </property>
   </bean>
</beans>
```

Ahora que se tienen definidos una DAO, un cifrador de contraseñas y un mecanismo para guardar la información de autenticación de usuario en caché, entonces es que se define un proveedor de autenticación que utiliza un DAO para acceder a la información, donde se puede configurar entre otros el cifrado de contraseñas y la caché de usuarios. A continuación se muestra la definición del mismo.

```
<beans>
   <bean id="daoAuthenticationProvider"
   class="org.acegisecurity.providers.dao.DaoAuthenticationProvider">
     <property name="saltSource">
       <bean class="org.acegisecurity.providers.dao.salt.ReflectionSaltSource">
         <property name="userPropertyToUse">
           <value>getUsername</value>
         </property>
       </bean>
     </property>
     <property name="userDetailsService">
       <ref bean="usuarioAuthenticationDAO" />
     </property>
     <property name="userCache">
       <ref local="userCache" />
     </property>
     <property name="passwordEncoder">
       <ref local="passwordEncoder" />
     </property>
   </bean>
   ...
</beans>
```

Es necesario definir además un proveedor de autenticación anónima, esto se utiliza para el caso de las páginas de la aplicación que pueden ser accedidas sin necesidad de autenticación. A continuación se muestra la definición del mismo.

```
<beans>
   ...
   <bean id="anonymousProcessingFilter"
      class="org.acegisecurity.providers.anonymous.AnonymousProcessingFilter">
      <property name="key">
        <value>foobar</value>
      </property>
      <property name="userAttribute">
        <value>anonymousUser,ROLE_ANONYMOUS</value>
      </property>
   </bean>
   <bean id="anonymousAuthenticationProvider"
      class="org.acegisecurity.providers.anonymous.AnonymousAuthenticationProvider">
      <property name="key">
        <value>foobar</value>
```

```
      </property>
   </bean>
</beans>
```

Teniendo definidos los proveedores de autenticación, se define entonces el gestor de autenticación “authenticationManager” en el que se referencia a los proveedores.

```
<beans>
   ...
   <bean id="authenticationManager"
      class="org.acegisecurity.providers.ProviderManager">
      <property name="providers">
        <list>
            <ref local="daoAuthenticationProvider" />
            <ref local="anonymousAuthenticationProvider" />
        </list>
      </property>
   </bean>
   ...
</beans>
```

En cuanto a autorización, las decisiones se tomarán basándose en los roles del usuario como se muestra a continuación.

```
<beans>
   ...
   <bean id="httpRequestAccessDecisionManager"
      class="org.acegisecurity.vote.AffirmativeBased">
      <property name="allowIfAllAbstainDecisions">
        <value>false</value>
      </property>
      <property name="decisionVoters">
        <list>
          <ref bean="roleVoter" />
        </list>
      </property>
   </bean>
   <bean id="roleVoter" class="org.acegisecurity.vote.RoleVoter" />
   ...
</beans>
```

El sistema utilizado por Acegi para implementar los mecanismos de seguridad en los recursos web está basado en el uso de filtros que interceptan las peticiones HTTP y realizan algún tipo de procesamiento tomando las medidas oportunas. Hasta el momento se han mostrado un conjunto de configuraciones que corresponden a las medidas a tomar por Acegi después haber sido interceptadas dichas peticiones. A

continuación corresponde mostrar de qué forma se configura Acegi para la intercepción de las peticiones HTTP, comenzando por el filtro que se encarga de procesar el formulario de login de un usuario “authenticationProcessingFilter”, donde se referencia el gestor de autenticación “authenticationManager” anteriormente configurado y la página a la que ir en caso de error en el login, en este también se define la página a la que ir en caso de que la autenticación sea correcta, y por último se define la url a la cual son redireccionadas las peticiones http para que Acegi realice el respectivo procesamiento.

```
<beans>
   ...
   <bean id="authenticationProcessingFilter"
      class="org.acegisecurity.ui.webapp.AuthenticationProcessingFilter">
      <property name="authenticationManager">
        <ref bean="authenticationManager" />
      </property>
      <property name="authenticationFailureUrl">
        <value>/login.do?login_error=1</value>
      </property>
      <property name="defaultTargetUrl">
        <value>/inicio.do</value>
      </property>
      <property name="filterProcessesUrl">
        <value>/j_acegi_security_check</value>
      </property>
   </bean>
   ...
</beans>
```

Además es necesario definir el “authenticationProcessingFilterEntryPoint” donde se configura la página donde se encuentra el formulario de login, además en este se define si se hará uso del mecanismo para redirección a canales seguros HTTPS.

```
<beans>
   ...
   <bean id="authenticationProcessingFilterEntryPoint"
      class="org.acegisecurity.ui.webapp.AuthenticationProcessingFilterEntryPoint">
      <property name="loginFormUrl">
        <value>/login.do</value>
      </property>
      <property name="forceHttps">
        <value>true</value>
      </property>
   </bean>
   ...
</beans>
```

Acegi además proporciona el mecanismo para redirección automática de peticiones HTTP a canales seguros HTTPS para aquellas URLS que lo requieran. La configuración para ello se muestra a continuación.

```
<beans>
   ...
   <bean id="secureChannelProcessor"
      class="org.acegisecurity.securechannel.SecureChannelProcessor" />
   <bean id="insecureChannelProcessor"
      class="org.acegisecurity.securechannel.InsecureChannelProcessor" />
   <bean id="channelDecisionManager"
      class="org.acegisecurity.securechannel.ChannelDecisionManagerImpl">
      <property name="channelProcessors">
        <list>
          <ref local="secureChannelProcessor" />
          <ref local="insecureChannelProcessor" />
        </list>
      </property>
   </bean>
   <bean id="channelProcessingFilter"
      class="org.acegisecurity.securechannel.ChannelProcessingFilter">
      <property name="channelDecisionManager">
        <ref local="channelDecisionManager" />
      </property>
      <property name="filterInvocationDefinitionSource">
        <value>
          CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
          PATTERN_TYPE_APACHE_ANT
          /login.do**=REQUIRES_SECURE_CHANNEL
          /cambiarcontrasenna.do**=REQUIRES_SECURE_CHANNEL
          /j_acegi_security_check=REQUIRES_SECURE_CHANNEL
          **=REQUIRES_INSECURE_CHANNEL
        </value>
      </property>
   </bean>
   ...
</beans>
```

En este caso se han definido dos procesadores de canales, uno para canal seguro “secureChannelProcessor” y otro para canal inseguro “insecureChannelProcessor” los cuales son usados posteriormente por el gestor de canales “channelDecisionManager”, este a su vez es utilizado por el filtro para procesamiento de canal "channelProcessingFilter" y en el es donde se definen las reglas mediante expresiones de ANT para que Acegi conozca cuales urls deben redireccionarse a canales seguros y cuales a canales inseguros.

Se debe configurar además un filtro para restringir el acceso a determinadas urls. Este filtro al igual que los anteriores se configura mediante el contexto de aplicación de Spring, donde se define un FilterSecurityInterceptor, “filterInvocationInterceptor”, que define los roles necesarios para acceder a las urls utilizando comodines y expresiones de ANT.

```
<beans>
   ...
   <bean id="filterInvocationInterceptor"
      class="org.acegisecurity.intercept.web.FilterSecurityInterceptor">
      <property name="authenticationManager">
        <ref bean="authenticationManager" />
      </property>
      <property name="accessDecisionManager">
        <ref local="httpRequestAccessDecisionManager" />
      </property>
      <property name="objectDefinitionSource">
        <value>
          CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
          PATTERN_TYPE_APACHE_ANT
          /login.do=ROLE_ANONYMOUS,ROLE_ADMINISTRADOR_SA
          /inicio.do=ROLE_ADMINISTRADOR_SA,ROLE
          /logoff.do=ROLE_ADMINISTRADOR_SA
          /cambiarcontrasenna.do**=ROLE_ADMINISTRADOR_SA,ROLE_CAMBIAR_CONTRASENNA
          /**=ROLE_ACCESO_DENEGADO
        </value>
      </property>
   </bean>
   ...
</beans>
```

A continuación se define un filtro que es el responsable de almacenar el SecurityContext de Acegi en la HtttpSession entre las peticiones HTTP.

```
<beans>
   <bean id="httpSessionContextIntegrationFilter"
            class="org.acegisecurity.context.HttpSessionContextIntegrationFilter">
   </bean>
   ...
</beans>
```

Se debe definir además un filtro que tiene la responsabilidad de detectar cualquier excepción de seguridad de Acegi que sea lanzada. La configuración del mismo se muestra a continuación.

```
<beans>
   ...
   <bean id="exceptionTranslationFilter"
      class="org.acegisecurity.ui.ExceptionTranslationFilter">
      <property name="authenticationEntryPoint">
            <ref local="authenticationProcessingFilterEntryPoint" />
      </property>
   </bean>
   ...
</beans>
```

Solo falta agregar un filtro que es imprescindible para que los demás funcionen ya que este invoca a todos los demás en el orden especificado en la expresión.

```
<beans>
   ...
   <bean id="filterChainProxy"
      class="org.acegisecurity.util.FilterChainProxy">
      <property name="filterInvocationDefinitionSource">
        <value>
          CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
          PATTERN_TYPE_APACHE_ANT
          /**=channelProcessingFilter,httpSessionContextIntegrationFilter,
          authenticationProcessingFilter,anonymousProcessingFilter,
          exceptionTranslationFilter,filterInvocationInterceptor
        </value>
      </property>
   </bean>
   ...
</beans>
```

Como se ha podido observar Acegi proporciona los mecanismos de seguridad necesarios para la aplicación, es flexible y sencillo de configurar, sin necesidad de modificar código ya existente. Su integración con Spring hace que sea el recomendado para añadir las funcionalidades de seguridad a las aplicaciones que utilicen este magnífico framework.

3.5 Ventajas y desventajas de la arquitectura propuesta

Los hombres han sido capaces de crear, hacer grandes descubrimientos e incluso ha llegado a modificar el ambiente natural en su propio beneficio y en pos del bien de la sociedad. Pero todo lo humanamente hecho es imperfecto y por tanto todo tiene sus ventajas y sus desventajas, aunque estas últimas sean mínimas y se trabaje incansablemente para reducirlas al máximo o al menos para que su efecto sea

menor. Comúnmente se comienza hablando acerca de las ventajas de cierto producto y luego en aquello que tiene debilidades. Es muy cierto que por lo general las ventajas son tales, son tan significativas y tan importantes que las adversidades pasan por completo a un plano nunca mencionado.

3.5.1 Ventajas

Entonces ¿cuáles son las ventajas que presupone el uso de esta arquitectura de software en el desarrollo de aplicaciones empresariales J2EE?

1. La primera ventaja se deriva de la modularidad del diseño:
   - Struts para la vista
   - Spring para la integración y el negocio
   - Hibernate para la persistencia
   - Acegi para la seguridad

   Cada una de las partes empleadas es intercambiable de forma sencilla y limpia por otras soluciones disponibles, teniendo en cuenta que el diseño se ha realizado reduciendo al mínimo posible las dependencias entre las diferentes capas.
2. Otra gran ventaja es la reutilización de código que propone el uso de frameworks, puesto, como se ha mencionado con anterioridad los frameworks son aplicaciones semicompletas y que facilitan el desarrollo de aplicaciones. Es decir, los frameworks permiten el hecho de no tener que reimplementar gran cantidad de aspectos y tópicos que vienen pre elaborados.
3. El uso de los patrones de diseño implícitos en los frameworks provee soluciones viables y escalables, dado que los patrones son soluciones probadas a problemas comunes. Esto no quiere decir otra cosa que el uso de patrones de diseño supone un código muy optimizado y eficiente, trayendo consigo las consecuentes ventajas.
4. El hecho de que se han utilizado los frameworks que más se utilizan en el mundo J2EE para el desarrollo de aplicaciones empresariales. Esto no es tanto una ventaja como una característica, aunque de manera abstracta se podría considerar como tal.

Cómo se puede apreciar es posible concluir que el uso de estos frameworks es una ventaja en sí. Por último y no menos substancial es muy bueno destacar la importancia de que esta arquitectura está pensada para un ambiente Java, específicamente J2EE. Por defecto, el uso de Java trae consigo independencia, comenzando por el sistema operativo. Es decir, si se desea migrar una aplicación

realizada con esta arquitectura de un servidor Windows a un servidor Unix o Linux, o viceversa, no representaría ningún tipo de problema. Esto está dado, como se ha mencionado ya en otra oportunidad por la gran portabilidad que ofrece Java, entonces y desde este punto, se podría tomar como otra ventaja el hecho de basar la arquitectura en un ambiente para dicho lenguaje.

3.5.2 Desventajas

Esta arquitectura, como todas las que se puedan encontrar o configurar, no solo para Java, sino para cualquiera que sea la plataforma de software escogida, tiene sus inconvenientes. Como ha sido mencionado con anterioridad, es una tarea de primer orden conocer las desventajas de algo para conocer sus debilidades y evitar a toda costa que pueda afectar el desarrollo, o que la afectación sea lo más leve posible. De modo que, las desventajas que puede presentar esta arquitectura son enumeradas a continuación:

1. El uso de múltiples frameworks: aunque esto trae muchas ventajas para la modularidad y la reutilización, en ocasiones puede resultar problemático para los desarrolladores menos experimentados, ya que cada framework tiene su propia estructura y características.
2. Las herramientas: la mayoría de las herramientas libres disponibles para implementar sistemas basados en esta arquitectura no brindan muchas facilidades para reducir el tiempo de desarrollo y por lo general no existe mucha documentación de cómo utilizarlas.
3. Struts: el uso de Struts para la capa de presentación presupone que el usuario deba implementar los eventos que ocurren en el lado del cliente mediante otro lenguaje de programación, como por ejemplo JavaScript, debido a que dicho framework no tiene las funcionalidades de manipulación de eventos, solamente conoce el evento de enviar información por sometimiento del formulario o submit. Normalmente los frameworks que si manipulan eventos lo que hacen es generar un código para manejarlos en el lado del cliente desde el lado del servidor que al final se renderiza en el cliente como Javascript, pero en este caso debe ser implementado por los desarrolladores.

3.6 Análisis de los resultados

Incluso el oro es probado con fuego y es en extremo importante el análisis de los resultados a los que se llega luego de completar una arquitectura como la especificada, lo cual se logra después de un exhaustivo, crítico y severo estudio y razonamiento de los logros y deficiencias obtenidos. Además, es importante decir que en cualquier proyecto de desarrollo de software es necesario ir analizando los

resultados por cada etapa y fase de la vida del mismo. El módulo de Servicio Autónomo del sistema SAREN no es una excepción y durante todo su tiempo de desarrollo pasó por varios procesos de análisis en todos y cada uno de los hitos según se iban alcanzando. Esto ocurre de igual manera con las arquitecturas de los sistemas informáticos, las que, por fuerza, se deben mantener en constante observación debido a que pueden sufrir cambios a medida que se desarrolle el producto.

El primer aspecto claramente visible es el relacionado con la arquitectura inicial. La arquitectura original sentaba las bases para lo que sería la arquitectura definitiva que regiría el desarrollo del sistema. Esta arquitectura inicial brindaba una serie de posibilidades y diversas funcionalidades con las que, evidentemente, sería completamente posible desarrollar el sistema. Es admisible decir que dicha arquitectura poseía un fundamento sencillo y consistente, pero tenía muchas deficiencias. Las principales deficiencias o carencias son enumeradas a continuación:

1. La principal deficiencia es indisputablemente lo referente a la seguridad, la que tendría que ser implementada en todos los sub-módulos o sub-sistemas de una manera muy invasiva. En este punto se podría haber pensado en alguna reutilización de código de algún tipo, pero aún así esto conllevaría a un código difícilmente mantenible, sin mencionar que invertiría tiempo extra en cuestiones de desarrollo.
2. El control de sesiones y roles debía ser implementado completamente por el usuario y, como el tema anterior, sería de una manera invasiva, dado que debería estar incluido en los diferentes sub-sistemas.
3. Por otra parte, la integración entre los frameworks, aunque posible mediante clase simples de Java o JavaBeans, no sería óptima, ni siquiera lo eficiente que se requiere para software empresarial de este tipo.
4. En otro orden, las transacciones y el negocio, además de que contaran con la intervención de los frameworks Hibernate y Struts respectivamente, tendrían una marcada dependencia en el código implementado por los desarrolladores. Esto no trae más que una amplia sumisión de la aplicación a no ser capaz de asumir cambios con facilidad en esos puntos, aparte del alto consumo de tiempo. Cuando se habla del desarrollo de un software a la medida como lo es el sistema SAREN esto no es factible, puesto que según se desarrolla es muy posible que existan cambios en los requerimientos funcionales.

El uso de frameworks por defecto incluye numerosos patrones de diseño que son trasparentes para los desarrolladores, es decir, los programadores no tienen que implementarlos, sino que el propio framework lo hace. Ya ha sido mencionado que esta arquitectura utilizaba dos frameworks, Struts e Hibernate, y con esto incluía varios patrones, algunos específicos para J2EE. Sin embargo, esto no era suficiente. En otras palabras, el hecho de los patrones está bien, pero esta característica no abarcaba todo lo realmente necesario para consolidar una arquitectura de software empresarial verdaderamente portable, escalable, que redujera el tiempo de desarrollo y facilitara este proceso. Además, era necesario que fuera algo de fácil mantenimiento. Estas características podrían haber sido logradas, pero nunca con la misma eficiencia y poco tiempo de desarrollo logrado posteriormente. Tampoco habrían tenido el mismo nivel de optimización.

Es entonces cuando, luego de haber analizado a fondo estas faltas, que se llega a la concesión de emplear un nuevo framework que, ante todo, permitiera una mayor modularización, una mayor facilidad y un menor tiempo de desarrollo, lo que sería de beneficio tanto para los desarrolladores como para los usuarios finales. Ahí es donde, como ya se ha dicho, entra en acción Spring, el que además de solucionar todas las deficiencias canalizadas con anterioridad, introduciría una serie de características y funcionalidades de importancia en beneficio de la arquitectura. Esto resultaría de provecho para todo lo que depende de esta, para el equipo de desarrollo y por ende para los usuarios. Con respecto a esto último a continuación se exponen algunas de las principales nuevas características introducidas por Spring:

- Desarrollo de aspectos para cubrir áreas transversales al desarrollo. El uso de la programación orientada a aspectos simplificaría muchas áreas de la programación, incrementando la reutilización de código y mejorando las posibilidades ofrecidas por la programación orientada a objetos clásica.
- Con la llegada de Spring el coste de integración se redujo de manera drástica.
- La inversión del control que, como su nombre lo indica, invierte el control, siendo el framework quien llama a los métodos del desarrollador.
- La integración de Acegi con Spring proporcionó un alto nivel de seguridad no intrusiva o invasiva, así como el control de sesiones y accesos. Todo esto a base de archivos de configuración.

Con respecto a este último punto cabe recordar que cuando las configuraciones se desarrollan con ficheros de configuración aumenta la escalabilidad y la mantenibilidad. Después de lo expuesto, está de

más decir que el uso de Spring solo trajo como resultado beneficios desde el punto de vista técnico y desde el punto de vista humano, ya que facilitó el desarrollo y propició una integración óptima.

El cambio de arquitectura dio, además, otro resultado tangible muy importante y fue que finalmente se creó una arquitectura de software empresarial para J2EE con una distintiva separación entre los niveles que forman las capas de la aplicación. En este sentido, el uso de los ficheros de configuración, tanto por parte de los frameworks empleados, como por los utilizados en el proceso de desarrollo por los programadores, proporcionan a la arquitectura un fundamento muy estable y portable. Esto se debe a que el código no necesita ser cambiado para nuevas configuraciones y porque además proporciona, aparte de la independencia proporcionada por Java en cuanto a la plataforma del sistema operativo, una alta portabilidad en cuanto a servidores de aplicaciones y gestores de bases de datos. Es válida la reiteración de que esto se logra a partir de la no necesidad de recompilar las aplicaciones basadas en esta arquitectura.

Además de esto, y un poco profundizando en la aplicación y sin perder el hilo de la arquitectura, también es un logro el hecho de poder realizar captura de imágenes desde un entorno web, así como el logro de la generación e impresión de reportes complejos desde un ambiente web. Esto es, y fue posible lograrlo a partir de la separación entre capas y la flexibilidad brindada por la arquitectura seleccionada de ser capaz de vincular distintos tipos de interfaces de usuario para proporcionar servicios prácticamente ilimitados. Estos dos servicios se lograron con soluciones libres, como el resto de la arquitectura y el uso de los llamados applets de Java. Los applets son aplicaciones Java que se ejecutan en el lado del cliente sobre un navegador web, y que por tanto, requieren la intervención de un ambiente de ejecución Java en el cliente. Entonces, el logro es precisamente ese, el hecho de ejecutar complejas aplicaciones Java en el navegador, la captura de imágenes incluso desde dispositivos como escáneres y la muestra, exportación en múltiples formatos e impresión de reportes, todo eso funcionando en el lado del cliente y con una estrecha relación con la lógica de negocios en el servidor de una manera rápida, sencilla, limpia y elegante, desde el punto de vista del software y para el usuario final.

Concluyendo este apartado, la arquitectura actual en sí misma es el mayor hito. Una arquitectura de software empresarial sólida, robusta, confiable, escalable y funcional, basada por entero en soluciones libres y de código abierto.

CONCLUSIONES

Contar con un modelo de arquitectura sólido, bien definido sobre el cual desarrollar, provee numerosos beneficios. De esta forma se hace más fácil la comprensión del sistema y se logra una mejor comunicación entre el equipo de trabajo. Esto permite a su vez organizar el proceso de desarrollo y definir la estructura que tendrá el sistema. Otra ventaja es que se fomenta la reutilización de componentes, lo que trae consigo ahorro de tiempo por no tener que implementar repetidas veces una misma funcionalidad, y el aumento de la calidad de los sistemas desarrollados. Además, se facilita la evolución del sistema, de forma tal que el mismo pueda ser modificado o incluso crecer sin ser afectado de forma considerable.

Como resultado de la investigación realizada en el presente trabajo se arribaron a las siguientes conclusiones:

- Se identificaron como elementos que definen una arquitectura de software, las decisiones significativas acerca de la organización de un sistema software, los elementos estructurales a partir de los cuales se compone un sistema y las interfaces entre ellos, su comportamiento y el estilo arquitectónico que guía esta organización. De igual forma se definen como elementos sustanciales de una arquitectura de software la escalabilidad, portabilidad, mantenibilidad, disponibilidad, extensibilidad, seguridad y rendimiento de los sistemas informáticos.
- Las tendencias y tecnologías actuales más comunes en el tema de las arquitecturas J2EE son los modelos que responden a aplicaciones web, la utilización de distintos frameworks para el manejo de las diferentes partes de estas aplicaciones, preferiblemente basadas en software libre, y como característica fundamental que puedan ser ejecutadas en contenedes ligeros, sin la necesidad de hacer uso de complejos servidores de aplicaciones.
- Las soluciones libres empresariales más comunes y utilizadas por la comunidad de desarrolladores J2EE son los frameworks, Struts, JSF, Spring, Hibernate, Acegi.
- Se identificaron como posibles mixturas de soluciones libres empresariales a través de las cuales se define una arquitectura J2EE robusta, escalable y de desarrollo rápido las siguientes: Struts + Spring + Hibernate, así como JSF+Spring+Hibernate, y en ambos casos utilizando Acegi para la gestión de la seguridad.

- Se identificaron como herramientas libres más comunes y utilizadas que contribuyen a la implementación de aplicaciones empresariales haciendo uso de dicha arquitectura las siguientes: Eclipse como entorno de desarrollo utilizando los plugins Hibernate-Tools, eclipseIde, Struts Studio, SubEclipse. Además del iReport como editor para plantillas de reportes.
- En la arquitectura aprobada para el desarrollo del sistema Servicio Autónomo se utilizaron un conjunto de frameworks: Struts para la capa de presentación, Spring para la capa de negocio, Hibernate para la capa de acceso a datos, y Acegi para gestionar la seguridad. Para esto se tomó en cuenta la ventaja que poseen dichos frameworks de tener implementados implícitamente varios patrones de diseño, lo cual brinda un alto rendimiento, optimización y eficiencia.
- El capítulo 3 constituye una guía técnica para el desarrollo de aplicaciones web sobre J2EE a partir del modelo arquitectura propuesto.

RECOMENDACIONES

Luego de haber analizado los resultados del presente trabajo de diploma, resulta factible llegar a las siguientes recomendaciones:

1. Utilizar el framework JSF para la capa de presentación. A pesar de tener ventajas y desventajas con respecto a Struts, es muy recomendable utilizar JSF para la capa de presentación dadas las facilidades y posibilidades que brinda. Además, es imposible tapar el sol con un dedo y negar o no querer ver que JSF se está convirtiendo, si no lo ha hecho ya, en un estándar.
2. Utilizar AJAX. Esto se relaciona mucho con la recomendación anterior, pues la mayor parte de los componentes de interfaz de usuario que vienen implícitos con JSF están previamente integrados con AJAX. AJAX provee interfaces visuales de usuario enriquecidas.
3. Mantener un seguimiento sistemático del estado del arte de los frameworks libres para J2EE, de manera que si en alguna oportunidad surge alguno nuevo, o se hagan modificaciones a alguno de los ya existentes, se evalúe y se inserte o cambie la arquitectura en aras de lograr un mejor producto de software.
4. Impulsar el desarrollo de aplicaciones basadas en soluciones libres como la propuesta en el presente trabajo de diploma.

BIBLIOGRAFÍA

Alur, D., J. Crupi, et al. (2003). Core J2EE™ Patterns: Best Practices and Design Strategies, Second Edition.

Amoroso, Y. M. (2004). El gobierno electrónico en el Ministerio de Justicia de la República de Cuba. Revista Jurídica, Revista Jurídica.

Ashmore, D. C. (2004). The J2EE Architect's Handbook, DVT Press.

BAUER, C. and G. KING (2005). Hibernate in Action, Manning Publications Co.

Bodoff, S., E. Armstrong, et al. (2004). The J2EE™ Tutorial Second Edition, Addison Wesley.

Buschmann, F., R. Meunier, et al. (1996). Pattern-oriented software architecture – A system of patterns, John Wiley & Sons.

Cambridge Cambridge Advanced Learner's Dictionary.

Corporation, R. S. (2003). Rational Unified Process.

Couch, J. and D. H. Steinberg (2002). Java 2 Enterprise Edition Bible, Hungry Minds.

Duc, E. V. l. (1868). Dictionnaire raisonné de L'Architecture Francaise.

Foundation, T. A. S. "The Apache Software Foundation." Retrieved 3/3/2007, from http://struts.apache.org.

Fowler, M. (2002). Patterns of Enterprise Application Architecture Addison-Wesley Professional.

G., J. C., C. DURANTE, et al. (2004). Factores estratégicos para desarrollar el gobierno electrónico en las Alcaldías de Venezuela. Revista de Ciencias Humanas y Sociales.

Husted, T., C. Dumoulin, et al. (2003). Struts in Action, Manning Publications Co.

IEEE (2000). IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.

Interface21. "Spring Framework." Retrieved 3/3/2007, from http://www.springframework.org.

Jacobson, I., G. Booch, et al. (1999). El Lenguaje Unificado de Modelado, Addison Wesley.

Jacobson, I., G. Booch, et al. (2000). El Proceso Unificado de Desarrollo de Software, Addison Wesley.

Kruchten, P. (1995). Architectural Blueprints--The 4+1 View Model of Software Architecture, IEEE Software, Institute of Electrical and Electronics Engineers.

Microsystems, S. "Java." Retrieved 1/3/2007, from http://java.Sun.com/javase/.

Microsystems, S. "Java 2 Enterprise Edition." Retrieved 3/3/2007, from http://java.Sun.com/javaee.

Microsystems, S. "JavaServer Faces Technology." Retrieved 3/3/2007, from http://java.Sun.com/j2ee/javaserverfaces.

Middleware, R. H. "Hibernate." Retrieved 3/3/2007, from http://www.hibernate.org.

Morris, W. (1999). William Morris on Art and Socialism, Dover Publications.

Peak, P. and N. Heudecker (2006). Hibernate Quickly, Manning Publications Co.

Pressman, R. S. (1998). Ingeniería de Software, un enfoque práctico. Cuarta Edición, Mc Graw Hill.

Raible, M. (2004). Spring Live, SourceBeat, LLC.

Serna, M. S. (2002). Gobierno electrónico y gobiernos locales: nuevos modelos de gestión y relación más allá de las modas, Panel: Gobierno Electrónico y Participación: factores de éxito para su desarrollo.

SourceForge. "Acegi Security System for Spring." Retrieved 3/3/2007, from http://www.acegisecurity.org.

Spielman, S. (2003). The Struts Framework: Practical Guide for Java Programmers, Morgan Kaufmann.

Vázquez, A. B. and E. R. Bombalier (2007). Modelación del Sistema del Servicio Autónomo para la gestión y control de los Registros y Notarías en la República Bolivariana de Venezuela, UCI.

Venezuela, A. N. d. l. R. B. d. (2001). Ley de Registro Público y del Notariado, Gaceta Oficial de la República Bolivariana de Venezuela.

Venezuela, A. N. d. l. R. B. d. (2004). Decreto N° 3.390, Gaceta Oficial de la República Bolivariana de Venezuela.

Venezuela, A. N. d. l. R. B. d. (2006). Ley del Registro Público y del Notariado, Gaceta Oficial de la República Bolivariana de Venezuela.

WALLS, C. and R. BREIDENBACH (2005). Spring in Action, Manning Publications Co.

GLOSARIO

Acegi: Uno de los mejores frameworks de seguridad existentes para Java. Se integra con Spring y el framework de Acegi propiamente dicho es un excelente ejemplo de extensibilidad a través de su abstracción.

AJAX: Siglas en inglés de Asinchronous Javascript And Xml (Javascript asincrónico y XML).

ALBA: Siglas de la ALternativa Bolivariana para las Américas.

AOP: Siglas en inglés de Aspect Oriented Programming (programación orientada a aspectos). Permite al desarrollador separar las tareas que no deben ser mezcladas entre módulos.

API: Siglas en inglés de Application Program Interface (programa de aplicación de interfaz). Conjunto de especificaciones de comunicación entre componentes de software. Representa un método para conseguir abstracción en la programación.

Applet: Componente de software que corre en el contexto de otro programa, por ejemplo un navegador web. El applet es una aplicación Java, por lo cual debe ser ejecutado sobre una máquina virtual.

Aplicación empresarial: Tipo de aplicación informática, generalmente web, que modela el negocio de alguna empresa en específico.

Aplicación web: Tipo de aplicación informática orientada a Internet, o redes de área local, que son accedidas desde un navegador web.

Código abierto: Open Source en inglés. Conlleva como idea más importante la posibilidad de acceder al código fuente, modificarlo y redistribuirlo como se considere conveniente, estando sujeto al marco legal que brinda el open source.

Código fuente: Sentencias o lenguaje en que está escrito un programa antes de ser compilado.

CORBA: Siglas en inglés de Common Object Request Broker Architecture (estandarte universal de la interfase términos independientes que permite la colaboración entre programas de diferentes plataformas).

DAO: Siglas en inglés de Data Access Object (objeto de acceso a datos). Se trata de un objeto que realiza una labor de interfaz entre la aplicación y los datos de la misma.

Eclipse: Entorno de desarrollo gratuito y de código abierto de Eclipse Foundation.

EJB: Siglas en inglés de Enterprise JavaBeans (beans empresariales de Java). Son una tecnología muy eficiente para el trabajo con sesiones, negocio, acceso a datos, invocación remota de métodos, etc. Pero requieren de un tipo extra de servidor para ser ejecutados y normalmente tiende a ser muy complicado su desarrollo. Actualmente solo se utilizan en aquellos proyectos que son muy grandes.

Framework: Marco de trabajo, solución reutilizable y extensible.

Gobierno Electrónico: consiste en el uso de las tecnologías de la información y el conocimiento en los procesos internos de gobierno y en la entrega de los productos y servicios del Estado tanto a los ciudadanos como a la industria. Se basa principalmente en la implantación de herramientas como portales.

Hibernate: Framework libre y de código abierto objeto relacional para el acceso a datos.

HTML: Siglas en inglés de HyperText Markup Language (lenguaje utilizado para escribir páginas web).

HTTP: Siglas en inglés de HyperText Transfer Protocol (protocolo de transmisión del hipertexto).

IDE: Siglas en inglés de Integrated Development Environment (ambiente integrado de desarrollo).

IoC: Siglas en inglés de Inversion of Control (inversión del control). Mueve la responsabilidad de realizar las tareas al interior del framework y fuera del código de la aplicación.

iReport: Herramienta libre y de código abierto empleada para la gestión de reportes con el framework JasperReports.

J2EE: Siglas en inglés de Java 2 Platform Enterprise Edition (edición empresarial de la plataforma Java 2). Comprende un conjunto de especificaciones y funcionalidades orientadas al desarrollo de aplicaciones empresariales.

JasperReports: Framework para la generación y manipulación de reportes.

Java: Tecnología, lenguaje de programación creado por Sun Microsystems.

JavaBean: Clase simple de Java.

JDBC: Siglas en inglés de Java DataBase Connectivity (conectividad de Java a bases de datos). Es un API que permite operaciones sobre bases de datos desde el lenguaje Java.

JDT: Siglas en inglés de Java Development Tool (herramienta de desarrollo de Java).

JFreeChart: Es un framework libre para la creación de gráficas dinámicas.

JSF: Siglas en inglés de JavaServer Faces. Framework desarrollado por Sun Microsystems como estándar para la vista de las aplicaciones web en Java.

JSP: Siglas en inglés de JavaServer Pages (páginas web Java del lado del servidor).

Máquina virtual: Ambiente sobre el cual se desarrollan y ejecutan aplicaciones Java.

Model 2: Aproximación a Java del patrón de diseño MVC realizada por Sun Microsystems.

MVC: Siglas en inglés de Model View Controller (patrón de diseño modelo vista controlador). Consiste en separar la interfaz de usuario, los datos y la lógica de control en tres componentes distintos.

OCDE: Organización para la Cooperación y el Desarrollo Económico.

Programación Orientada a Objetos: Paradigma de programación que define los programas en términos de “clases de objetos”, entidades que combinan estados (datos) y comportamientos (métodos y procedimientos).

RUP: Siglas en inglés de Rational Unified Process (proceso unificado de desarrollo de software).

SAREN: Acrónimo de Servicio Autónomo de Registros y Notarías, sistema informático.

Servlet: Objeto Java que se ejecuta dentro del contexto de un servidor web y que sirve para manipular peticiones y respuestas del cliente web. La mayor parte de los frameworks utilizan un servlet como núcleo.

SOAP: Siglas en inglés de Simple Object Access Protocol (protocolo simple de acceso a objetos). Es un protocolo estándar que permite que dos objetos de diferente arquitectura puedan comunicarse por intercambio de XMLs. Es muy utilizado en los servicios web.

Software: Término genérico utilizado en informática para designar programas o fragmentos de programas.

Software libre: Propone que los programas deben estar al alcance de todos de manera gratuita.

Spring: Framework de desarrollo para la plataforma Java que por su diseño ofrece muchas facilidades y libertades a los desarrolladores y se considera una alterna a la tecnología de Enterprise JavaBeans.

Struts: Framework web desarrollado por Apache Foundation para manipular las vistas en una aplicación web Java.

Tags: Etiquetas empleadas para construir páginas web, archivos XML, etc.

TIC: Siglas de Tecnologías de la Informática y las Comunicaciones.

Tomcat: Servidor web de Apache Foundation que funciona como contenedor de JSPs y servlets. Se le considera un servidor de aplicaciones.

UML: Siglas en inglés de Unified Modeling Languaje (lenguaje unificado de modelado).

Web: Red de documentos HTML intercomunicados y distribuidos entre servidores web.

Web Service: Servicio web. Se utilizan para intercambiar información entre aplicaciones de diferente arquitectura.

XML: Siglas en inglés de eXtensible Markup Languaje (lenguaje extensible de marcas).