

PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR SEDE AMBATO

SERÉIS MIS TESTIGOS

ESCUELA DE INGENIERÍA DE SISTEMAS

TEMA:

ESTUDIO DEL FRAMEWORK DE DESARROLLO SPRING CON UN ENFOQUE PRÁCTICO BASADO EN EL DESARROLLO EN JAVA DE UNA APLICACIÓN WEB PARA EVALUACIÓN DEL PERSONAL DE LA PUCESA

DISERTACIÓN DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS Y COMPUTACIÓN

AUTOR:

JAIRO ALEXANDER ROLDÁN AGUDELO

DIRECTOR:

ING. VERÓNICA PAILIACHO

PONTIFICIA UNIVERSIDAD CATOLICA DEL ECUADOR
SEDE AMBATO
BIBLIOTECA

AMBATO – ECUADOR

Abril, 2009

PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR SEDE AMBATO
ESCUELA DE INGENIERIA DE SISTEMAS

HOJA DE APROBACIÓN

TEMA:

ESTUDIO DEL FRAMEWORK DE DESARROLLO SPRING CON UN ENFOQUE PRÁCTICO BASADO EN EL DESARROLLO EN JAVA DE UNA APLICACIÓN WEB PARA EVALUACIÓN DEL PERSONAL DE LA PUCESA

AUTOR:

JAIRO ALEXANDER ROLDAN AGUDELO

Verónica Pailiacho, Ing.
DIRECTOR DE LA DISERTACIÓN

F _____

Galo López, Ing.
CALIFICADOR

F _____

Andrés López, Ing.
CALIFICADOR

F _____

Santiago Acurio, Ing.
DIRECTOR DE LA ESCUELA DE SISTEMAS

F _____

Pablo Poveda, Ab.
SECRETARIO GENERAL PROCURADOR

F _____

SECRETARIA GE
PROCURADU

DECLARACIÓN DE AUTENTICIDAD Y RESPONSABILIDAD

Yo, Jairo Alexander Roldán Agudelo portador de la cédula de ciudadanía No. 0400830634 declaro que los resultados obtenidos en la investigación que presento como informe final, previo la obtención del título de Ingeniero en Sistemas son absolutamente originales, auténticos y personales.

En tal virtud, declaro que el contenido, las conclusiones y los efectos legales y académicos que se desprenden del trabajo propuesto de investigación y luego de la redacción de este documento son y serán de mi sola y exclusiva responsabilidad legal y académica.



Jairo Alexander Roldán Agudelo

CI. 0400830634

AGRADECIMIENTO

Remarco una mención especial a mi gran amigo Roberto Peña, quien con su gran conocimiento y habilidad técnica ha sido la persona que me ha apoyado incondicionalmente en mi vida académica y profesional.

Agradezco inmensamente a mi colega Marco Toscano, el contar con su aporte hizo posible la realización de este proyecto.

También un profundo agradecimiento a los profesores, directivos, y compañeros de la Pontificia Universidad Católica del Ecuador Sede Ambato, ya que sin su colaboración este trabajo no hubiera podido concluirse.

DEDICATORIA

Para mi querida madre Ofelia y para mis hermanos Carla y Junior. El contar con su amor y compañia es algo más que le debo a mi gran DIOS.

RESUMEN

Actualmente, los programadores de software tienen que desarrollar complejas aplicaciones empresariales que faciliten la reutilización y puedan crecer conforme a las necesidades del negocio. Razón por la cual, es muy importante seleccionar frameworks como Spring que permitan a los desarrolladores construir este tipo de aplicaciones.

Spring es un framework de código abierto, creado para solventar el desarrollo de aplicaciones empresariales. Además, Spring es un framework completo que maneja todos los aspectos concernientes al desarrollo de este tipo de aplicaciones. Con Spring se desarrolló un Sistema para evaluación del Personal de la PUCESA que reúne muchas de las características del desarrollo de aplicaciones empresariales.

Este sistema provee un sólido modelo de programación, reutilización de código y buenas prácticas de programación, como la programación de interfaces en lugar de la programación de clases. Este framework también facilita el Diseño Orientado a Objetos, el intercambio y la disminución de la dependencia de código entre objetos. El Sistema Evaluador del Personal de la PUCESA, desarrollado con Spring, provee un Mapeo Objeto-Relacional (ORM, por sus siglas en Inglés) que permite trabajar con cualquier Sistema de Base de Datos sin la necesidad de utilizar lenguaje SQL. El sistema

Evaluador del Personal Administrativo de la PUCESA también tiene una estructura de múltiples capas (basado en Spring) que permite la administración de configuración de software.

En conclusión, El sistema Evaluador del Personal Administrativo de la PUCESA es una aplicación empresarial desarrollada con Spring, este sistema es escalable y su código puede ser reutilizado y configurado fácilmente.

ABSTRACT

Nowadays, software programmers need to develop complex enterprise applications that grow with business necessities. Consequently, it is extremely essential to look for frameworks such as Spring that support developers to build these software programs.

Spring is an open source framework, created to address the complexity of enterprise application development. Furthermore, Spring is an extended framework that holds all aspects about enterprise application development. With Spring, a Staff Evaluator Application System for the PUCESA, that compiles several enterprise development features, can be developed.

This application provides a consistent programming model, code reuse and good programming practice, such as programming to interfaces, rather than classes. This framework also aims to facilitate Object Oriented design, pluggability and loosely coupled application code. The Staff Evaluator Application System for the PUCESA, developed with Spring, provides an Object-relational mapping (ORM) that allows working with any Data Base System without using a SQL language. The Staff Evaluator Application System for the PUCESA also has a multiple-layers structure (based in Spring) that allows software configuration management.

In conclusion, the Staff Evaluator Application System for the PUCESA system is an enterprise application developed with Spring framework. This application is scalable and its code could be reused and configured easily.

PONTIFICIA UNIVERSIDAD CATOLICA DEL ECUADOR
SEDE AMBATO
BIBLIOTECA

TABLA DE CONTENIDOS

CAPITULO I 1

PROYECTO DE LA INVESTIGACIÓN 1

1.1 INTRODUCCIÓN 1
1.2 PLANTEAMIENTO DEL PROBLEMA 3
1.3 PROBLEMATIZACIÓN 4
1.4 DELIMITACIÓN 4
1.5 JUSTIFICACIÓN 8
1.5.1 Técnica 8
1.5.2 Económica 9
1.5.3 Académica 9
1.6 OBJETIVOS 10
1.6.1 Objetivo general 10
1.6.2 Objetivos específicos 10
1.7 HIPÓTESIS 11

CAPITULO II 12

MARCO TEÓRICO 12

2.1 INTRODUCCIÓN AL FRAMEWORK DE DESARROLLO SPRING 12
2.1.1 El inicio de las aplicaciones empresariales 12
2.1.2 Definición de Spring 13
2.1.3 La solución los problemas con Spring 15
2.1.4 La estructura de Spring 17
2.2 CONCEPTOS FUNDAMENTALES DEL FRAMEWORK SPRING 19
2.2.1 Los Módulos de Spring 19
2.2.1.1 El núcleo contenedor (Core) 20
2.2.1.2 Módulo del contexto de la aplicación (Context) 21
2.2.1.3 Modulo AOP 21
2.2.1.4 Modulo DAO y abstracción JDBC 21
2.2.1.5 Modulo ORM 22
2.2.1.6 Administración de extensiones Java (JMX) 22
2.2.1.7 Conector Java EE API (JCA) 23
2.2.1.8 El framework MVC de Spring 23
2.2.1.9 MVC Portable 24
2.2.1.10 Módulo Web de Spring 24
2.2.1.11 Acceso Remoto (Remoting) 24
2.2.1.12 Servicios de Mensajes Java (JMS) 25

2.2.2 Inyección de dependencia (DI) ... 25
2.2.2.1 Introducción ... 25
2.2.2.2 Análisis de la inyección de dependencia ... 29
2.2.3 Programación orientada a aspectos (AOP) ... 31
2.2.3.1 Introducción ... 31
2.2.3.2 Análisis de la programación orientadas a aspectos ... 33
2.2.4 El contenedor de Spring y el empleo de beans ... 37
2.2.4.1 Introducción al Bean Factory ... 38
2.2.4.2 Introducción al application context ... 40
2.2.4.3 Relacionando beans ... 42
2.2.4.4 Ámbito de los beans en Spring ... 47
2.3 TRABAJANDO CON BASE DE DATOS EN SPRING ... 48
2.3.1 El DAO de Spring ... 49
2.3.1.1 Data soucers y drivers de Spring ... 50
2.3.2 Integrando Hibernate con Spring ... 52
2.3.2.1 Plantillas de Hibernate ... 54
2.3.3 Administración de transacciones ... 57
2.3.4 Los manejadores de transacciones de Spring ... 58
2.3.5 Transacciones de Hibernate ... 59
2.4 EL FRAMEWORK WEB DE SPRING ... 60
2.4.1 Introducción a Spring MVC ... 60
2.4.2 Configuración del DispatcherServlet de Spring ... 62
2.4.3 Configurando el controlador de Spring ... 64
2.4.4 Configuración de la vista resultante ... 66

CAPITULO III ... 69

DESARROLLO DE LA APLICACIÓN ... 69

3.1 METODOLOGÍA DE DESARROLLO ... 69
3.1.1 Programación orientada a objetos y aspectos ... 69
3.1.2 Paradigma de desarrollo ... 70
3.2 ETAPA DE ANÁLISIS ... 72
3.2.1 Levantamiento de requerimientos ... 72
3.2.2 Especificaciones funcionales ... 73
3.2.2.1 Registro de cronograma ... 74
3.2.2.2 Formatos de evaluación ... 74
3.2.2.3 Formato de observación ... 75
3.2.2.4 Formatos de prácticas ... 76
3.2.2.5 Encuestas ... 76
3.2.2.6 Proceso de cuantificación ... 77
3.2.2.7 Parametrización de entidades generales ... 77

3.2.2.8 Usuarios del sistema 78
3.2.2.9 Reportes y estadísticas 78
3.2.3 Selección de las arquitecturas y plataformas de desarrollo 79
3.2.3.1 Framework de desarrollo empresarial 79
3.2.3.2 Lenguaje de programación 81
3.2.3.3 Framework de presentación 82
3.2.3.4 Herramienta ORM 83
3.2.3.5 Motor de base de datos 84
3.2.3.6 Herramientas adicionales 85
3.3 ETAPA DE DISEÑO 85
3.3.1 Estructura en capas del proyecto 85
3.3.2 Estructura de los paquetes 88
3.3.3 Diagrama de clases 89
3.3.4 Diagrama físico de datos 98
3.3.5 Definición de los patrones de diseño 104
3.3.5.1 Modelo Vista Controlador 104
3.3.5.2 Inversión de control 104
3.3.5.3 Singleton 105
3.3.5.4 Método de plantilla 106
3.3.6 Diseño de entradas 107
3.3.7 Diseño de Salidas 109
3.4 ETAPA DE CODIFICACIÓN 110
3.4.1 Configuración del proyecto Web 110
3.4.2 Configuración de ZK 115
3.4.3 Configuración de Spring 117
3.4.4 Clases de negocio 123
3.4.5 Etapa de mantenimiento 127

CAPITULO IV 130

DESARROLLO DE LA APLICACIÓN 130

4.1 VERIFICACIÓN DE LA HIPÓTESIS 130
4.2 VALIDACIÓN DE LOS RESULTADOS 131
4.3 CONCLUSIONES 132
4.4 RECOMENDACIONES 133

BIBLIOGRAFÍA 135

GLOSARIO 136

ANEXOS 138

TABLA DE GRÁFICOS

GRÁFICOS

Figura 2.1 Módulos que integran Spring 20
Figura 2.2 Con DI las dependencias son dadas a los objetos. 30
Figura 2.3 Las capas de servicios conseguidas con AOP. 33
Figura 2.4 Un cantor como aspecto orientado cubre al caballero. 35
Figura 2.5 Capa de acceso a datos basada en DAO 49
Figura 2.6 LocalSessionFactoryBean de Spring [1] 55
Figura 2.7 Relación de los manejadores de transacciones de Spring. 59
Figura 2.8 Solicitud procesada por *DispatcherServlet* 62
Figura 3.1 Etapas del paradigma utilizado 70
Figura 3.2 Estructura de las capas de la aplicación 86
Figura 3.3 Estructura de paquetes 88
Figura 3.4 Diagrama de clases 90
Figura 3.5 Diagrama de la clase departamento 91
Figura 3.6 Diagrama de la clase departamento 91
Figura 3.7 Jerarquía de las clases de entidad 92
Figura 3.8 Diagrama de la clase tipo formato 92
Figura 3.9 Diagrama de la clase formato 93
Figura 3.10 Diagrama de la clase factor 93
Figura 3.11 Diagrama de la clase de subfactor 94
Figura 3.12 Diagrama de la clase Nivel de evaluación 94
Figura 3.13 Diagrama de la jerarquía de las clases de evaluación 95
Figura 3.14 Diagrama de la clase actividad 96
Figura 3.15 Diagrama de la interfaz de persistencia 96
Figura 3.16 Diagrama de la clase de implementación de persistencia 97
Figura 3.17 Diagrama de una clase vista de ZK y sus relaciones 97
Figura 3.18 Modelo Físico de Datos 98
Figura 3.19 Diseño de la GUI de plantilla de la aplicación 108
Figura 3.20 Diseño básico de una GUI de mantenimiento 108

Figura 3.21 Diseño básico de una GUI de mantenimiento 109
Figura 3.22 Estructura de un proyecto Web de Java 110
Figura 3.23 Archivos de configuración de ZK 115

TABLAS

Tabla 2.1 Implementaciones de Resource. 40
Tabla 2.2 Los manejadores de transacciones de Spring 58
Tabla 3.1 Tabla comparativa de los lenguajes soportados por Spring 81
Tabla 3.2 Diccionario de datos de la tabla Departamento 99
Tabla 3.3 Diccionario de datos de la tabla Cargo 99
Tabla 3.4 Diccionario de datos de la tabla Entidad 99
Tabla 3.5 Diccionario de datos de la tabla Empleado 100
Tabla 3.6 Diccionario de datos de la tabla Usuario 100
Tabla 3.7 Diccionario de datos de la tabla TipoFormato 100
Tabla 3.8 Diccionario de datos de la tabla Formato 101
Tabla 3.9 Diccionario de datos de la tabla Factor 101
Tabla 3.10 Diccionario de datos de la tabla Subfactor 101
Tabla 3.11 Diccionario de datos de la tabla NivelesEvaluacion 102
Tabla 3.12 Diccionario de datos de la tabla Evaluacion 102
Tabla 3.13 Diccionario de datos de la tabla DetalleEvaluacion 103
Tabla 3.14 Diccionario de datos de la tabla Actividad 103

REGLAS Y CONVENCIONES DE LECTURA

A continuación se mencionan las reglas y convenciones que facilitan la lectura de este documento.

Idioma

Algunas palabras de este documento se encuentran en idioma Inglés, debido simplemente a que algunos términos, al tratarse de referencias tecnológicas, no tienen una traducción propia al español. Adicionalmente, para la simplificación de escritura se utilizan abreviaturas para algunos términos técnicos en su idioma original (normalmente Inglés). Sin embargo, antes de emplear abreviaturas se describe el significado de las mismas. Para mayor información del significado de las abreviaturas, se puede revisar el glosario al final de este documento.

Código de programación

Los códigos de programación o los nombres que hacen referencia a fragmentos u objetos de programación se encuentran diferenciados con letra cursiva. A continuación se ilustra el formato empleado:

Import com.pucesa.nomenclatura.programacion;
Int i;
BeanFactory

Citas Bibliográficas

Un número dentro de corchetes ubicado en un párrafo, tabla o figura se emplea en este documento para remarcar que pertenece o se basa en un libro específico de la bibliográfica, Por ejemplo: [1] hace referencia al libro: "Spring in Action" de Craig Walls.

CAPITULO I

PROYECTO DE LA INVESTIGACIÓN

1.1 Introducción

El desarrollo de sistemas informáticos de gran tamaño es un proceso complejo y la construcción de aplicaciones empresariales efectivas es una actividad aun más complicada. Por lo tanto, es común hoy en día encontrar a una gran cantidad de desarrolladores haciendo frente a un sinnúmero de problemas concernientes al desarrollo de aplicaciones empresariales sostenibles y escalables.

La exigencia que tienen las aplicaciones empresariales de cumplir con necesidades como: compartir datos entre múltiples usuarios, escalar de acuerdo a las necesidades corporativas y facilitar la reutilización; impulsa a la búsqueda y surgimiento de tecnologías de desarrollo que satisfagan las necesidades y hagan frente a los problemas.

En la actualidad, se han difundido algunas tecnologías y especificaciones que facilitan la construcción de arquitecturas sólidas enmarcadas dentro del desarrollo de soluciones empresariales y que dictan las pautas a emplearse en la construcción de aplicaciones. Un ejemplo de esto son las especificaciones J2EE.

Sin embargo, no todas las tecnologías desarrolladas; inclusive algunas de ellas consideradas estándares; han alcanzado el éxito en su uso y aplicación, esto puede ser debido al hecho de que son difíciles de implementar, exigen una extensa curva de aprendizaje para los desarrolladores, obligan al uso de conceptos de diseño y análisis extremadamente difíciles de conseguir en ambientes reales de producción y además de generar soluciones empresariales pesadas.

Basado en todo lo anterior, es importante remarcar la necesidad de emplear arquitecturas o tecnologías que faciliten el desarrollo de aplicaciones empresariales sólidas, productivas y escalables. Además, estas arquitecturas deben ser: ampliamente difundidas, livianas, fáciles de utilizar y confiables. Dentro de este contexto se engloba a la arquitectura de desarrollo denominada Spring.

Spring es una tecnología o más comúnmente llamado framework que fue creado para disminuir la complejidad del desarrollo de aplicaciones empresariales y lograr soluciones escalables con baja sobrecarga. Además, Spring aplica una serie de conceptos que son reconocidos dentro del desarrollo de sistemas orientados a la reutilización, la escalabilidad y el mantenimiento.

Adicionalmente, y dentro del marco de la evaluación que es realizada actualmente al personal de la PUCESA, se presentan una serie de problemas que responden al hecho de que este proceso de evaluación es

realizado manualmente, dentro de este contexto se identifican los siguientes problemas: la gran cantidad de tiempo y recursos que se emplean, y la pérdida de información que puede ocasionarse al llevar registros manuales. Por lo tanto, es importante para la Universidad contar con una aplicación Web que solucione estos inconvenientes.

1.2 Planteamiento del problema

La construcción de una aplicación Web para evaluación del personal de la PUCESA, que reúna las características de escalabilidad, reutilización y rendimiento que las aplicaciones empresariales deben incluir hoy en día, y que solo pueden ser conseguidas con el empleo de las apropiadas tecnologías de desarrollo, como el framework Spring.

La disminución de la complejidad que el desarrollo de aplicaciones empresariales involucra, basados en el uso de adecuadas arquitecturas de desarrollo prefabricadas y fáciles de implementar como Spring.

Dentro del ámbito de la Universidad, la necesidad de cubrir con dos aspectos de contextos diferentes; el primero, la automatización de un proceso de evaluación del personal de la Universidad; y el segundo, sentar las bases académicas para profesores y alumnos que quieran incursionar en el desarrollo empresarial de alto nivel con el uso de Spring.

1.3 Problematización

Se puede definir como problema a la necesidad de estudiar el framework de desarrollo Spring, con el objetivo de realizar la construcción de una aplicación Web para evaluación del personal de la PUCESA que reúna las características de escalabilidad, reutilización y baja complejidad de desarrollo, características que son inherentes al empleo de este framework.

1.4 Delimitación

La investigación que se realizará del framework Spring comprenderá únicamente los temas concernientes al desarrollo de una aplicación Web para evaluación del personal del Área de Secretaría de la PUCESA. Se detallan a continuación los temas específicos que se abordarán en esta investigación:

Los conceptos básicos de Spring.

El framework Web de Spring.

El framework JDBC de Spring y la integración con Hibernate.

La administración de transacciones con Spring.

Este sistema será desarrollado utilizando el lenguaje de programación de código abierto Java y utilizará los conceptos definidos en el framework Spring. A continuación se detalla el ámbito de la aplicación Web para evaluación del personal de la PUCESA:

Área de implantación: El sistema se utilizará únicamente para la evaluación del personal del Área de Secretaría de la PUCESA.

Actividades de evaluación: El sistema se utilizará para realizar los formatos de evaluación de observación y pruebas. Adicionalmente, el sistema registrará las encuestas de clima laboral y las encuestas de satisfacción.

Administración del sistema: Los procesos de administración y configuración funcional del sistema serán realizados exclusivamente por el personal del Departamento Administrativo de la PUCESA. Es este departamento únicamente el encargado de definir, parametrizar y configurar los formatos de evaluación y todo el tipo de encuestas a ser utilizadas dentro de la aplicación. La actividad de esta área también se extiende al registro de los formatos de evaluación, la generación de reportes y estadísticas, y toda actividad concerniente a la administración.

Encuestas al personal: Las encuestas del clima laboral y las encuestas de satisfacción deben ser registradas directamente en el sistema por el personal que está siendo evaluado, que para el ámbito de este proyecto es el personal del Área de Secretaría.

Proceso de cuantificación: El sistema realizará automáticamente los procesos de cuantificación de la evaluación realizada sobre el personal, este proceso es realizado independientemente para cada persona evaluada,

basado en la información registrada en los formatos de evaluación o encuestas, para este proceso es necesario que en el sistema se registren los niveles de calificación, los niveles de calificación pueden tener diferentes valores para cada tipo de encuesta o formato de evaluación. Por ejemplo: para un caso estos valores pueden incluirse entre un rango de uno a cinco, paro otro caso los valores pueden incluirse entre un rango de uno a cien. Sin embargo, para el proceso de cuantificación del sistema, estos niveles de calificación deben ser solamente numéricos.

Registro de cronogramas: El sistema permitirá tener un registro de las actividades referentes a la evaluación del personal. Dentro de este contexto es necesario registrar: la hora y fecha de cuando la actividad o actividades serán realizadas, el personal a ser evaluado, quien realizará la evaluación, en qué lugar se realizará la evaluación, y que tipo de evaluación se realizará (observación, práctica, clima laboral, encuesta de satisfacción).

Gestión de información: Con el propósito de que la evaluación del personal sea realizada, es necesario que el sistema permita la gestión de algunas entidades. Se entiende por gestión a las actividades como: inserción, modificación, consulta y eliminación. Se detallan las entidades que deben ser gestionadas:

Personal del Área de Secretaría.

Cargos de la Universidad.

Áreas de la Universidad.

Factores, subfactores de evaluación y preguntas.

Tipos de formatos de evaluación y encuestas.

Tipos de contrato

Reportes y estadísticas: El sistema debe permitir la generación de reportes y estadísticas que visualicen los resultados obtenidos en las evaluaciones. Dentro de los reportes necesarios se incluyen:

Reporte de evaluación de las observaciones, incluye gráficos de barra por subfactores.

Reporte de evaluación de las pruebas (prácticas), incluye gráficos de barra por subfactores.

Reporte de los cronogramas agrupados por fecha.

El presente proyecto de disertación de grado será realizado en el periodo de marzo de 2008 a Julio de 2008.

Con el objetivo de instalar la aplicación Web para la evaluación del personal del Área de Secretaría de la PUCESA, se donará al Centro de Computo de la Universidad una computadora que incluirá solamente el case. Las características de esta computadora son:

Mainboard INTEL DG33BU S775,1333GHZ,DDR2,V,S,R

Procesador INTEL CORE 2 DUO E6550 2.33GHZ

Memoria RAM 1GB PC-667

Disco Duro de 160GB SAMSUNG SATA 7200RPM

DVD-RWRITER SAMSUNG

1.5 Justificación

1.5.1 Técnica

En los últimos días Spring se ha convertido en uno de los frameworks de mayor difusión y utilización por parte de los desarrolladores. Spring disminuye significativamente la complejidad en la construcción de aplicaciones empresariales escalables.

Spring es un framework de código abierto que provee a los programadores un conjunto de características, las mismas que se detallan a continuación:

Provee conceptos de programación que son utilizables en cualquier ambiente.

Facilita la reutilización de código.

Soluciona los problemas que actualmente se presentan en aplicaciones J2EE.

Promueve el uso de buenas prácticas de programación, como el uso de interfaces y el desarrollo en múltiples capas.

Permite la selección de cualquier arquitectura.

Adicionalmente, el desarrollo de la aplicación Web para evaluación del personal de la PUCESA con Spring, permitirá a la Universidad obtener los siguientes beneficios:

Disminuir el tiempo utilizado en la evaluación del personal de la PUCESA.

Disminuir los recursos empleados en el proceso de evaluación del personal de la PUCESA.

Evitar la pérdida de información concerniente al proceso de evaluación del personal de la PUCESA, proceso que actualmente es realizado manualmente.

1.5.2 Económica

Spring es un framework gratuito que puede ser descargado de Internet. Además, Spring se puede integrar con una gran cantidad de otras arquitecturas complementarias y herramientas. La elección de estas arquitecturas es totalmente libre en Spring posibilitando la elección de tecnologías abiertas y libres.

1.5.3 Académica

El desarrollo con Spring de la aplicación Web para evaluación del personal de la PUCESA, dejará una base académica para nuevas investigaciones que quieran incursionar en el ámbito del desarrollo de aplicaciones empresariales con Spring. Esta base podrá ser de utilidad tanto a los profesores como a los alumnos de la Universidad.

1.6 Objetivos

1.6.1 Objetivo general

Estudiar el framework de desarrollo Spring y su enfoque práctico basado en el desarrollo de una aplicación Web para evaluación del personal del Área de Secretaría de la PUCESA.

1.6.2 Objetivos específicos

Implementar una aplicación Web para evaluación del personal de la PUCESA que incluya las características inherentes a las soluciones creadas con el framework Spring: escalabilidad, reutilización y baja complejidad de desarrollo.

Automatizar con la aplicación Web el proceso de evaluación del personal del Área de Secretaría de la PUCESA.

Sentar las bases académicas en la PUCESA para profesores y alumnos que quieran incursionar en el desarrollo empresarial de alto nivel con Spring.

1.7 Hipótesis

El estudio del framework de desarrollo Spring y su empleo práctico, permite la construcción de una aplicación Web para evaluación del personal de la PUCESA que sea escalable, reutilizable y que no exija complicados procesos de desarrollo.

CAPITULO II

MARCO TEÓRICO

2.1 Introducción al framework de desarrollo Spring

2.1.1 El inicio de las aplicaciones empresariales

En el año de 1998, no hacía mucho tiempo del que Java se empezó a utilizar para la construcción de componentes mediante la especificación JavaBeans, Sun publicó la versión 1.0 de la especificación Enterprise JavaBeans (EJB, por sus siglas en inglés). Esta especificación estaba orientada a la construcción de aplicaciones que ofrecían servicios como transacción soporte, seguridad y servicios computacionales distribuidos. Esta especificación también extendió la noción de componentes a lado del servidor, proveyendo lo que se conoce como servicios empresariales. Sin embargo, esta especificación no pudo continuar con la simplicidad que originalmente tenía la especificación JavaBeans. [2]

A pesar del hecho de que muchas aplicaciones exitosas han sido desarrolladas con EJB, esta tecnología nunca alcanzó el propósito inicial que tenía, el cual era simplificar el desarrollo de aplicaciones empresariales. EJB complica el desarrollo por el uso de descriptores y código enmarañado. Eventualmente, muchos desarrolladores han desistido de utilizar EJB, como

resultado su popularidad ha decrecido considerablemente en los últimos años, animando a los desarrolladores en la búsqueda de arquitecturas más fáciles.

Actualmente, el desarrollo de componentes Java está utilizando nuevas técnicas de programación, incluyendo programación orientada a aspectos (AOP, por sus siglas en inglés) e inyección de dependencia (DI, por sus siglas en inglés). Además, estas nuevas técnicas deben estar basadas en la potencialidad de JavaBeans y en los simples y viejos objetos de java (POJOs, por sus siglas en inglés). Al igual que EJB, estas nuevas tecnologías utilizan un modelo de programación declarativa sin toda la complejidad que EJB involucra.

Estas nuevas tecnologías o frameworks basados en el uso de POJOs, dentro de los cuales se incluye Spring, se han establecido por sí mismos como facto estándares (oficialmente no reconocidos, pero ampliamente difundidos y utilizados) en la comunidad de Java, permitiendo sin complejidad la construcción de aplicaciones livianas y escalables.

2.1.2 Definición de Spring

Spring es un framework de tecnología abierta (open source, por su término en inglés), fue creado para disminuir la complejidad del desarrollo de aplicaciones empresariales. Spring hace posible usar los JavaBeans para lograr cosas que solo eran posibles con el uso de los EJBs. Sin embargo, el

ámbito del empleo de Spring no está limitado al desarrollo en el lado del servidor. Cualquier aplicación Java puede beneficiarse de Spring en términos de simplicidad, testeo, y bajo acoplamiento (loose coupling, por su término en inglés). [2]

A resumidas cuentas, Spring es un framework contenedor y liviano que incluye inyección de dependencia y programación orientada a aspectos. Sin embargo, para facilitar el entendimiento de Spring a continuación se detalla esa descripción: [1]

Framework: Spring hace posible configurar y construir aplicaciones complicadas desde simples componentes ya prefabricados. En Spring, los objetos de la aplicación son construidos declarativamente, normalmente en archivos XML (Lenguaje de Marcado Extendido). Spring también provee arquitectura funcional prefabricada (administración de transacciones, integración con frameworks de persistencia, etc.) dejando que el desarrollador se enfoque en la programación de la lógica de la aplicación.

Contenedor: Spring es un contenedor en el sentido que contiene y administra el ciclo de vida y configuración de los objetos de la aplicación. En Spring, se puede definir como cada objeto de la aplicación puede ser declarado, como ellos pueden ser configurados y como ellos deberían estar relacionados el uno con el otro.

Liviano: Spring es liviano en términos de tamaño y sobrecarga. El framework de Spring es distribuido en un simple archivo JAR que tiene un tamaño de

aproximadamente 2,5 MB. Y la sobrecarga de procesos requerida por Spring es muy baja.

Inyección de dependencia: Spring promueve el bajo acoplamiento a través de la técnica conocida como inyección de dependencia (DI, por sus siglas en inglés). Cuando DI es aplicado, los objetos están dando pasivamente sus dependencias en lugar de crear o buscar objetos dependientes por ellos mismos.

Programación orientada a aspectos: Spring incluye un amplio soporte para la programación orientada a aspectos (AOP, por sus siglas en inglés) que permite el desarrollo cohesivo, separando las aplicaciones lógicas de negocios de los sistemas de servicios (como por ejemplo las transacciones administrativas). Los objetos de la aplicación hacen lo que se supone deben hacer (realizar la lógica de negocios) y nada más. Estos objetos no son responsables de otros ámbitos del sistema, como son procesos transacciones y autorizaciones.

2.1.3 La solución los problemas con Spring

En esta sección se describe como Spring soluciona los problemas típicos de la construcción de aplicaciones empresariales:

Ambientes soportados: Spring puede ser utilizado en un amplio rango de ambientes, no solo incluye la construcción de aplicaciones del lado del servidor. El propósito inicial de Spring era el ambiente J2EE y Spring ofrece muchos servicios valiosos para utilizar en aplicaciones J2EE. [2]

Reutilización de código: Spring promueve la reutilización de código, Spring ayuda a evitar la necesidad de escoger al inicio de un desarrollo los servicios a utilizarse, por ejemplo, si se va utilizar JNDI o JTA (API para transacciones de Java). Las abstracciones de Spring permiten desarrollar código en diferentes ambientes y adaptarse a cualquiera de ellos si es necesario. Spring permite seleccionar las plataformas de desarrollo que presentan los mejores beneficios, como la compra de un servidor de aplicaciones que se ajuste a los requerimientos de escalabilidad de la aplicación. [1]

Solución a los problemas de diseño de J2EE: Muchas aplicaciones J2EE no son aplicaciones orientadas a objetos, Spring facilita superar algunos de los impedimentos con relación a este tema, permitiendo escribir código mejor estructurado (basado en patrones de diseño), más cohesivo y reutilizable. [2]

Fomentando buenas prácticas de programación: Spring alcanza buenas prácticas de programación, empleando interfaces, mas que clases, Spring reduce la complejidad del código a interfaces, simplemente realizando la implementación requerida de una interfaz a través de una clase. Las implementaciones pueden ser cambiadas de acuerdo a los cambios que la aplicación involucre. [1]

Facilitando las pruebas: Spring es diseñado para que las aplicaciones que son construidas con Spring sean fáciles de testear, Spring utiliza POJOs y estos son fáciles de testear; la dependencia sobre las APIs de Spring normalmente están representadas por interfaces que son fáciles de examinar. A diferencia de JNDI o Struts (ambas tecnologías parte de las especificaciones J2EE) que obligan a heredar de clases que por sí mismas involucran complicadas dependencias. [1]

Seleccionando las arquitecturas deseadas: Spring provee una arquitectura de núcleo, permitiendo reemplazar cada capa. Por ejemplo, con una capa media de Spring, se podría intercambiar un framework de mapeo Objeto/Entidad (ORM, por sus siglas en inglés) con otro framework con un mínimo impacto en el código de la lógica de negocio, o también, cambiar Struts a Spring MVC o WebWork con ningún impacto en la capa media de Spring. [1]

2.1.4 La estructura de Spring

Spring es el más popular y ambicioso framework, es el único que comprende todas las capas arquitecturales de las aplicaciones J2EE y el único que ofrece una amplia gama de servicios. A continuación se detallas los módulos más importantes de Spring:

Contenedor de Inversión de Control: el núcleo de Spring provee una administración configurable de POJOs. El contenedor de Inversión de Control de Spring puede administrar POJOs y trabajar con otras partes de Spring para ofrecer servicios. [2]

Framework con Programación Orientada a Aspectos: Spring utiliza Programación Orientada a Aspectos para entregar importantes servicios como declaraciones de transacciones administrativas. La Programación Orientada a Aspectos de Spring puede ser usada para implementar código personalizado que de otra forma estaría esparcido entre clases de aplicación o métodos estructurales. [2]

Abstracción para el acceso a datos: Spring proporciona un acceso arquitectural consistente a los datos y garantiza una poderosa abstracción al implementarla. Spring provee una extensa jerarquía de excepciones para acceso a datos independientemente del producto de persistencia que se utilice. También provee una amplia gama de servicios para manejar APIs de persistencia, permitiendo a los desarrolladores escribir interfaces para acceso a datos e implementaras con las herramientas de su elección. [1]

Simplificación a JDBC: Spring proporciona una capa de abstracción sobre JDBC (Java DataBase Controller) que es significativamente más simple y menos problemática de utilizar que JDBC cuando usa SQL para acceso a base de datos relacionales. [1]

Administrador de transacciones: Spring provee una abstracción transaccional que puede situarse sobre transacciones globales JTA o transacciones locales que utilizan JDBC, Hibernate, JDO, o cualquier otro API de persistencia. Esta abstracción aporta un consistente modelo de programación en un amplio rango de ambientes. [2]

Web Framework MVC: Spring proporciona un framework Web basado en MVC (Model View Controller). Al utilizar instancias compartidas de multihilos "controladores" se asemeja al proceso de Struts, pero el framework de Spring es mucho más flexible, y se integra con el contenedor de inversión de control. Todas las otras características de Spring pueden ser usadas con otros frameworks Web como Struts o JSF (Java Server Faces). [1]

Simplificación para trabajar con JNDI, JTA, y otros APIs J2EE: Spring ayuda a disminuir la cantidad de código innecesario. Con Spring se puede seguir utilizando JNDI o EJB, pero no es necesario escribir otros JNDI para

recuperar los recursos. Spring garantiza que los recursos como los contextos JNDI son siempre cerrados, incluso si se ha producido una excepción. La ganancia es que el desarrollador se enfoca en escribir el código que realmente necesita, el código que se orienta a la lógica de negocio. [1]

Acceso remoto liviano: Spring proporciona soporte para POJOs remotos basado en un rango de protocolos, que incluye RMI, IIOP, Hessian, Burlap, HTTP Invoker, y otros protocolos basados en servicios Web. [1]

Soporte JMS: Spring provee soporte para enviar y recibir mensajes JMS (Java Message Service) de una manera mucho más fácil que la proporcionada por J2EE. [1]

Soporte JMX: Spring soporta administración y configuración de objetos de aplicación JMX (Java Management Extensions). [1]

Spring facilita la implementación de una estrategia para pruebas de aplicaciones: Spring no solo ayuda a facilitar un buen diseño, permite efectivas pruebas unitarias. Además, proporciona una solución para la integración de pruebas de una aplicación que se encuentra en el lado del servidor. [2]

2.2 Conceptos fundamentales del framework Spring

2.2.1 Los Módulos de Spring

Spring está comprendido por un conjunto de módulos bien definidos, (ver Figura 1 [1]). La integración de estos módulos provee todo lo que se necesita para crear aplicaciones empresariales listas para funcionar. Sin embargo, no

es necesario que una aplicación utilice en su totalidad los módulos de Spring. Con Spring se es libre de escoger que módulos se van a emplear, inclusive se puede usar módulos de otros frameworks y librerías de terceros.

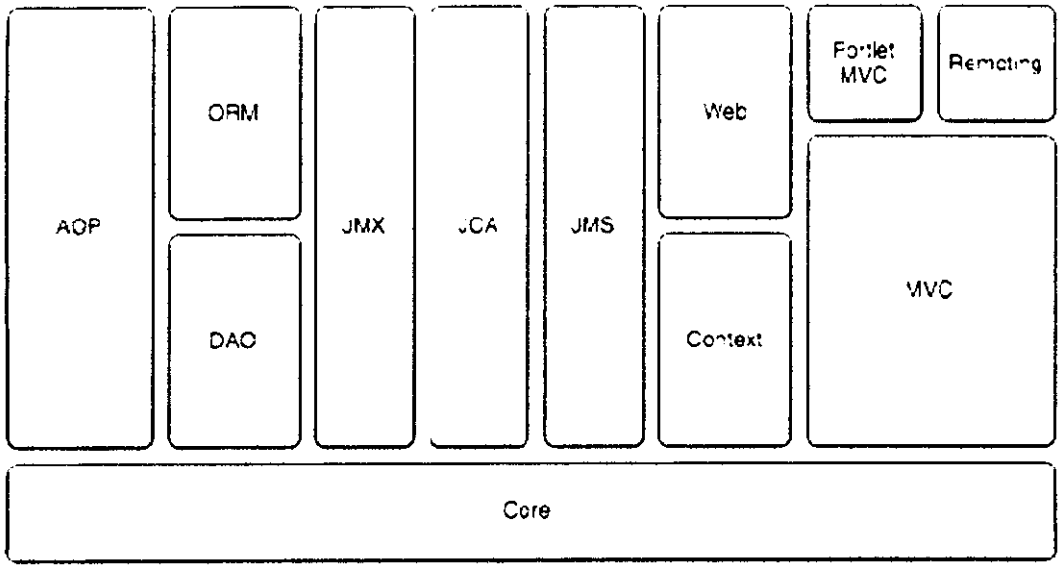


Figura 2.1 Módulos que integran Spring

Todos los módulos de Spring están construidos basados en el contenedor Core. El contenedor define como los beans son creados, configurados y administrados.

2.2.1.1 El núcleo contenedor (Core)

El núcleo contenedor provee la funcionalidad fundamental del framework Spring. Este modulo contiene el *BeanFactory* que es el contenedor fundamental de Spring y la base sobre la cual la Inyección de Dependencia está basada. [1]

2.2.1.2 Módulo del contexto de la aplicación (Context)

El contexto de las aplicaciones de Spring está construido sobre el núcleo contenedor. El *BeanFactory* hace de Spring un contenedor, pero el módulo de contexto es el que lo hace un framework. Este módulo extiende el concepto de *BeanFactory*, añadiendo soporte para mensajes de internacionalización, eventos del ciclo de vida de la aplicación y validación. Además, este módulo simplifica muchos servicios empresariales como acceso JNDI, integración con EJB, e integración con otros frameworks. [1]

2.2.1.3 Modulo AOP

Spring provee un amplio soporte para la programación orientada a aspectos en el módulo AOP. Este módulo sirve como base para el desarrollo de aspectos personalizados en las aplicaciones de Spring. [1]

2.2.1.4 Modulo DAO y abstracción JDBC

Trabajar con JDBC involucra escribir bastante código para abrir conexiones, crear transacciones, procesar consultas y cerrar conexiones. El JDBC de Spring y el módulo DAO (Data Access Objetct) abstraen el código por lo que se puede mantener el código de base de datos claro y simple, previniendo problemas que se producen al cerrar los recursos de la base de datos. Este

módulo incluye su propia capa de excepciones que genera mensajes de error más descriptivos que los propios mensajes de error SQL. [1]

Adicionalmente, este módulo utiliza el módulo AOP de Spring para proveer servicios de administración de transacciones para objetos en una aplicación de Spring.

2.2.1.5 Modulo ORM

Mapeo Objeto-Relacional (ORM, por sus siglas en inglés) es una opción a utilizar en lugar de JDBC, El módulo ORM soporta construcciones sobre DAO, proveyendo una conveniente manera de construir DAOs para algunas soluciones ORM. [1]

No es el objetivo de Spring implementar su propia solución ORM, pero soporta algunos frameworks ORM, incluyendo: Hibernate, Java Persistence API, Java Data Objects, y iBatis SQL Maps. La administración de transacciones de Spring soporta cada uno de estos framework ORM tan bien como lo hace con JDBC.

2.2.1.6 Administración de extensiones Java (JMX)

El módulo de Spring JMX permite fácilmente exponer los beans de una aplicación Spring como JMX MBeans. Esto hace posible monitorear y reconfigurar una aplicación corriendo en tiempo real. [1]

2.2.1.7 Conector Java EE API (JCA)

JCA (J2EE Connector Architecture) es un API Java EE que permite una integración de aplicaciones Java con una variedad de sistemas de información empresariales, incluyendo base de datos y servidores dedicados. [1]

Se podría decir que JCA es muy parecido a JDBC, con la excepción que JDBC está enfocado al trabajo con base de datos, JCA tiene un propósito más general ya que es un API de conexión para servidores dedicados.

2.2.1.8 El framework MVC de Spring

El framework Modelo Vista Controlador de Spring (MVC, por sus siglas en inglés) está basado en el paradigma que es ampliamente aceptado en la construcción de aplicaciones Web, que separa la capa de presentación, conocida comúnmente como la interfaz gráfica de usuario (GUI, por sus siglas en inglés), de la capa de negocio. Frameworks similares a este se pueden encontrar en gran cantidad para Java, como por ejemplo: Apache Struts, JSF, WebWork, y Tapestry. [1]

Aunque Spring se integra con algunos de estos frameworks, también incluye su propio framework MVC que promueve la técnica de bajo acoplamiento en la capa Web de la aplicación.

2.2.1.9 MVC Portable

Este framework de Spring está orientado a la desplegar información en formatos específicos enfocados a todo el conjunto de controladores que soportan el API Portable de Java. [1]

2.2.1.10 Módulo Web de Spring

Spring MVC y Spring MVC Portable requieren una consideración especial cuando se carga el contexto de una aplicación de Spring. Por consiguiente, el módulo Web de Spring provee clases con soporte especial para estos frameworks. El módulo Web de Spring también contiene soporte a varias tareas relacionadas con la Web, como soporte a la integración con Apache Struts y JSF. [1]

2.2.1.11 Acceso Remoto (Remoting)

Este es el módulo que permite tratar con las funcionalidades de otros sistemas. Se puede accede a través de la red a otros sistemas y también permite que otros sistemas puedan tener acceso a los servicios que se están exponiendo en una aplicación Java. Para conseguir esto es necesario escribir objetos remotos en la aplicación como si fueran POJOs locales. Existen las siguientes alternativas que se pueden emplear: Invocación de

métodos remotos (RMI, por sus siglas en inglés), Hessian, Burlap, JAX-RPC, y el propio de Spring HTTPInvoker. [1]

2.2.1.12 Servicios de Mensajes Java (JMS)

Este módulo permite el envío de mensajes independientemente de existir una red, al mismo tiempo este módulo ayuda a crear POJOs que son capaces de consumir mensajes asincrónicos. [1]

2.2.2 Inyección de dependencia (DI)

2.2.2.1 Introducción

La inyección de dependencia está basada en el uso de interfaces y clases, una interfaz puede ser implementada por una o varias clases. En los archivos de contexto de Spring se crean los beans que indica que implementación de la interfaz se va utilizar, es decir la clase que la implementa. A continuación se detalla un ejemplo básico de la utilización de la inyección de dependencia en Spring, este ejemplo fue tomado de "Spring in Action" [1]. El ejemplo consiste en el típico programa de imprimir un mensaje de saludo, con la diferencia que se emplea una interface que define el contrato para las clases de servicio.

```
package com.springinaction.chapter01.hello;
public interface GreetingService {
        void sayGreeting();
}
```

Ahora es necesario realizar la implementación de ese contrato, nótese que aunque no es necesario utilizar una interfaz, esta es una buena práctica para mantener separada la implementación del contrato.

```
package com.springinaction.chapter01.hello;
public class GreetingServiceImpl implements GreetingService {
private String greeting;
public GreetingServiceImpl() {}
public GreetingServiceImpl(String greeting) {
        this.greeting = greeting;
}
public void sayGreeting() {
        System.out.println(greeting);
}


public void setGreeting(String greeting) {
        this.greeting = greeting;
}
}
```

Esta clase tienen una propiedad simple "*greeting*", esta es una propiedad de tipo *String* que almacenará la cadena que será impresa cuando se realice la llamada al método *sayGreeting()*. Adicionalmente, en la clase la propiedad *greeting* puede ser asignada de dos maneras diferentes: a través del constructor de la clase, o utilizando el método de asignación de la propiedad.

En este momento es necesario definir quien realizará el llamado al constructor de la clase o al método de asignación, en este ejemplo esto va ha ser realizado por el contenedor de Spring mediante un archivo de

configuración denominado "*hello.xml*" que contendrá la configuración del servicio *greeting*.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
<bean id="greetingService"
        class="com.springinaction.chapter01.hello.GreetingServiceImpl">
        <property name="greeting" value="Buenos Dias!" />
</bean>
</beans>
```

El archivo de configuración XML declara una instancia de la clase en el contenedor de Spring y configura su propiedad *greeting* con el valor "*Buenos Dias!*"

La base de este archivo XML es el elemento <*beans*>, que es la raíz de cualquier archivo de configuración de Spring. El elemento <*bean*> es usado para decirle al contenedor de Spring como la clase debe ser configurada. Adicionalmente, el atributo *id* es usado para nombrar al bean como *greetingService* y el atributo *clase* especifica contiene el nombre de la clase de implementación.

Dentro del elemento <*bean*>, el elemento <*property*> es usado para asignar una propiedad, en este caso la propiedad *greeting*. El elemento <*property*> le dice al contenedor de Spring que tiene que llamar al método *setGreeting()*, pasando como parámetro "Buenos Dias!", esto es equivalente en Java a tener el siguiente código:

```
GreetingServiceImpl greetingService = new GreetingServiceImpl();
greetingService.setGreeting("Buenos Dias!");
```

Opcionalmente, se podría utilizar el archivo de configuración para que Spring emplee en constructor en lugar del método *setGreeting()*, el *<bean>* quedaría de la siguiente manera:

```
<bean id="greetingService"
    class="com.springinaction.chapter01.hello.GreetingServiceImpl">
    <constructor-arg value="Buenos Dias!" />
    </bean>
```

El código equivalente en Java de lo anterior sería:

```
GreetingServiceImpl greetingService =
new GreetingServiceImpl("Buenos Dias");
```

La última clase que es necesario realizar será la que pruebe el programa, para esto es necesario que esta clase cargue el contenedor de Spring y lo utilice para retornar el servicio.

```
package com.springinaction.chapter01.hello;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;
public class HelloApp {
public static void main(String[] args) throws Exception {
        BeanFactory factory =
        new XmlBeanFactory(new FileSystemResource("hello.xml"));
        GreetingService greetingService =
        (GreetingService) factory.getBean("greetingService");
        greetingService.sayGreeting();
}
}
```

La clase *BeanFactory* utilizada en el código anterior es el contenedor de Spring. Después de cargar el archivo *hello.xml* dentro de contenedor, se llama al método *getBean()* del BeanFactory para obtener una referencia al servicio *greetingService* con esta referencia finalmente se llama al método *sayGreeting()* y se imprime el mensaje.

Este simple ejemplo ilustró como configurar un bean inyectando un valor String dentro de una propiedad. Sin embargo, Spring es tan potente que permite inyectar beans dentro de otros beans utilizando DI, esto es realmente la esencia de Spring.

2.2.2.2 Análisis de la inyección de dependencia

Se puede decir que la inyección de dependencia es el núcleo de Spring, la DI es un patrón de diseño, sin embargo este patrón no complica el desarrollo, en su lugar DI permite construir aplicaciones cuyo código es más fácil de programar, entender y testear.

Inicialmente la inyección de dependencia fue referida con otro nombre inversión de control, dónde básicamente lo que se invierte es la dependencia y por eso finalmente decidieron que el nombre actual era el más adecuado.

Normalmente las aplicaciones están constituidas por un conjunto de clases que realizan alguna lógica de negocio. Tradicionalmente cada objeto es responsable de obtener sus propias referencias de los objetos con los que

colabora (dependencias), esto implica alto acoplamiento y código difícil de testear (basado en pruebas unitarias).

Cuando se aplica DI, los objetos están dando sus dependencias al momento de la creación que es realizada por una entidad externa que se encarga de coordinar cada objeto en el sistema. En otras palabras, las dependencias son inyectadas dentro de los objetos. Por lo tanto, DI significa una inversión de responsabilidad con respecto a cómo un objeto obtiene referencias de los objetos con los que colabora (ver Figura 2 [1]).

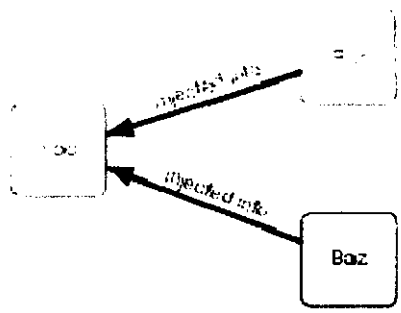


Figura 2.2 Con DI las dependencias son dadas a los objetos.

El gran beneficio de la inyección de dependencia es el bajo acoplamiento, bajo acoplamiento es definido como patrón dentro del diseño orientado a objetos. Si un objeto solo conoce sus dependencias a través de sus interfaces (no por su implementación ni por la forma como ellas están inicializadas) entonces la dependencia puede ser cambiada a una diferente implementación sin que el objeto dependiente conozca la diferencia.

Por ejemplo, si la clase *Foo* en la Figura 2 solo conoce la dependencia con *Bar* a través de una interfaz entonces la actual implementación de *Bar* no es

importante para *Foo*. *Bar* podría ser un POJO local, un servicio Web, un EJB a fin de cuentas *Foo* no tiene que preocuparse de eso.

Resumiendo, con la inyección de dependencia la responsabilidad de coordinar colaboración entre objetos dependientes ya no es un trabajo que realizan los objetos por sí mismos.

2.2.3 Programación orientada a aspectos (AOP)

2.2.3.1 Introducción

La programación orientada a aspectos es definida como una técnica que promueve la separación de funcionalidades de diferente contexto (seguridad, transacciones, etc.) dentro de un sistema. Los sistemas están compuestos de varios componentes, cada componente encargado de una funcionalidad específica. Sin embargo, y muy a menudo estos componentes añaden responsabilidad a su funcionalidad básica. Servicios como autentificación de usuarios, administración de transacciones, y seguridad suelen comunicarse con otros componentes para conseguir funcionalidad extra. El propósito de estos servicios es el de dividir en múltiples componentes a un sistema provocando que se presenten los siguientes problemas: [2]

Cuando el código que implementa un gran sistema es duplicado a través de múltiples componentes y es necesario cambiar la funcionalidad del sistema, es necesario cambiar múltiples componentes.

Los componentes están incluyendo funcionalidad que no es propia de su contexto. Un método que añade un registro de un cliente debería solo preocuparse de cómo insertar el cliente y no de la transaccionalidad ni de la seguridad.

La programación orientada a aspectos provee la manera de modularizar los servicios y entonces aplicarlos declarativamente a los componentes que ellos deberían afectar. Esto resulta en componentes mucho más cohesivos que se enfoque en sus propias funcionalidades, sin tener conocimiento alguno de los servicios en lo que ellos podrían ser involucrados. Resumiendo, la programación orientada a aspectos asegura que los POJOs permanezcan simples.

Analógicamente, se podría describir a los aspectos como capas que cubren muchos componentes de una aplicación como se ilustra en la Figura 3. Una aplicación consiste de módulos que implementan la funcionalidad del negocio (núcleo). Con AOP se puede cubrir el núcleo de la aplicación con capas de funcionalidad. Estas capas pueden ser aplicadas declarativamente a lo largo de toda la aplicación de una manera tan flexible que la aplicación no tendrá conocimiento de que existen. Esto mantiene a los servicio como seguridad, transaccionalidad, y autentificación de usuarios separados del núcleo de la aplicación (lógica de negocio). [1]

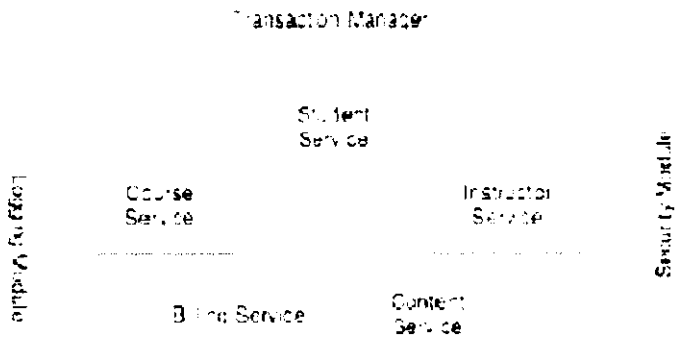


Figura 2.3 Las capas de servicios conseguidas con AOP.

2.2.3.2 Análisis de la programación orientadas a aspectos

Para analizar cómo trabaja la programación orientada a aspectos se empleará el ejemplo del libro "Spring in Action" [1], aunque es un ejemplo que no se encuentra dentro de un contexto de una aplicación real, es un ejemplo bastante útil para comprender la programación orientada a aspectos. En este ejemplo se tiene una clase que representa un caballero que va embarcarse en la búsqueda del Santo Grial. Suponiendo que esta implementación ya existe. Se pide mejorar la funcionalidad con un nuevo requerimiento, un trovador acompañará a cada caballero y se encargará de cantar las crónicas del caballero. Inicialmente, no parece un requerimiento difícil de implementar, se crea primero la clase que representará al trovador.

```
package com.springinaction.chapter01.knight;
import org.apache.log4j.Logger;
public class Minstrel {
private static final Logger SONG =
Logger.getLogger(Minstrel.class);
```

```
public void singBefore(Knight knight) {
SONG.info("Fa la la; Sir " + knight.getName() +
" is so brave!");
}
public void singAfter(Knight knight) {
SONG.info("Tee-hee-he; Sir " + knight.getName() +
" did embark on a quest!");
}
}
```

Canta antes de la búsqueda

Canta después de la búsqueda

Teniendo en cuenta la inyección de dependencia se tendrá una clase *KnightOfTheRoundTable* que tenga una instancia de *Minstrel*.

```
public class KnightOfTheRoundTable implements Knight {
...
private Minstrel minstrel;
public void setMinstrel(Minstrel minstrel) {
this.minstrel = minstrel;
}
...
public HolyGrail embarkOnQuest() throws QuestFailedException {
minstrel.singBefore(this);
HolyGrail grail = quest.embark();
minstrel.singAfter(this);
return grail;
}
}
```

Hasta aquí solo hay un pequeño problema. Cada caballero debe detenerse para decirle al trovador que cante la canción inicial (*singBefore*), luego el caballero puede continuar con la búsqueda. Después de la búsqueda el caballero debe recordarle al cantor que continúe cantando (*singAfter*). Sin lugar a dudas, el que el caballero tenga que recordar al trovador que realice el canto es un problema ya que esto puede afectar a su búsqueda, que es su

misión real. Por lo tanto, sería ideal que el trovador tomara la iniciativa de cantar automáticamente sin necesidad de que se lo tengan que decir.

Adicionalmente, el caballero no tendría que preocuparse que sus crónicas estén siendo cantadas dándole total independencia al caballero.
Los servicios de un trovador trascienden las responsabilidades de un caballero, en otras palabras el servicio del trovador es independiente a los deberes del caballero. Por consiguiente, esto da la sensación de que el cantor es un aspecto que añade sus servicios al caballero. Entonces los servicios del trovador pueden cubrir la funcionalidad del caballero, todo sin que el caballero conozca incluso de que el trovador está ahí (Figura 4).

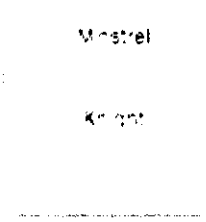


Figura 2.4 Un cantor como aspecto orientado cubre al caballero.

Ya dentro de código, es relativamente fácil convertir en aspecto al trovador utilizando la programación orientada a aspectos de Spring. Para esto se utiliza el nuevo namespace AOP de Spring 2.0, hay que declarar el namespace en el archivo de definición de contexto XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
http://www.springframework.org/schema/beans/
➥spring-beans-2.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">
...
</beans>
```

Con el namespace declarado, se puede crear el aspecto declarando un bean para el trovador como un bean en el contexto de Spring y luego se crea un aspecto que se comunica con el bean del caballero.

```
<bean id="minstrel"                                          } Declara bean para el cantor
class="com.springinaction.chapter01.knight.Minstrel"/>       }
<aop:config>
<aop:aspect ref="minstrel">          } Crea aspecto para el cantor
<aop:pointcut id="questPointcut"                             } Crea un punto de quiebre
expression="execution(* *.embarkOnQuest(..))                 }
➥and target(bean)" />                                       }
<aop:before
method="singBefore"                                          } Entrelaza a un cantor antes
pointcut-ref="questPointcut"                                 }
arg-names="bean" />                                          }
<aop:after-returning
method="singAfter"                                           } Entrelaza a un cantor después
pointcut-ref="questPointcut"                                 }
arg-names="bean" />                                          }
</aop:aspect>
</aop:config>
```

En el código anterior primero se realiza la creación del bean para el trovador, posteriormente se crea el aspecto y se lo referencia al bean del trovador para que sea esta clase la que implemente el aspecto. Se definen los puntos de corte es decir los lugares donde la funcionalidad del aspecto va a ser ejecutada basada en la ejecución del método *embarkOnQuest()*. Por último,

hay dos salidas para la comunicación del aspecto, se indica que el método *singBefore()* va a ser ejecutado antes del punto de corte y el método *singAfter()* va a ser ejecutado después del punto de corte.

El trovador sigue siendo un simple POJO ya que fue declarado como aspecto en el contexto de Spring. Por otro lado, y más importante todavía es que el caballero no necesita conocer de la existencia de trovador. Por consiguiente, la clase *KnightOfTheRoundTable* puede regresar a su estado original.

```
public class KnightOfTheRoundTable implements Knight {
private String name;
private Quest quest;
public KnightOfTheRoundTable(String name) {
        this.name = name;
}
public HolyGrail embarkOnQuest() throws QuestFailedException {
        return quest.embark();
}
public void setQuest(Quest quest) {
        this.quest = quest;
}
}
```

2.2.4 El contenedor de Spring y el empleo de beans

En una aplicación de Spring, todos los objetos de la aplicación están dentro de un contenedor de Spring. El contenedor es el encargado de crear los objetos y vincularlos, configurarlos y administrar su ciclo de vida.

El contenedor es el núcleo del framework, el contenedor utiliza inyección de dependencia para administrar los componentes de la aplicación. Esto incluye la creación de relaciones entre componentes colaboradores.

El contenedor de Spring tiene varias implementaciones que pueden ser categorizadas en dos distintos tipos. *Bean factories,* definidos por la interfaz *org.springframework.beans.factory.BeanFactory,* que son los contenedores más simples, proveyendo suporte básico para DI. Application Contexts (contextos de la aplicación), que son definidos por la interfaz *org.springframework.context.ApplicatiapliaplicacionContext* y está basado en la noción de un bean factory, provee servicios del framework como la habilidad para resolver mensajes textuales de archivos de propiedades y la habilidad para publicar eventos de la aplicación que van a ser tratados como eventos de tipo listener.

2.2.4.1 Introducción al Bean Factory

Un *bean factory* es básicamente la implementación del patrón de diseño Factory [4], es decir una clase cuya responsabilidad es la de crear y administrar beans. Sin embargo, a diferencia de muchas implementaciones de Factory, que a menudo tratan con simples objetos, el Factory de Spring tiene como propósito general crear y administrar muchos tipos de beans, además de conocer muchos de los objetos de la aplicación y de ser capaz de crear las relaciones entre objetos colaboradores que han sido inicializados.

Hay muchas implementaciones de Factory en Spring. Sin embargo, una de las más utilizadas es *org.springframework.beans.factory.xml.XmlBean-Factory* que carga sus beans basado en las definiciones contenidas en un archivo XML.

Para crear un *XmlBean Factory* es necesario pasar una instancia de *org.springframework.core.io.Resource* al constructor. El *Resource* se encargará de proveer el archivo XML al Factory de Spring. Por ejemplo, el siguiente código utiliza *FileSystemResource* para crear un *XmlBeanFactory* cuyas definiciones son leídas de un archivo XML:

```
BeanFactory factory =
new XmlBeanFactory(new FileSystemResource("c:/beans.xml"));
```

Esta línea de código le dice al bean factory que lea las definiciones del archivo XML, pero el bean factory no instancia los beans todavía. Los beans son cargados "perezosamente" por lo que el bean factory cargará las definiciones de los beans (descripción y propiedades) inmediatamente cuando estas sean necesarias.

Para retraer un bean de un *BeanFactory*, simplemente es necesario llamar al método *getBean()* pasando el ID del bean que se quiere retraer:

```
MyBean myBean = (MyBean) factory.getBean("myBean");
```

Cuando el método *getBean()* es llamado, el factory inicializará el bean y colocará las propiedades del bean usando DI. Así inicia el ciclo de vida del

bean dentro del contenedor de Spring. A continuación en la Tabla 1, se muestra todas las implementaciones posibles de Resource para que XmlBeanFactory pueda ser creado:

Implementación de Resource	Propósito
org.springframework.core.io.ByteArray Resource	Define el resource a partir de un array de bytes.
org.springframework.core.io.ClassPath Resource	Define el resource que va ser retraído de un classpath.
org.springframework.core.io.Descriptive Resource	Define el resource que contiene un resource descriptivo.
org.springframework.core.io.FileSystem Resource	Define el resource que es retraído de un archivo del sistema.
org.springframework.core.io.InputStream Resource	Define el resource a partir de una entrada stream.
org.springframework.web.portlet.context. PortletContextResource	Define el resource a partir de un contexto de portlet.
org.springframework.web.context.support. ServletContextResource	Define el resource a partir de un contexto de un servlet.
org.springframework.core.io.UrlResource	Define el resource que es retraído de un URL.

Tabla 2.1 Implementaciones de Resource [1].

2.2.4.2 Introducción al application context

Los bean factories están bien para simples aplicaciones. Sin embargo, para explotar las grandes ventajas de Spring, probablemente sea necesario cargar los beans de la aplicación utilizando el contexto de la aplicación.

Tanto el *ApplicationContext* como el *BeanFactory* son muy similares ya que ambos cargan definiciones de beans, relaciones de beans entre si y administra beans bajo pedidos. Sin embargo, el *ApplicationContext* ofrece mucho más:

- Provee un significado para resolver mensajes de texto, incluyendo soporte para un gran tipo de mensajes.
- Provee una manera genérica de cargar archivos de sistema, como imágenes.
- Permite publicar eventos para beans que van a ser registrados como listeners.

Debido a esta funcionalidad extra, el *ApplicationContext* es preferido sobre el *BeanFactory* en casi todas las aplicaciones. Aunque se debería considerar utilizar *BeanFactory* donde los recursos son escasos, como en los dispositivos móviles. Dentro de las implementaciones que son más comunes para el *ApplicationContext* se tiene:

- ClassPathXmlApplicationContext: Carga una definición de contexto desde un archivo XML localizado en un classpath. Los archivos de definición de contexto son tratados como fuentes classpath.
- FileSystemXmlApplicationContext: Carga una definición de contexto desde un archivo XML localizado en el sistema.
- XmlWebApplicationContext: Carga una definición de contexto desde un archivo XML contenido dentro de una aplicación Web.

El código para cargar un contexto de la aplicación desde un archivo de sistema es muy similar al código utilizado en el *BeanFactory*:

```
ApplicationContext context =
new FileSystemXmlApplicationContext("c:/foo.xml");
```

Para retraer un bean desde un *ApplicationContex* se emplea el método *getBean()* que ya se utilizo en el *BeanFactory*, debido a que la del *ApplicationContext* hereda de la interfaz del *BeanFactory*.

Otra principal diferencia entre al *ApplicationContext* y el *BeanFactory* es la manera como el patrón de diseño Singleton es manejado, es decir como los beans se cargan. Un bean factory realiza la carga de beans perezosamente es decir cuando los beans son necesarios, específicamente cuando se realiza el llamado al método *getBean()*. Un objeto de contexto de aplicación es un poco más inteligente ya que precarga todo los beans una vez que el contexto inicia, esto asegura que los beans estén listos cuando se los necesite la aplicación no tiene que esperar a que los beans sean creados.

2.2.4.3 Relacionando beans

Una de las características más interesantes y potentes de Spring es que permite relacionar beans y proveer una colaboración entre ellos. Por ejemplo tenemos el código de la siguiente clase:

```
package com.springinaction.springidol;
public class Instrumentalist implements Performer {
public Instrumentalist() {}
public void perform() throws PerformanceException {
```

```
System.out.print("Playing " + song + " : ");
instrument.play();
}
private String song;
public void setSong(String song) {
        this.song = song;
}
private Instrument instrument;
public void setInstrument(Instrument instrument) {
        this.instrument = instrument;
}
}
```

La clase este integrada por dos propiedades *song* e *instrument*, cada una de estas propiedades incluye sus respectivos métodos de asignación. Con Spring se puede utilizar estos métodos de asignación en la definición de los beans.

```
<bean id="kenny"
class="com.springinaction.springidol.Instrumentalist">
<property name="song" value="Jingle Bells" />
</bean>
```

Sin embargo, el código anterior inyecta valores simples (como debería ser al tratarse de un POJO) utilizando únicamente el método de asignación de la propiedad *song*. El código equivalente en Java a lo anterior se muestra a continuación, sin embargo hay que tener muy en cuenta que el código en Spring es totalmente desacoplado:

```
Instrumentalist kenny = new Instrumentalist();
kenny.setSong("Jingle Bells");
```

Suponiendo que nuestro objeto *kenny* puede tocar cualquier instrumento, es recomendable que la propiedad *instrument* sea una interface para que permite al objeto *kenny* tocar cualquier instrumento, en otras palabras que exista una implementación para cada instrumento:

```
public interface Instrument {
void play();
}
```

Suponiendo que el objeto *kenny* quiere tocar un saxofón es necesario crear la implementación para este instrumento:

```
package com.springinaction.springidol;
public class Saxophone implements Instrument {
public Saxophone() {}
public void play() {
        System.out.println("TOOT TOOT TOOT");
}
}
```

Antes de que el objeto *kenny* pueda tocar el instrumento es necesario crear el <*bean*> en Spring, el siguiente código realiza esa actividad:

```
<bean id="saxophone"
class="com.springinaction.springidol.Saxophone" />
```

Se puede notar que el saxofón no tiene propiedades por lo que no es necesario asignar nada y se puede omitir la declaración de <*property*>.

Con el objeto *saxophone* declarado, se lo puede asociar al objeto *kenny* para que lo toque. Se modifica el bean inicial de *kenny* para que realice la inyección a la propiedad *instrument*:

```
<bean id="kenny"
class="com.springinaction.springidol.Instrumentalist">
<property name="song" value="Jingle Bells" />
<property name="instrument" ref="saxophone" />
</bean>
```

El objeto *kenny* ha sido inyectado con sus propiedades y está listo para ser utilizado y tocar el saxofón, incluso gracias a que se está empleando interfaces, es posible que *kenny* puede tocar otro instrumento sin que sea necesario modificar el código. Es por esta razón que Spring promueve el uso de las interfaces ya que lo que se consigue es un bajo acoplamiento basado en la inyección de dependencia. Por ejemplo, asumiendo que *kenny* necesita tocar el piano es necesario crear la siguiente clase que implemente Instrument:

```
package com.springinaction.springidol;
public class Piano implements Instrument {
public Piano() {}
public void play() {
        System.out.println("PLINK PLINK PLINK");
}
}
```

La clase *Piano* debe ser declarada como un bean en Spring, el siguiente código XML lo realiza:

```
<bean id="piano"
class="com.springinaction.springidol.Piano" />
```

Ahora es turno de actualizar el bean de *kenny* para que pueda tocar el piano, simplemente es necesario realizar un pequeño cambio en la declaración:

```
<bean id="kenny"
```

```
class="com.springinaction.springidol.Instrumentalist">
<property name="song" value="Jingle Bells" />
<property name="instrument" ref="piano" />
</bean>
```

Con este cambio *kenny* puede tocar el piano en lugar del saxofón. De hecho, la clase *Instrumentalist* solo conoce su propiedad instrumento a través de la interface *Instrument* por lo que fue necesario modificar la clase *Instrumentalist* para soportar una nueva implementación de Instrument, un *Instrumentalist* puede tocar ambos un saxofón y un piano ya que está totalmente desacoplada de ambos. Incluso, si fuese necesario tocar un tercer instrumento tan solo se tendría que crear una nueva clase que implemente la interfaz *Instrument* y hacer el pequeño cambio en la declaración del bean de *kenny*.

Suponiendo que ahora se quiere crear un objeto llamado *hank* que es capaz de tocar varios instrumentos, al igual que *kenny*, con la pequeña diferencia que *hank* lo puede hacer simultáneamente. Para conseguir esto se empleará colecciones. La clase *OneManBand* representará a *hank*:

```
package com.springinaction.springidol;
import java.util.Collection;
public class OneManBand implements Performer {
public OneManBand() {}
public void perform() throws PerformanceException {
        for(Instrument instrument : instruments) {
        instrument.play();
}
}
private Collection<Instrument> instruments;
public void setInstruments(Collection<Instrument> instruments) {
```

```
        this.instruments = instruments;
    }
}
```

Esta clase incluye una colección de instrumentos y realiza una iteración de la misma. Lo interesante de la clase es que tienen un método de asignación para una colección de instrumentos que permitirá inyectarlos. El código del bean sería:

```
<bean id="hank" class="com.springinaction.springidol.OneManBand">
<property name="instruments">
<list>
<ref bean="guitara" />
<ref bean="armonica" />
</list>
</property>
</bean>
```

El elemento <list> contiene uno o más valores. La declaración <ref> es empleada para referenciar a otros beans dentro del contexto de Spring, en otras palabras se está configurando para que el objeto *hank* toque la guitarra y la armónica.

2.2.4.4 Ámbito de los beans en Spring

Por defecto los beans en Spring son creados basados en Singleton, es decir para un bean se crea una sola instancia y la misma es compartida para todas las solicitudes que se realicen a ese bean [4]. Sin embargo, Spring permite cambiar el ámbito de los beans consintiendo que un bean pueda ser

inicializado para cada llamada o solicitud que realice, en este caso cada solicitud tendrá su propia inicialización del bean y no es compartida. A continuación se describen los ámbitos que tienen los beans en Spring: [1]

Singleton: Es el ámbito por defecto y se crea una instancia para todo el contendedor.

Prototype: Permite que un bean sea inicializado cualquier número de veces (una por usuario).

Request: Por cada pedido HTTP que se realice en aplicaciones Web (por ejemplo: Spring MVC).

Session Por cada sesión HTPP que se realice en aplicaciones Web (por ejemplo: Spring MVC).

2.3 Trabajando con base de datos en Spring

Spring incluye un conjunto de frameworks para acceso de datos que se integra con una variedad de tecnologías para acceso a datos. Si los datos se están persistiendo mediante JDBC, iBATIS, o un mapeador objeto/relación (ORM, por sus siglas en inglés) como Hibernate, Spring elimina la molestia de tener que escribir código para acceso a datos en el medio de la persistencia. Spring se encarga de realizar el trabajo de acceso de datos de bajo nivel para que los desarrolladores puedan administrar los datos de la aplicación.

2.3.1 El DAO de Spring

Spring incluye soporte DAO (patrón de diseño [4]) que se basa, como todo en Spring, en una interfaz. DAO provee una manera de leer y escribir datos en una base de datos. DAO expone esta funcionalidad a través de una interfaz a la que pueden acceder todas las aplicaciones. La Figura 5 muestra el diseño de la capa de acceso a datos basada en DAO. [1]

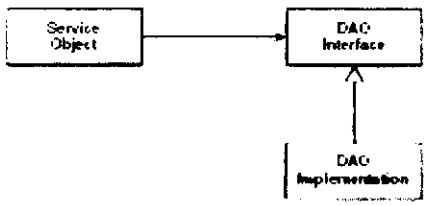


Figura 2.5 Capa de acceso a datos basada en DAO

Los objetos de servicio están accediendo a DAO a través de interfaces. Esto tiene un par de ventajas. Primero, permite que los objetos de servicios sean fáciles de testear ya que están desacoplados de la implementación para acceso a datos. De hecho, esto permite que se pueda probar los objetos de servicio sin conexión a una base de datos, por lo que las pruebas son más rápidas e incluso se puede verificar si hay reglas para la inconsistencia de datos.

Adicionalmente, la tecnología de persistencia está aislada a la interfaz de DAO, lo que facilita escoger cualquier framework de persistencia y cambiarlo por otro con un mínimo impacto en el resto de la aplicación.

Spring emplea el patrón Template Method [4] que define por un lado una estructura base para tomar la responsabilidad de los procesos comunes. Por ejemplo: conexiones, transacciones, liberación de recursos de conexión, tratamiento de excepciones de acceso a datos. Y por otro lado delega el trabajo a sub clases para cubrir algunas responsabilidades específicas, como operaciones de inserción y actualización de datos. Al estar definido de esta manera Spring provee la selección de cualquier tecnología de administración de datos como JDBC, Hibernate, IBATIS, TopLink de Oracle.

2.3.1.1 Data soucers y drivers de Spring

Spring ofrece varias opciones para configurar los beans para data sources en una aplicación, las opciones son: [1]

- Data sources que son definidos por los drivers JDBC.
- Data sources basados en JNDI.
- Data sources situados en un pool de conexiones.

Para ambientes de producción es recomendable utilizar conexiones que provengan de un pool de conexiones. Sin embargo, el driver más simple de configurar en Spring es el que está definido a través de un driver JDBC. Spring ofrece dos clases para acceso a datos basados en este driver: [1]

- DriverManagerDataSource: retorna una nueva conexión cada vez que una conexión es solicitada. Las conexiones que provee DriverManagerDataSource no se encuentran en un pool.
- SingleConnectionDataSource: Retorna la misma conexión cada vez que la conexión es solicitada. Aunque *SingleConnectionDataSource* no es exactamente un pool de conexiones, se podria pensar que es un pool con una sola conexión.

A continuación se muestra el código que crea un bean data source que emplea el driver JDBC:

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.
➥ DriverManagerDataSource">
<property name="driverClassName"
value="org.hsqldb.jdbcDriver" />
<property name="url"
value="jdbc:hsqldb:hsql://localhost/roadrantz/roadrantz" />
<property name="username" value="sa" />
<property name="password" value="" />
</bean>
```

En el anterior código se asume que se está empleando un data source para una base de datos denominada “roadrantz” cuyo usuario es “sa” y no dispone de contraseña, lógicamente estos valores cambiaran para la conexión real de una aplicación.

Adicionalmente, la diferencia utilizando *SingleConnectionDataSource* en lugar de *DriverManagerDataSource* es que se provee una conexión compartida pero que no es necesariamente un pool de conexiones.

Sin embargo, hay que tener muy en cuenta que para ambientes reales de producción de una aplicación *SingleConnectionDataSource* tiene una sola conexión que no trabaja muy bien en aplicaciones con procesos multihilos.

Adicionalmente, y a pesar de que *DriverManagerDataSource* puede ser utilizado en aplicaciones multihilos, incurre en problemas de rendimiento debido a que crea una nueva conexión cada vez que una conexión es solicitada. Por esas limitaciones, ser recomienda considerablemente usar un pool de conexiones.

2.3.2 Integrando Hibernate con Spring

Actualmente debido a la complejidad de las aplicaciones, los requerimientos de persistencia han evolucionada y JDBC no tiene todas las características que se requieren al trabajar con persistencia. Por ejemplo, JDBC no permite mapear las propiedades de los objetos a las columnas de una tabla y tener todas las sentencias y consultas SQL preestablecidas para la aplicación. Además, hay que tener en cuenta las siguientes características avanzadas que también son importantes al hablar de persistencia: [4]

Carga perezosa: Como la colaboración entre objetos es cada vez más compleja, no es necesario contar con toda la estructura de la relación

inmediatamente. Por ejemplo, si se está seleccionando una colección de objetos órdenes de compra y cada objeto contienen una colección de objetos línea de ítems. Si se está interesado solamente en los atributos de las órdenes de compra no hay sentido en obtener también los datos de líneas de ítem, ya que esto podría ser caro en términos de rendimiento. Con la carga perezosa se puede obtener solo los datos que se necesitan.

Carga Implícita: Este proceso es lo contrario a la carga perezosa. La carga implícita permite obtener un conjunto completo de objetos en una consulta. En los casos en que se esté interesado por lo objetos órdenes de compra y su objetos asociados (línea de ítem). La carga implícita permite obtener los datos necesarios y vinculados en una sola operación, ahorrando costosas solicitudes que normalmente sobrecargan el canal de comunicación.

Cascada: En algunas ocasiones un cambio en una tabla puede resultar en un cambio también en otra tabla. Por ejemplo en el caso de la orden de compra, cuando una orden es borrada, también es necesario borrar las líneas de ítems asociadas a la orden.

Hay algunos frameworks que están disponibles a brindar estos servicios. Normalmente estos frameworks son conocidos como modeladores objeto/relación (ORM, por sus siglas en inglés). Utilizar un ORM en la capa de persistencia de una aplicación puede ahorrar un gran número de líneas de código y tiempo de desarrollo.

Spring provee soporte para varios frameworks ORM, incluyendo Hibernate, iBATIS, Apache OJB, Java Data Objects (JDO), TopLink de Oracle y el API de persistencia de Java (JPA). [1]

Hibernate es actualmente el framework ORM más popular, es un framework abierto que provee no solo mapeo objeto/relación sino que también un conjunto de características muy sofisticadas que se esperan en un completo ORM como cache, carga perezosa, carga implícita y cache distribuida.

2.3.2.1 Plantillas de Hibernate

La principal interfaz para interactuar con Hibernate es *org.hibernate.Session*. La interfaz *Session* provee funcionalidad básica para acceso a datos como la habilidad para grabar, actualizar, borrar y cargar objetos desde la base de datos. Esta interfaz cubrirá todas las necesidades de persistencia. La manera de obtener una referenciarse del objeto *Session* de Hibernate es a través de una implementación de la interfaz *SessionFactory* de Hibernate.
El *HibernateTemplate* de Spring provee una capa abstracta para simplificar el trabajo de abrir y cerrar sesiones de Hibernate y para convertir las excepciones de Hibernate en las excepciones ORM de Spring. El siguiente código XML muestra como configurar HibernateTemplate en Spring: [1]

```
<bean id="hibernateTemplate"
class="org.springframework.orm.hibernate3.HibernateTemplate">
<property name="sessionFactory" ref="sessionFactory" />
</bean>
```

La propiedad *sessionFactory* está haciendo referencia a una implementación de *org.hibernate.SessionFactory* que se encuentra más adelante en esta sección del documento.

Para el mapeo de archivos con Hibernate se utiliza archivos XML y el *LocalSessionFactoryBean* de Spring. Como se muestra en la Figura 6 el *LocalSessionFactoryBean* es un factory de Spring que produce una instancia *SessionFactory* de Hibernate que realiza el mapeo de metadatos a partir de archivos XML.



Figura 2.6 LocalSessionFactoryBean de Spring [1]

El siguiente código XML muestra como configurar un *LocalSessionFactory-Bean* que carga los archivos de mapeo:

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.
➥ LocalSessionFactoryBean">
<property name="dataSource" ref="dataSource" />
<property name="mappingResources">
<list>
<value>com/roadrantz/domain/Rant.hbm.xml</value>
<value>com/roadrantz/domain/Motorist.hbm.xml</value>
<value>com/roadrantz/domain/Vehicle.hbm.xml</value>
</list>
</property>
<property name="hibernateProperties">
<props>
<prop key="hibernate.dialect">${hibernate.dialect}</prop>
</props>
```

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

La propiedad *dataSource* hace referencia a una implementación de *javax.sql.DataSource*. La propiedad *mappingResources* toma una lista de unas o más rutas para ubicar los archivos de mapeo en el classpath. Finalmente la propiedad *hibernateProperties* provee una configuración adicional propia de la sesión de Hibernate. Es necesario especificar el dialecto de Hibernate (el lenguaje SQL para cada motor de base de datos).

Para trabajar con anotaciones (esto es disponible si se utiliza la versión 3 Hibernate y se trabaja con el JRE 5 de Java) la configuración necesaria es similar a la de *LocalSessionFactoryBean*. Sin embargo, es necesario utilizar el *AnnotationSessionFactoryBean* de Spring, este se encargará de crear un *SessionFactory* de Spring basado en anotaciones. El XML requerido para configurar un *AnnotationSessionFactoryBean* se muestra a continuación:

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.
➥ annotation.AnnotationSessionFactoryBean">
<property name="dataSource" ref="dataSource" />
<property name="annotatedClasses">
<list>
<value>com.roadrantz.domain.Rant</value>
<value>com.roadrantz.domain.Motorist</value>
<value>com.roadrantz.domain.Vehicle</value>
</list>
</property>
<property name="hibernateProperties">
<props>
<prop key="hibernate.dialect">${hibernate.dialect}</prop>
</props>
```

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

El *datasource* y las propiedades de Hibernate son similares en ambos casos. Pero en lugar de configurar una o más archivos de mapeo, es necesario configurar *AnnotationSessionFactoryBean* con una o más clases que usen anotaciones de persistencia.

Hibernate no es el único framework ORM que se puede integrar con Spring, existen muchos más, y la gran ventaja de Spring es que la forma de comunicarse con todos ellos es muy similar, por lo que no debería ser problema alguno cambiar un framework ORM por otro.

2.3.3 Administración de transacciones

Al habla de sistemas muchas veces es necesario agrupar muchas operaciones en una unidad de trabajo, ese es lo que se denomina transacciones. Si todas las operaciones de una transacción son realizadas correctamente se puede hablar de una transacción exitosa. Pero si algo sale mal todo regresa al estado inicial antes de la transacción, en otras palabras es como si nada hubiera pasado. [4]

Las transacciones son importantes sobre todo en procesos que involucran persistencia de datos, donde los datos no pueden quedar en estados inconsistentes. Spring incluye un gran soporte para la administración de transacciones, ya sea de manera declarativa o a través de programación.

En Spring existen dos maneras de manejar transacciones. La primera realizada en el código de programación. La segunda es realizada declarativamente lo que permite desacoplar la transaccionalidad de las operaciones. [1]

2.3.4 Los manejadores de transacciones de Spring

Spring no incluye directamente manejadores de transacción, por lo que viene con un conjunto de administradores de transacción basados en JTA (el API de transacciones de Java) para que sean utilizados de acuerdo a la plataforma en la que se vaya a implementar. A continuación en la Tabla 2 se delinea los manejadores de transacciones y para que plataforma aplican:

Manejadores de transacciones de Spring	Uso
jca.cci.connection. CciLocalTransactionManager	Empleado como soporte de conexión J2EE, arquitectura JCA e interfaces comunes de cliente (CCI).
jdbc.datasource. DataSourceTransactionManager	Empleado para persistencia con JDBC e IBATIS.
jms.connection.JmsTransactionManager	Empleado para JMS 1.x.
jms.connection. JmsTransactionManager102	Empleada para JMS 2.x.
orm.hibernate. HibernateTransactionManager	Empleado para persistencia con Hibernate versión 2.
orm.hibernate3. HibernateTransactionManager	Empleado para persistencia con Hibernate versión 3.
orm.jdo.JdoTransactionManager	Empleado para persistencia con JDO.
orm.jpa.JpaTransactionManager	Empleado para persistencia con JPA.
orm.toplink.TopLinkTransactionManager	Empleado para persistencia con TopLink de Oracle.
transaction.jta.JtaTransactionManager	Empleado para distribución de transacciones.
transaction.jta. OC4JJtaTransactionManager	Empleado para utilizar el contenedor OC4J JEE de Oracle.
transaction.jta. WebLogicJtaTransactionManager	Empleado para distribuir aplicaciones utilizando WebLogic.

Tabla 2.2 Los manejadores de transacciones de Spring [1]

Cada manejador de transacción actúa como fachada para una implementación de transaccionabilidad. La Figura 7 muestra la relación entre los manejadores de transacciones y la implementación específica de cada plataforma. Para emplear los manejadores de transacciones es necesario declararlos en el contexto de la aplicación. [1]

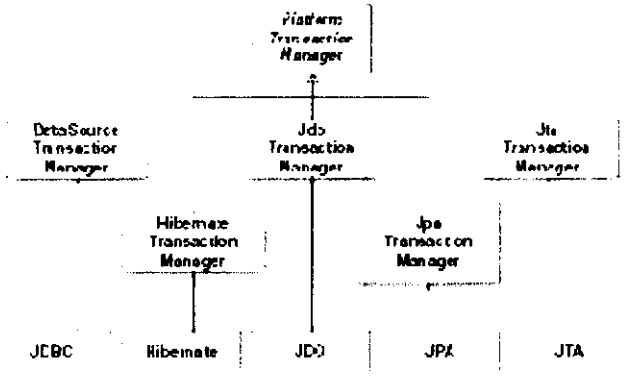


Figura 2.7 Relación de los manejadores de transacciones de Spring.

2.3.5 Transacciones de Hibernate

Para soportar transacciones en una aplicación que trabaja con una capa de persistencia de Hibernate, es necesario emplear *HibernateTransaction-Manager* y declararlo como bean usando código XML:

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

El código anterior asume que se está utilizando la versión 3 de Hibernate. La propiedad *sessionFactory* hace referencia al bean que configura la sesión de Hibernate (existe un ejemplo de la configuración de un bean de sesión de Hibernate en la sección 2.3.2 de este documento). [1]

HibernateTransactionManager delega la responsabilidad para la administración de transacciones a un objeto *org.hibernate.Transaction*. Cuando una transacción exitosa se completa *HibernateTransactionManager* llamará al método *commit()* en el objeto transacción. Similarmente, cuando una transacción no se ejecuta con éxito, se realiza la llamada al método *rollback()* del objeto transacción. [1]

2.4 El framework Web de Spring

El framework Web de Spring está diseñado para permitir a los desarrolladores la construcción de aplicaciones Web. Está basado en el patrón de diseño modelo vista – controlador (MVC, por sus siglas en inglés). Spring MVC ayuda a construir aplicaciones Web que son flexibles y tienen bajo acoplamiento. [4]

2.4.1 Introducción a Spring MVC

En una aplicación Web se recibe solicitudes de clientes, esas solicitudes se procesan en un servidor y luego regresan al cliente como resultados. Bajo

este contexto Spring MVC procesa esas solicitudes a través de servlets despachadores, mapeadores, controladores y vistas resultantes. [1]

Cada vez que se realiza una solicitud usando cualquier browser se dirige a una clase de Spring denominada *DispatcherServlet*. Esta clase está basada en un servlet y está encargada de despachar todas las solicitudes. El *DispatcherServlet* despacha la solicitud directamente a un controlador (basado en el patrón de diseño) el cual se encarga de delegar la responsabilidad de una solicitud a otros componentes de la aplicación que realizan el proceso respectivo a la solicitud. Sin embargo, una aplicación podría tener varios *DispatcherServlet* y controladores. Por esta razón es necesario disponer de un mapeador que se encargue de definir que controlador es utilizado para una solicitud. El mapeador se basa en la información que viene con el URL para tomar su decisión. Una vez que el apropiado controlador ha sido escogido, *DispatcherServlet* envía su solicitud al controlador el que se su vez delega la responsabilidad a las clases de negocio. Normalmente, la lógica procesada por el controlador debe regresar de vuelta al cliente para ser visualizada en un navegador de Internet (modelo). Sin embargo, enviar la información no es suficiente, necesita ser formateada en un formato que sea aceptado por un navegador como por ejemplo HTML. Esta información que debe ser dada al cliente se le conoce como vista y en la mayoría de los casos se trata de un JSP. El objeto modelo *ModelAndView* es el encargado de tener la información tanto del modelo como de la vista responsable de mostrar los resultados. Una vez que *DispatcherServlet* conoce que vista mostrará los resultados la solicitud

termina. A continuación la Figura 7 indica todo el proceso delineado anteriormente: [1]

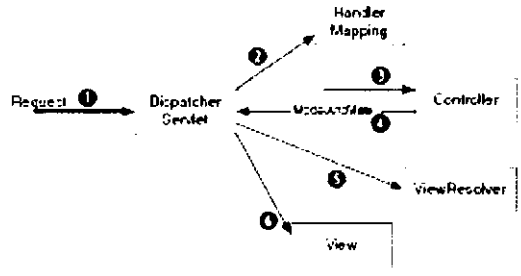


Figura 2.8 Solicitud procesada por *DispatcherServlet*

2.4.2 Configuración del DispatcherServlet de Spring

El núcleo principal de Spring es el *DispatcherServlet*, un servlet que funciona como el controlador frontal de Spring MVC. Un *DispatcherServlet* debe ser configurado en el archivo web.xml de la aplicación. A continuación se muestra el código que configura un *DispatcherServlet*:

```
<servlet>
<servlet-name>evaluation</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet
</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
```

El <*servlet-name*> indica el nombre que se debe emplear para el archivo de contexto de Spring, este archivo de contexto es iniciado cuando el *DispatcherServlet* es cargado. Por ejemplo, porque el nombre usado en la declaración es *evaluation*, el archivo de contexto de Spring debe ser nombrado como *evaluation- servlet.xml.*

Posteriormente, se debe indicar que URLs deben ser controlados por *DispatcherServlet*. Hay que añadir en el archivo web de la aplicación los mapeos, es decir los <servlet-mapping> que van a controlar todas los URLs que terminen con la extensión htm:

```
<servlet-mapping>
<servlet-name>roadrantz</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

La extensión .htm puede ser reemplazada por cualquier otra extensión. Sin embargo, HTML es el estándar de las páginas Web de un sitio estático y podría resultar una buena idea usar esta extensión que tienen una amplia difusión. Esta es toda la configuración necesaria para empezar a escribir la capa de presentación de la aplicación.

Una técnica que permite mejorar el mantenimiento general de una aplicación, consiste en distribuir el contexto de la aplicación en varios archivos, en lugar de tener un solo archivo. Por ejemplo, se puede tener un archivo de contexto para cada servicio tal es el caso: un archivo de contexto para la configuración de la capa de persistencia, un archivo para la configuración de MVC, un archivo para la seguridad. Para este efecto es necesario primeramente configurar el objeto *ContextLoaderListener* de Spring en el archivo web de configuración de la aplicación (web.xml):

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

En caso que no se ingrese esta configuración, Spring utilizará como archivo de contexto el definido por defecto. Para especificar los archivos de contexto de Spring que se cargaran al iniciar la aplicación es necesario utilizar el objeto *contextConfigLocation* de Spring:

```
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>
/WEB-INF/roadrantz-service.xml
/WEB-INF/roadrantz-data.xml
/WEB-INF/roadrantz-security.xml
</param-value>
</context-param>
```

El código anterior le dice a Spring que archivos de contexto debe cargar y dónde se encuentran ubicados.

2.4.3 Configurando el controlador de Spring

En este momento se puede realizar la configuración del controlador de Spring, el controlador es básicamente una clase Java, específicamente un servlet, que se encarga de atender la solicitud realizada por el cliente (el cliente realiza la solicitud usando un URL específico) y realizar el llamado al servicio o lógica de negocio. Normalmente un método de otra clase de Java. A continuación se describe una clase controladora de Spring: [1]

```
package com.roadrantz.mvc;
```

```
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;
import com.roadrantz.service.RantService;
public class HomePageController extends AbstractController {
public HomePageController() {}
protected ModelAndView handleRequestInternal(
HttpServletRequest request, HttpServletResponse response)
throws Exception {
List recentRants = rantService.getRecentRants();
return new ModelAndView("home",
"rants", recentRants);
}
private RantService rantService;
public void setRantService(RantService rantService) {
        this.rantService = rantService;
}
}
```

Básicamente, lo que realiza el controlador es una respuesta a la solicitud del cliente, donde se crea una colección *recentRants* a partir de objetos obtenidos a través de un servicio (el servicio está llamado a través de una interfaz) y este objeto colección es retornado como parámetro a la vista que para el ejemplo tienen el nombre de *home*.

Ahora que ya se dispone del controlador, es hora de configurarlo en el archivo de configuración de contexto del *DispatcherServlet*, suponiendo que se tienen varios archivos de configuración de acuerdo a los servicios como se recomendó anteriormente:

```
<bean name="/home.htm"
class="com.roadrantz.mvc.HomePageController">
```

```
<property name="rantService" ref="rantService" />
</bean>
```

Como se menciono *rantService* es una propiedad cuya implementación será inyectada gracias a la interfaz *RantService*. Este bean tiene como particularidad el hecho que no tiene un ID de identificación, en su lugar tiene un atributo *name*, lo que significa que se empleará un patrón URL de /home.htm. Toda solicitud que emplee en su URL el patrón será redirigido por *DispatcherServlet* al controlador que se acabo de crear.

2.4.4 Configuración de la vista resultante

Para devolver el servicio solicitado por el cliente, es necesario crear la vista que mostrará el resultado al cliente en un navegador de Internet. Por lo que es necesario que la vista tenga un formato que puede ser soportado por cualquier navegador de Internet. Normalmente, el formato que se emplea para las vistas es JSP, que es un formato reconocido por cualquier navegador web.

Es necesario indicarle a Spring que página JSP va utilizar para exponer los resultados. Para este propósito es necesario declarar un bean en al archivo de contexto. Para realizar esto, se debe tomar el nombre de la vista que es retornada en el objeto *ModelAndView* y mapearla con la vista real que visualizará el resultado. En el caso de *HomePageController* se necesita una vista que resuelva el nombre lógico "home" que es el que se retorna en el

objeto *ModelAndView*. Para que Spring puede utilizar una vista JSP es necesario utilizar el objeto *InternalResourceViewResolver* de Spring: [1]

```
<bean id="viewResolver"
class="org.springframework.web.
➥ servlet.view.InternalResourceViewResolver">
<property name="prefix">
<value>/WEB-INF/jsp/</value>
</property>
<property name="suffix">
<value>.jsp</value>
</property>
</bean>
```

El objeto *InternalResourceViewResolver* incluye dos propiedades que definen valores para el prefijo y sufijo. Una vez que HomePageController retorna le nombre de home en el objeto *ModelAndView*, *InternalResource-ViewResolver* encontrará una vista en */WEBINF/jsp/home.jsp*.

Finalmente, después de que el controlador ha sido creado y que se ha configurado los archivos de contexto, es necesario crear la vista que visualizará el resultado, debe ser creada bajo el directorio */WEBINF/jsp* como se definió en el archivo de configuración. A continuación se visualiza el código de la vista en un simple formato JSP:

```
<%@ page contentType="text/html" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<html>
<head><title>Rantz</title></head>
<body>
<h2>Welcome to RoadRantz!</h2>
<h3>Recent rantz:</h3>
<ul>
```

```
<c:forEach items="${rants}" var="rant">
<li><c:out value="${rant.vehicle.state}"/> /
<c:out value="${rant.vehicle.plateNumber}"/> --
<c:out value="${rant.rantText}"/>
</li>
</c:forEach>
</ul>
</body>
</html>
```

En el controlador *HomePageController* se retorno el objeto *ModelAndView* pasando como parámetro una colección de objetos a la vista. En este el archivo JSP (la vista) recibe como parámetro esa colección y visualiza cada uno de los elementos de la colección en el navegador de Internet.

CAPITULO III

DESARROLLO DE LA APLICACIÓN

3.1 Metodología de desarrollo

3.1.1 Programación orientada a objetos y aspectos

Spring es una arquitectura innatamente orientada a objetos, que garantiza y exige el empleo de este tipo de programación en todas las aplicaciones que se desarrollen con este framework.

Adicionalmente, Spring permite el empleo de la programación orientada a aspectos (a diferencia de la programación orientada a objetos, su uso es opcional en Spring) que es un esquema nuevo, que facilita al desarrollador enfocarse en la lógica de negocio y dejar otros temas que no se relacionan con esta fuera de su competencia.

Es importante remarcar que la programación orientada a objetos y la programación orientada a aspectos no son excluyentes, y que ambas pueden ser utilizadas en conjunto, ya que cada una tiene un propósito diferente.

3.1.2 Paradigma de desarrollo

Existen un sinnúmero de paradigmas que se pueden aplicar a los proyectos de ingeniería de software. Sin embargo, la selección de un paradigma o de otro depende de características como: el ámbito del proyecto, su complejidad, e incluso de los recursos disponibles.

Para el desarrollo de este proyecto, y considerando que la complejidad del sistema de evaluación del personal de la PUCESA no es demasiada amplia, se ha optado por emplear el paradigma de desarrollo en cascada con las etapas que se muestran en la Figura 3.1.

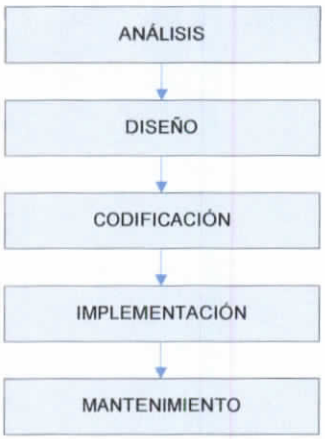


Figura 3.1 Etapas del paradigma utilizado

Aunque las etapas mostradas en la figura anterior constituyen las fases comúnmente usadas en este paradigma, es importante remarcar que las actividades que se realizan en cada etapa pueden variar dependiendo de la metodología que se utilice. Por ejemplo, las actividades de un proyecto que utiliza programación estructurada son diferentes a las actividades de un

proyecto que utiliza programación orientada a objetos. Incluso, considerando que dos proyectos utilizan una metodología orientada a objetos, las actividades de desarrollo del primer proyecto pueden ser diferentes a las del segundo proyecto, esto se fundamenta en el hecho que los dos proyectos pueden tener ámbitos diferentes. Por ejemplo, si el primer proyecto es un sistema distribuido que requiere la participación de varios componentes entre sí, requiere de la elaboración del diagrama de colaboración de componentes (un artefacto del Lenguaje de Modelamiento Unificado, UML pos sus siglas en inglés) en la fase de diseño. Sin embargo, el segundo proyecto que en un sistema que responde a eventos, necesita más un diagrama de estados que un diagrama de componentes.

Considerando todo lo anterior, se puede a continuación describir las macro actividades que se realizaron en cada una de las etapas. Estas actividades no pretenden ser una guía de desarrollo reconocida ni mucho menos estandarizada. Mas bien representan las actividades que a criterio personal son necesarias para un proyecto con las características que tiene el sistema de evaluación del personal de la PUCESA:

Análisis: En esta etapa se realizó el levantamiento de requerimientos, la formulación de las especificaciones funcionales, y la selección y justificación de las arquitecturas y plataformas idóneas para el desarrollo del sistema.

Diseño: En esta etapa se realizaron los diagramas de clases y el modelo físico de datos, la definición de los patrones de diseño a emplearse, y el diseño de las capas de la aplicación.

Codificación: En esta fase se realizó la configuración del proyecto Web y la configuración de los archivos descriptores, la codificación de las clases e interfaces, la generación del esquema de base de datos y la programación de las interfaces gráficas de usuario.

Implementación: Consiste en la puesta en operación del sistema de evaluación del personal Administrativo de la PUCESA en los ambientes de la Universidad.

Mantenimiento: Esta última fase está enfocada al mantenimiento de la aplicación, este mantenimiento puede responder a cambios que se basan en nuevos requerimientos, corrección de defectos, y otros.

3.2 Etapa de análisis

3.2.1 Levantamiento de requerimientos

La primera actividad que se realizó en esta etapa consistió en el levantamiento de requerimientos, para el efecto se realizó una reunión con el personal del Departamento de Recursos Humanos de la Universidad, quienes serán los usuarios del sistema. Se conoció el esquema actual de cómo se realiza la evaluación del personal. En esta reunión a más de definir las necesidades del sistema, se identificó los problemas que tienen con el proceso de evaluación actual, el cual es realizado manualmente, a continuación se detallan los mismos:

La gran cantidad de tiempo y recursos que se emplean.

La pérdida de información que puede ocasionarse al llevar registros manuales.

Las necesidades principales del sistema también fueron identificadas y se detallan a continuación:

Mantener un cronograma de las evaluaciones.

Incluir el proceso de evaluación a través de formatos de evaluación y encuestas.

Desplegar los resultados de las evaluaciones a través de pantallas y reportes.

3.2.2 Especificaciones funcionales

Una vez entendido las necesidades del sistema, se elaboró las especificaciones funcionales de la aplicación. Un buen artefacto para la especificación de funcionalidades son los casos de uso (diagrama UML). Sin embargo, considerando que este diagrama se elabora a partir de actores o roles del sistema y sus funciones, no es del todo aplicable al contexto de este proyecto, debido a que el sistema está enfocado a un solo tipo de usuario, el termino de roles o actores no es aplicable. Por lo tanto, se dio más importancia al hecho de identificar las funcionalidades que debe tener el sistema y no de quien debe hacerlas. Para este efecto, se elaboró una descripción detallada de las funciones del sistema.

3.2.2.1 Registro de cronograma

El sistema permitirá tener un registro de las actividades referentes a la evaluación del personal, similar a un cronograma. Dentro de este contexto es necesario registrar:

La hora y fecha de cuando la actividad o actividades serán realizadas.

El personal a ser evaluado: El personal debe ser seleccionable de los ingresados previamente en el sistema.

Lugar de la evaluación.

Actividad: descripción de la actividad.

Tipo de formato (seleccionable de los valores preestablecidos)

3.2.2.2 Formatos de evaluación

El sistema se utilizará para realizar los formatos de evaluación de observación y pruebas no se incluirá el de entrevistas. También se evaluará las encuestas de clima laboral y satisfacción del personal administrativo del área de Secretaría de la PUCESA. Los formatos de evaluación se elaboran para cada área de la universidad. Los formatos de evaluación deben incluir la siguiente información:

Información del personal evaluado: Nombre, cargo, fecha de ingreso, tipo de contrato.

Código del puesto

Unidad

Área (seleccionable de los valores preestablecidos)

Evaluador (usuario)

Fecha de evaluación

Niveles de calificación

Factores

Subfactores

Calificación por subfactor

Observaciones

Tipo de formato (seleccionable de los valores preestablecidos)

3.2.2.3 Formato de observación

Para el formato de observación se tienen las siguientes consideraciones:

Los factores deben ser parametrizables y deben vincularse a un área.
Los subfactores o preguntas deben ser parametrizables y deben vincularse a un factor.

Los niveles de calificación deben ser parametrizables, los niveles de calificación pueden tener diferentes valores para cada tipo de encuesta o formato de evaluación. Por ejemplo: para un caso estos valores pueden incluirse entre un rango de uno a cinco (observación), paro otro caso los valores pueden incluirse entre un rango de uno a cien (porcentajes para prácticas). Sin embargo, para el proceso de cuantificación del sistema, estos niveles de calificación deben ser solamente numéricos. En el formato

observación los niveles de calificación aplican para todos los factores o subfactores. Por ejemplo: se aplicaría una escala de 1 a 5 para todos los factores. También es necesario que el sistema permita registrar el plan de mejora del evaluado (compromiso), e incluso razones de inconformidad si las hay.

3.2.2.4 Formatos de prácticas

La diferencia con el formato de observación es que para cada factor existe en su proceso de cuantificación diferente que incluye un porcentaje ideal y el porcentaje obtenido por el evaluado. Por ejemplo, para un factor el porcentaje ideal puede ser un 30 por ciento y el obtenido por el evaluado es de 23 por ciento. Para otro factor el porcentaje ideal puede ser un 20 porciento y el porcentaje obtenido por el evaluado es de 15 por ciento.

3.2.2.5 Encuestas

Las encuestas tienen la misma estructura de los formatos de observación, es decir están constituidas por factores y subfactores, y estos deben tener los mismos niveles de cuantificación. El sistema debe permitir realizar las siguientes encuestas:

Clima laboral

Satisfacción personal administrativo (Secretaría).

3.2.2.6 Proceso de cuantificación

El sistema realizará automáticamente los procesos de cuantificación de la evaluación que se haya realizado sobre el personal de Secretaría de la PUCESA, este proceso es ejecutado independientemente para cada persona evaluada, basado en la información registrada en los formatos de evaluación o encuestas, para este proceso es necesario que en el sistema se registren los niveles de calificación.

3.2.2.7 Parametrización de entidades generales

Dentro de las entidades generales que se necesitan y que el sistema debe permitir parametrizar se encuentran:

Personal con la siguiente información: Nombres, apellidos, fecha de ingreso, cargo (seleccionable), unidad, área (seleccionable), tipo de contrato (relación de dependencia y horas).

Cargos del personal con la información: Nombre, descripción.

Áreas de la universidad con la información: Nombre, descripción. Este define el departamento que el personal tendrá dentro del sistema.

Tipos de formato: Incluye todos los tipos de formato (observación, pruebas, encuestas de clima laboral, satisfacción de personal administrativo).

3.2.2.8 Usuarios del sistema

Los procesos de administración y configuración funcional del sistema serán realizados exclusivamente por el personal del Departamento Administrativo de la PUCESA. Es este departamento únicamente el encargado de definir, parametrizar y configurar los formatos de evaluación, los factores y subfactores, las encuestas, y las entidades generales. La actividad de esta área también se extiende al registro de los formatos de evaluación, procesos de cuantificación, la generación de reportes y estadísticas, y toda actividad concerniente a la administración y configuración del sistema.

Las encuestas del clima laboral y las encuestas de satisfacción del personal administrativo pueden ser registradas directamente en el sistema por el personal que está siendo evaluado, que para el ámbito de este proyecto es el personal del Área de Secretaría.

3.2.2.9 Reportes y estadísticas

El sistema debe permitir la generación de reportes y estadísticas que visualicen los resultados obtenidos en las evaluaciones. Dentro de los reportes necesarios se incluyen:

Reporte de evaluación de las observaciones: Incluye gráficos de barra por subfactores con las estadísticas de los resultados.

Reporte de evaluación de las pruebas (prácticas): incluye gráficos de barra por subfactores con las estadísticas de los resultados.

Reporte de los cronogramas agrupados por fecha.

Histórico de todas las evaluaciones realizadas al personal. Cada evaluación se distingue por la fecha de realización, lo que ayuda a la generación de este reporte.

3.2.3 Selección de las arquitecturas y plataformas de desarrollo

La última actividad que se realizó en esta etapa consistió en la selección de las arquitecturas y plataformas que se emplearon en el desarrollo del sistema de evaluación del personal de la PUCESA. Para esta selección se tomaron en cuenta diferentes consideraciones, que se describen a continuación.

3.2.3.1 Framework de desarrollo empresarial

El desarrollo de aplicaciones empresariales requiere el empleo de frameworks que permitan la construcción de aplicaciones escalables y fáciles de mantener. La selección de Spring como framework de desarrollo empresarial se justifica en el hecho de que es uno de los únicos frameworks que se preocupa de todos los aspectos concernientes al desarrollo de sistemas empresariales. Spring proporciona un amplio abanico de posibilidades que incluyen: Manejos de transacciones, mensajería, seguridad, administración de servicios persistentes, administración de

servicios remotos, por mencionar algunos. A continuación se delinean algunas de las características principales que sentaron las bases para emprender el estudio de Spring[1] como tema de este trabajo de grado:

Permite la construcción de aplicaciones empresariales escalables, que son fáciles de mantener y testear. Además, Spring no requiere de una alta complejidad para su empleo, esto con relación a las especificaciones J2EE.

Spring provee una arquitectura de desarrollo bien estructurada que está basada en patrones de diseño, que facilita: la reutilización de código, el desarrollo multicapas, y la libre elección del resto de plataformas de desarrollo.

Spring es un framework muy difundido actualmente, tiene disponible una amplia documentación en el sitio web oficial y un gran número de foros en Internet. Hoy en día Spring es considerado como un framework facto estándar (oficialmente no reconocido pero ampliamente difundido y utilizado).

Spring es un framework gratuito y de código abierto. Esto permite no incurrir en gastos para su adquisición y además, de ser necesario, consultar su estructura directamente en su código.

---

[1] En el primer capítulo de este documento se encuentra una amplia información que explica el porqué Spring es un framework ideal para el desarrollo de aplicaciones empresariales. En el segundo capítulo se habla explícitamente de los beneficios y de las características de Spring.

3.2.3.2 Lenguaje de programación

Teniendo en cuenta la premisa de utilizar Spring, la selección del lenguaje de programación está delimitada considerablemente. Concretamente, existen dos plataformas de programación que son soportadas por Spring, la primera de ellas es Java y la segunda Microsoft .NET. Por consiguiente, la selección del lenguaje radica en cualquiera de estas dos tecnologías.

No obstante, definir que plataforma de programación se adapta mejor al desarrollo de aplicaciones empresariales con Spring esta fuera del contexto e interés de este documento, ya que ambas plataformas son realmente aplicables al desarrollo de este tipo de proyectos y ambas funcionan perfectamente . Incluso, teniendo en cuenta que los conceptos de Spring son generales e independientes del lenguaje de programación, un cambio de plataforma de programación podría ser menos impactante. Sin embargo hay ciertos detalles que definieron la selección del lenguaje para este proyecto, los cuales se describen en la Tabla 3.1.

Condiciones	Java	.NET
Licenciamiento	Gratuito, no incurre en gastos de licenciamiento.	Es un producto comercial.
Difusión	Es el lenguaje de mayor uso a nivel mundial.	Tiene una difusión considerablemente menor con respecto a Java.
Integración de Spring	Spring esta desarrollado en Java, por lo que su código fuente es depurable en un proyecto Java.	No se puede depurar código de Spring en un proyecto de .NET.

Tabla 3.1 Tabla comparativa de los lenguajes soportados por Spring

Todo lo expuesto anteriormente, justificó la selección de Java como lenguaje de programación idóneo para el desarrollo del sistema de evaluación del personal administrativo de la PUCESA.

3.2.3.3 Framework de presentación

La flexibilidad de Spring se basa en el hecho que permite al desarrollador la selección de un framework de presentación de una gran variedad de frameworks soportados. Por ejemplo, soporta los siguientes frameworks para la capa de presentación: Struts, JSF, Velocity, MVC de Spring, ZK entre otros.

Para la generación de la capa de presentación de la aplicación se utilizo el framework ZK. Las consideraciones que definieron la selección de este framework son:

ZK es un framework que permite la construcción de presentaciones ricas de Internet (RIA, por sus siglas en inglés) y se basa en eventos AJAX (Asynchronous JavaScript and XML, nombre original en inglés).

ZK es un framework que utiliza programación declarativa y se estructura utilizando el Patrón MVC.

ZK permite la creación de interfaces gráficas de usuario de una manera simple y rápida, ya que se utiliza Java para el manejo de las solicitudes que son realizadas por los clientes a través de un navegador Web. Esto se realiza a través de un mecanismo denominado "codebehind".

3.2.3.4 Herramienta ORM

Anteriormente se hablo de la flexibilidad de Spring para permitir la selección de cualquier framework para la capa de presentación. Esta flexibilidad también se extiende a la selección de una herramienta o framework ORM. Por ejemplo, Spring soporta los siguientes ORMs: Hibernate, Top Link, IBatis, entre otros.

El ORM que se empleo para la construcción del sistema es Hibernate, este framework permite aproximadamente un ahorro del noventa y cinco por ciento de código enfocado a la manipulación de los datos en una base.

Algunas de las caracteristicas que destacan a Hibernate y su empleo en la elaboración del sistema de evaluación del personal de la PUCESA son:

Hibernate puede trabajar con cualquier motor de base de datos, y el cambio de un motor de base de datos a otro ya no es una actividad impactante.
Hibernate incluye su propio lenguaje para consulta de datos (SQL, por sus siglas en inglés) llamado HQL (Hibernate Query Language) que es similar al lenguaje SQL pero orientado a objetos.
Hibernate es actualmente el ORM más empleado en la plataforma Java, su difusión es tan amplia que es considerado un estándar dentro de las especificaciones J2EE.

Hibernate permite el empleo de anotaciones que facilita el mapeo de las clases con las tablas de la base, evitando la generación de los complicados archivos hbm de mapeo.

3.2.3.5 Motor de base de datos

La selección del motor de base de datos es libre dentro de la arquitectura empleada, la cual que incluye Spring y Hibernate. Existe soporte para una gran diversidad de motores de base de datos. Por ejemplo: Oracle, DB2, MySQL, PostgreSQL, Informix, Sybase, Progress, Microsoft SQL Server, entre otros.

Para este proyecto la selección dependió de la familiaridad que se tiene con Oracle, además considerando que Oracle es una de los motores de datos más robustos, completos y que presenta un buen desempeño, la selección está más que justificada.

Sin embargo, se puede pensar que existe el inconveniente de licenciamiento de Oracle, ya que es un producto comercial, pero el licenciamiento de Oracle se enfoca a la versión que se emplee, existiendo versiones gratuitas como la versión Express, que fue la versión empleada en la implementación del sistema. Además, todas las versiones de Oracle, incluyendo las empresariales, son libres para fines de desarrollo.

3.2.3.6 Herramientas adicionales

Finalmente, esta la selección del entorno de desarrollo utilizado que al igual que la selección del motor de base de datos se fundamento en la familiaridad que se tiene de las herramientas. A continuación se detalla las herramientas empleadas y su funcionalidad:

Eclipse: Entorno o plataforma de desarrollo gratuita que permite la creación de proyectos Java. Además, incluye plugins gratuitos para la creación de modelos de diseño.

Apache Tomcat: Servidor de aplicaciones gratuito que permite la publicación de las aplicaciones Java realizadas para Internet.

3.3 Etapa de diseño

3.3.1 Estructura en capas del proyecto

El sistema de evaluación del personal de la PUCESA es una aplicación que está diseñada en capas. Esto facilita el mantenimiento de cualquier aplicación en términos generales, y la hace más flexible y escalable. Por ejemplo, suponiendo que una aplicación está diseñada inicialmente para la Web, sería suficiente una capa de presentación que se encargue de controlar los requerimientos de los clientes, los cuales son realizados a través de un navegador de Internet, y los comunique a la capa de negocios

(también conocida como capa lógica). Posteriormente se considera necesario ampliar la funcionalidad de la aplicación para dispositivos móviles inteligentes, si la capa de negocios de la aplicación está exponiendo su funcionalidad a través de servicios, lo único necesario es crear otra capa de presentación para dispositivos móviles que solicite estos servicios a la capa de negocios, consiguiendo con esto que las capa de negocios como el resto de capas subsecuentes no se vean afectadas con esta ampliación a la funcionalidad.

La Figura 3.2 visualiza la estructura básica de cómo las capas están constituidas en la aplicación de evaluación del personal de la PUCESA.

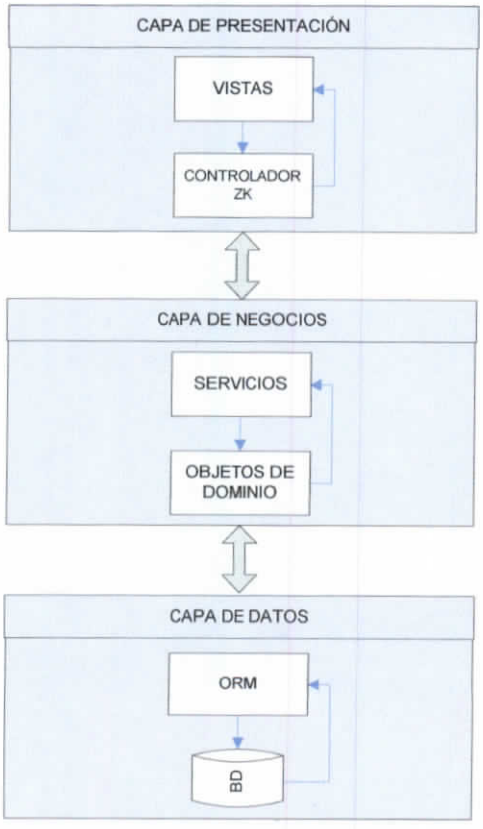


Figura 3.2 Estructura de las capas de la aplicación

La capa de presentación está constituida por todas las vistas, que no son más que simplemente archivos con programación declarativa que contienen todos los controles visuales. Estos archivos de configuración tienen una extensión ".zul" que es un requerimiento de ZK.

ZK también incluyen los controladores que son clases de Java que implementan toda la funcionalidad de este framework. Los controladores al ser clases propias de Java no requieren el empleo de llamadas remotas (RPC, por sus siglas en inglés) coma la mayoría de los frameworks AJAX (Asynchronous JavaScript and XML, nombre original en inglés). Por lo tanto, las llamadas a los servicios que expone la capa de negocios son meramente llamadas a métodos Java de otras clases.

La capa de negocios expone los servicios que consumirá la capa de presentación, estos servicios son normalmente interfaces que definen el contrato del servicio más que la implementación. Esto permite tener varias implementaciones de un servicio y utilizar la implementación más adecuada basado posiblemente en un archivo de configuración de Spring (Patrón de diseño inversión de control, IOC por sus siglas en inglés).

La capa de negocios también incluye los objetos de dominio, o mejor definidas como las clases de dominio. Las clases de dominio constituyen normalmente: las clases persistentes, las clases administradoras (Factory, por su término en inglés) y las implementaciones de las interfaces de servicios.

Por último, la capa de datos está integrada por el framework Hibernate que es el ORM encargado de traducir las operaciones de manipulación de datos que se realizan sobre los objetos persistentes de la aplicación a los registros de la base de datos.

3.3.2 Estructura de los paquetes

La Figura 3.3 visualiza la jerarquía y la estructura de los paquetes del proyecto. La nomenclatura empleada para nombrar los paquetes es la considerada estándar dentro de la plataforma Java.

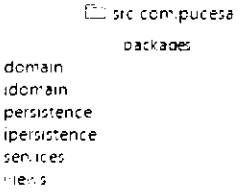


Figura 3.3 Estructura de paquetes

El paquete considerado la raíz está identificado como "src.com.pucesa", este paquete a su vez contiene los siguientes sub-paquetes:

- domain: En este paquete se ubican todas las clases que proveen los servicios de negocios de la aplicación (para este tipo de proyecto las clases de negocios son muy escasas). Adicionalmente, también se ubican aquí las clases persistentes y las implementaciones de las interfaces de negocio.

domain: Este paquete incluya todas la interfaces que definen las reglas de negocios. Un ejemplo dentro del sistema de evaluación lo constituye la interfaz que devuelve el resultado de una cotización.

persistence: Este paquete incluye la clase o clases que implementa la interfaz de persistencia, es decir la interfaz que contiene la firma de los métodos comunes de operaciones persistentes, como por ejemplo: insertar, actualizar, eliminar y buscar.

ipersistence: Este paquete incluye la interfaz que firma los métodos de las operaciones persistentes.

services: Este paquete incluye los servicios generales tanto para los objetos de la capa de negocios como para los objetos de la capa de presentación. Normalmente, las clases que incluye este paquete utilizan el patrón de diseño Singleton.

views: Este paquete incluye las clases de la capa de presentación, es decir los controladores de vistas. Estas clases son las encargadas de dirigir los requerimientos que son realizados en la capa de presentación a la capa de negocios y a su vez de retornar el resultado de la capa de negocios a la capa de presentación.

3.3.3 Diagrama de clases

En la Figura 3.4 se muestra las clases del sistema y la relación entre ellas, para el efecto se emplea UML.



Figura 3.4 Diagrama de clases

La Figura 3.5 incluye el diagrama de la clase Departamento, utilizada para representar las áreas de la universidad.

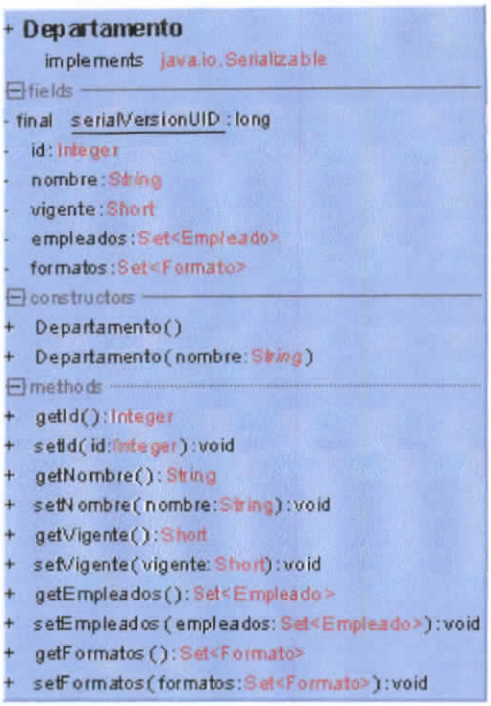


Figura 3.5 Diagrama de la clase departamento

La Figura 3.6 incluye el diagrama de la clase Cargo, utilizada para representar los cargos de los empleados de la universidad.

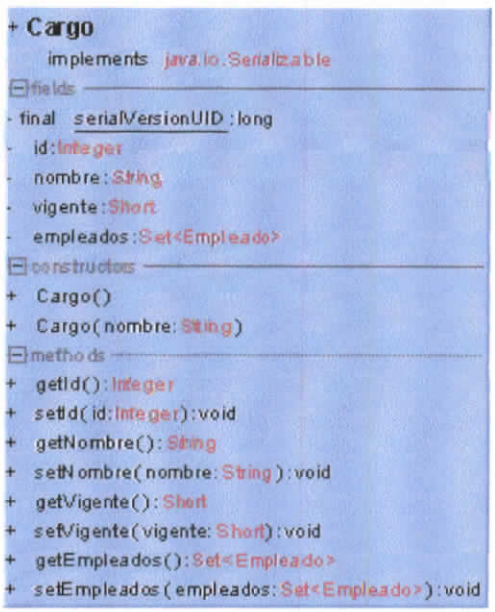


Figura 3.6 Diagrama de la clase departamento

La Figura 3.7 incluye el diagrama de las clases: Entidad, Empleado y Usuario con sus respectivas relaciones.

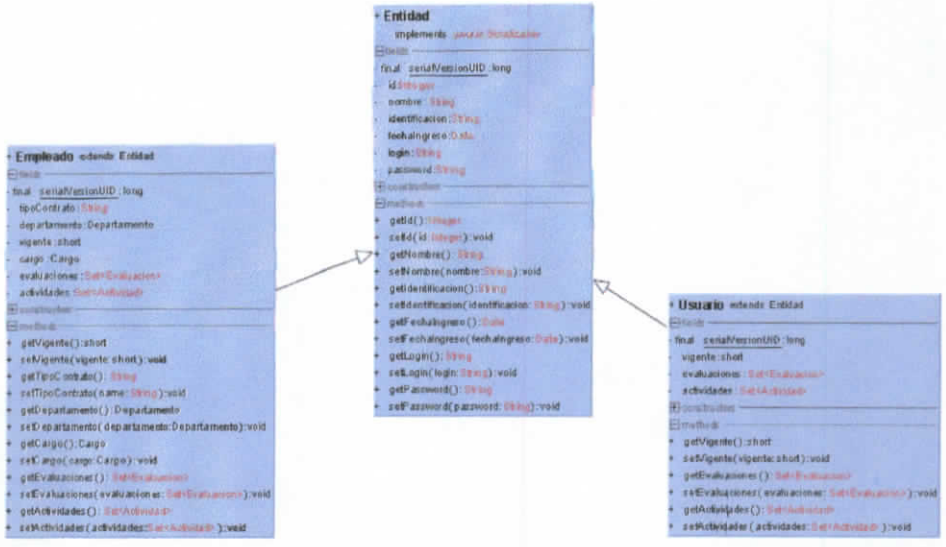


Figura 3.7 Jerarquía de las clases de entidad

La Figura 3.8 incluye el diagrama de la clase Tipo Formato, utilizada para representar la tipificación de los formatos de evaluación de la universidad.

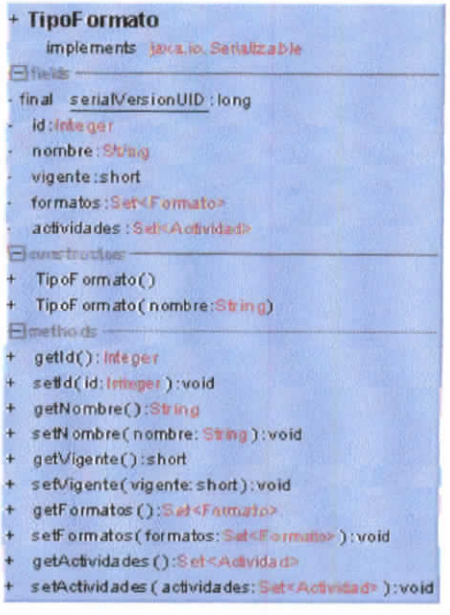


Figura 3.8 Diagrama de la clase tipo formato

La Figura 3.9 incluye el diagrama de la clase Formato, utilizada para representar los formatos de evaluación de la universidad.

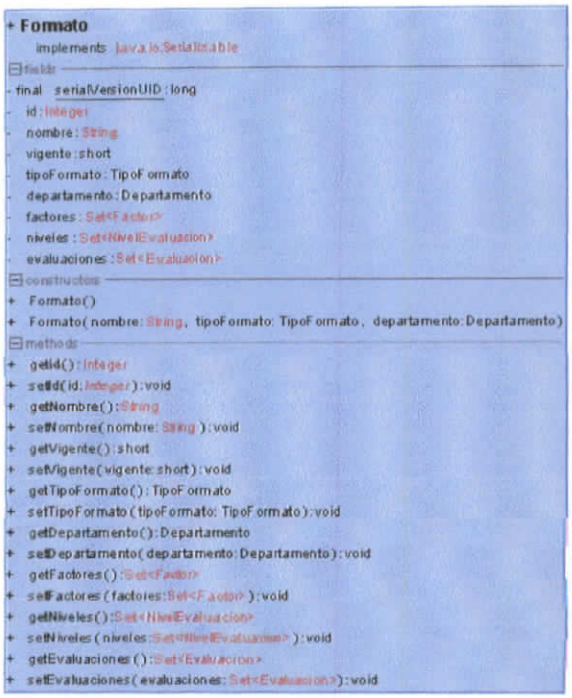


Figura 3.9 Diagrama de la clase formato

La Figura 3.10 incluye el diagrama de la clase Factor, utilizada para representar los factores empleados en las evaluaciones.

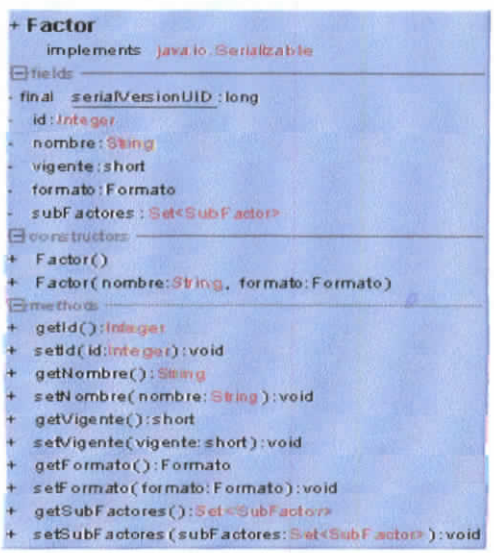


Figura 3.10 Diagrama de la clase factor

La Figura 3.11 incluye el diagrama de la clase Subfactor, utilizada para representar los subfactores empleados en las evaluaciones.

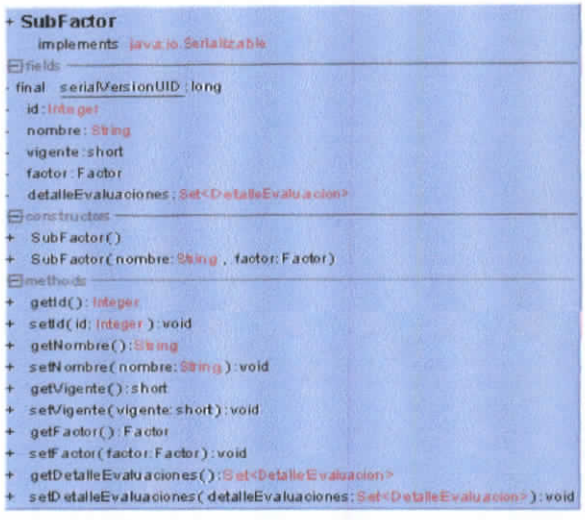


Figura 3.11 Diagrama de la clase de subfactor

La Figura 3.12 incluye el diagrama de la clase Nivel, utilizada para representar los niveles de evaluación.

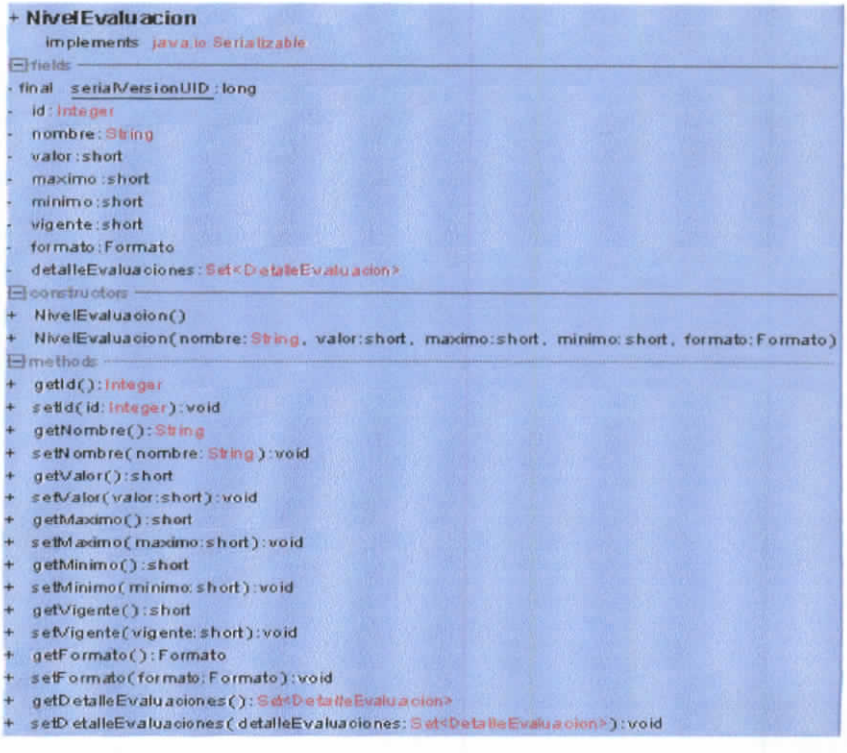


Figura 3.12 Diagrama de la clase Nivel de evaluación

La Figura 3.13 incluye la relación de jerarquía entre las clases: Evaluación y Detalle de evaluación.



Figura 3.13 Diagrama de la jerarquía de las clases de evaluación

La Figura 3.14 incluye el diagrama de la clase Actividad, empleada para el cronograma de actividades.

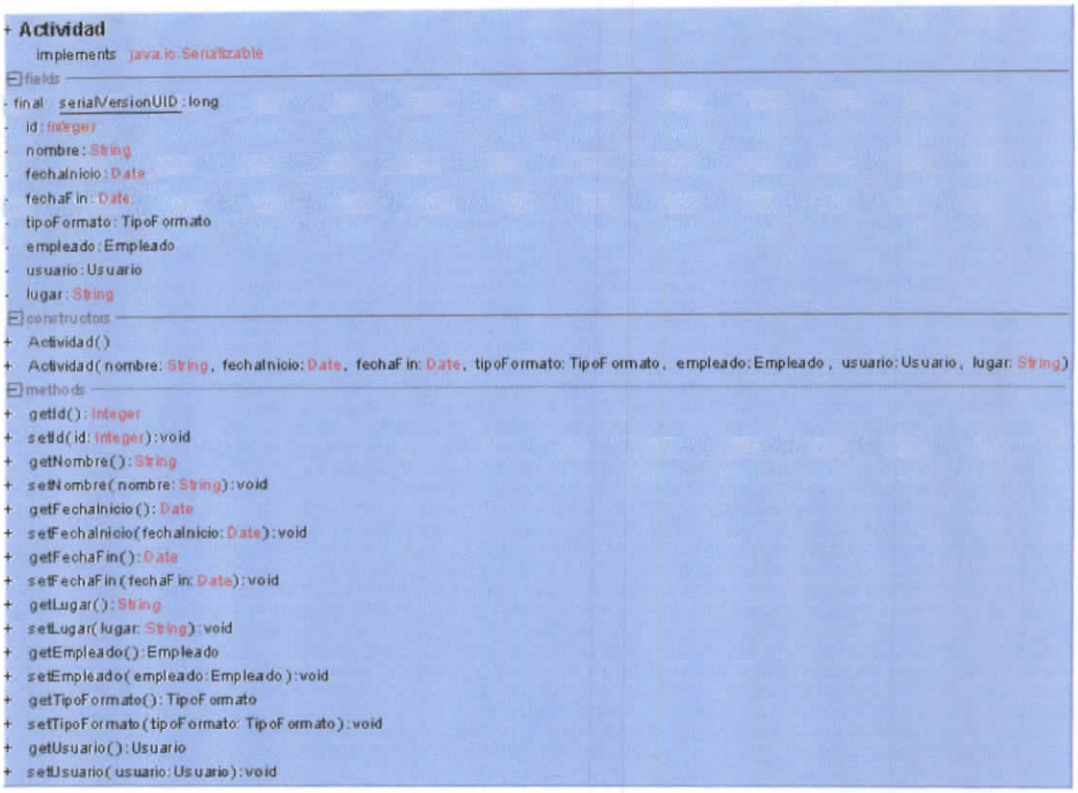


Figura 3.14 Diagrama de la clase actividad

La Figura 3.15 incluye el diagrama de la interfaz empleada en la definición de los métodos de persistencia.

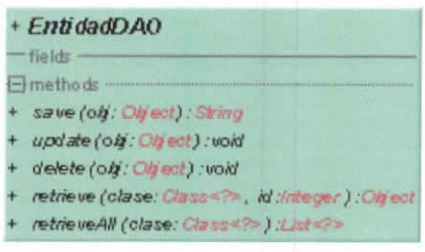


Figura 3.15 Diagrama de la interfaz de persistencia

La Figura 3.16 incluye el diagrama que implementa la interfaz de definición de los métodos de persistencia y sus relaciones.

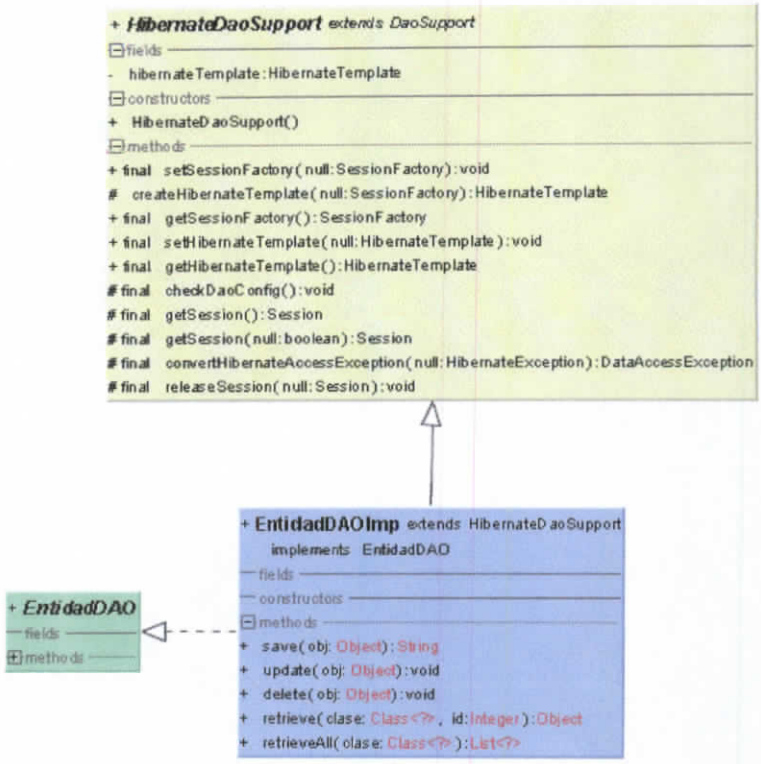


Figura 3.16 Diagrama de la clase de implementación de persistencia

La Figura 3.17 incluye el diagrama de una clase de vista de ZK que utiliza el servicio de inicialización de contexto para Spring.

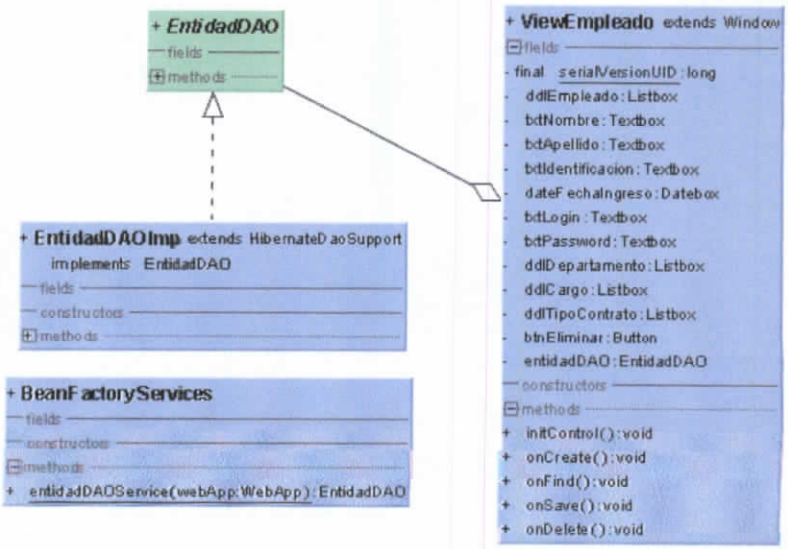


Figura 3.17 Diagrama de una clase vista de ZK y sus relaciones

3.3.4 Diagrama físico de datos

La Figura 3.18 muestra el diagrama físico de datos, que contiene todas las tablas de la aplicación y sus relaciones.

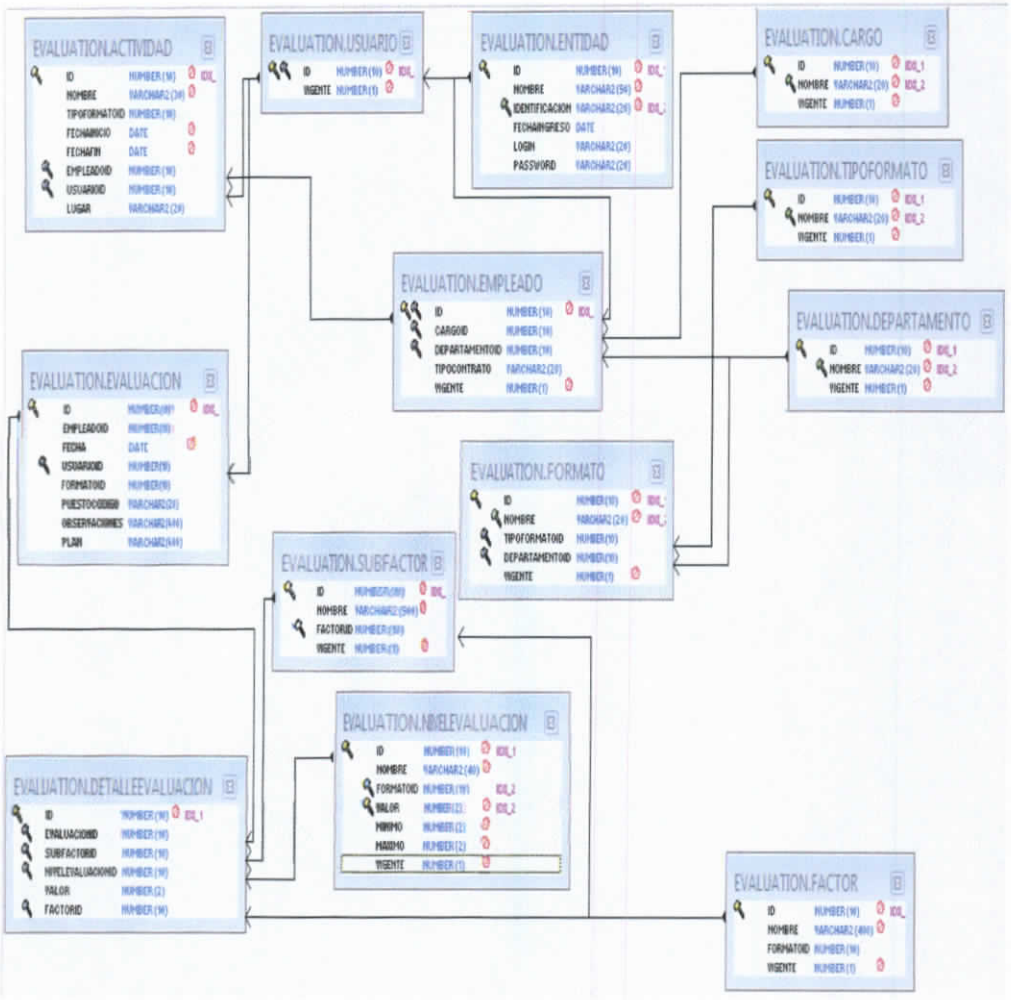


Figura 3.18 Modelo Físico de Datos

La Tabla 3.2 tiene el detalle de los campos que se emplean en la tabla Departamento, esta tabla es utilizada para almacenar la información referente a los departamentos de la universidad.

Campos	Tipo	Descripción
ID	NUMBER(10)	Clave principal de la tabla.
NOMBRE	VARCHAR2(20) NOT NULL	El nombre del departamento, campo requerido.
VIGENTE	NUMBER(1) DEFAULT 1 NOT NULL	Estado lógico que determina si el registro es activo.

Tabla 3.2 Diccionario de datos de la tabla Departamento

La Tabla 3.3 tiene el detalle de los campos que se emplean en la tabla Cargo, esta tabla es utilizada para almacenar la información referente a los cargos de los empleados.

Campos	Tipo	Descripción
ID	NUMBER(10)	Clave principal de la tabla.
NOMBRE	VARCHAR2(20) NOT NULL	El nombre del cargo, campo requerido.
VIGENTE	NUMBER(1) DEFAULT 1 NOT NULL	Estado lógico que determina si el registro es activo.

Tabla 3.3 Diccionario de datos de la tabla Cargo

La Tabla 3.4 tiene el detalle de los campos que se emplean en la tabla Entidad, esta tabla es utilizada para almacenar los datos de la clase base para entidades.

Campos	Tipo	Descripción
ID	NUMBER(10)	Clave principal de la tabla.
NOMBRE	VARCHAR2(50) NOT NULL	El nombre y apellido de todas las entidades.
IDENTIFICACION	VARCHAR2(20) NOT NULL	La identificación de las entidades.
FECHAINGRESO	DATE	Fecha de ingreso para empleados.
LOGIN	VARCHAR2(20)	El login para control de acceso de las entidades.
PASSWORD	VARCHAR2(20)	El password para control de acceso de las entidades.

Tabla 3.4 Diccionario de datos de la tabla Entidad

La Tabla 3.5 tiene el detalle de los campos que se emplean en la tabla Empleado, esta tabla es utilizada para almacenar los datos de la clase hija Empleado.

Campos	Tipo	Descripción
ID	NUMBER(10)	Clave principal de la tabla.
CARGOID	NUMBER(10)	Clave foránea para relación con la tabla cargo.
DEPARTAMENTOID	NUMBER(10)	Clave foránea para relación con la tabla departamento.
TIPOCONTRATO	VARCHAR2(20)	El tipo contrato de los empleados.
VIGENTE	NUMBER(1) DEFAULT 1 NOT NULL	Estado lógico que determina si el registro es activo.

Tabla 3.5 Diccionario de datos de la tabla Empleado

La Tabla 3.6 tiene el detalle de los campos que se emplean en la tabla Usuario, esta tabla es utilizada para almacenar los datos de la clase hija Usuario.

Campos	Tipo	Descripción
ID	NUMBER(10)	Clave principal de la tabla.
VIGENTE	NUMBER(1) DEFAULT 1 NOT NULL	Estado lógico que determina si el registro es activo.

Tabla 3.6 Diccionario de datos de la tabla Usuario

La Tabla 3.7 tiene el detalle de los campos que se emplean en la tabla TipoFormato, esta tabla es utilizada para almacenar la información referente a los tipos de formato de evaluación.

Campos	Tipo	Descripción
ID	NUMBER(10)	Clave principal de la tabla.
NOMBRE	VARCHAR2(20) NOT NULL	El nombre del tipo de formato.
VIGENTE	NUMBER(1) DEFAULT 1 NOT NULL	Estado lógico que determina si el registro es activo.

Tabla 3.7 Diccionario de datos de la tabla TipoFormato

La Tabla 3.8 tiene el detalle de los campos que se emplean en la tabla Formato, esta tabla es utilizada para almacenar la información referente a los formatos de evaluación.

Campos	Tipo	Descripción
ID	NUMBER(10)	Clave principal de la tabla.
NOMBRE	VARCHAR2(20) NOT NULL	El nombre del formato.
TIPOFORMATOID	NUMBER(10)	Clave foránea para relación con la tabla tipoformato.
DEPARTAMENTOID	NUMBER(10)	Clave foránea para relación con la tabla departamento.
VIGENTE	NUMBER(1) DEFAULT 1 NOT NULL	Estado lógico que determina si el registro es activo.

Tabla 3.8 Diccionario de datos de la tabla Formato

La Tabla 3.9 tiene el detalle de los campos que se emplean en la tabla Factor, esta tabla es utilizada para almacenar la información referente a los factores de evaluación.

Campos	Tipo	Descripción
ID	NUMBER(10)	Clave principal de la tabla.
NOMBRE	VARCHAR2(400) NOT NULL	El nombre del factor de evaluación.
FORMATOID	NUMBER(10)	Clave foránea para relación con la tabla Formato.
VIGENTE	NUMBER(1) DEFAULT 1 NOT NULL	Estado lógico que determina si el registro es activo.

Tabla 3.9 Diccionario de datos de la tabla Factor

La Tabla 3.10 tiene el detalle de los campos que se emplean en la tabla Subfactor, esta tabla es utilizada para almacenar la información referente a los subfactores de evaluación.

Campos	Tipo	Descripción
ID	NUMBER(10)	Clave principal de la tabla.
NOMBRE	VARCHAR2(400) NOT NULL	El nombre del Subfactor de evaluación.
FACTORID	NUMBER(10)	Clave foránea para relación con la tabla Factor.
VIGENTE	NUMBER(1) DEFAULT 1 NOT NULL	Estado lógico que determina si el registro es activo.

Tabla 3.10 Diccionario de datos de la tabla Subfactor

La Tabla 3.11 tiene el detalle de los campos que se emplean en la tabla NivelEvaluacion, esta tabla es utilizada para almacenar la información referente a los niveles de evaluación.

Campos	Tipo	Descripción
ID	NUMBER(10)	Clave principal de la tabla.
NOMBRE	VARCHAR2(40) NOT NULL	Nombre del nivel de evaluación.
FORMATOID	NUMBER(10)	Clave foránea para relación con la tabla Formato.
VALOR	NUMBER(2) NOT NULL	Valor de calificación.
MINIMO	NUMBER(2) NOT NULL	Valor mínimo de calificación.
MAXIMO	NUMBER(2) NOT NULL	Valor máximo de calificación.
VIGENTE	NUMBER(1) DEFAULT 1 NOT NULL	Estado lógico que determina si el registro es activo.

Tabla 3.11 Diccionario de datos de la tabla NivelesEvaluacion

La Tabla 3.12 tiene el detalle de los campos que se emplean en la tabla Evaluacion, esta tabla es utilizada para almacenar la información de todas las evaluaciones que se realizan en el sistema.

Campos	Tipo	Descripción
ID	NUMBER(10)	Clave principal de la tabla.
EMPLEADOID	NUMBER(10)	Clave foránea para relación con la tabla Empleado.
FECHA	DATE NOT NULL	La fecha de elaboración de la cotización.
USUARIOID	NUMBER(10)	Clave foránea para relación con la tabla Usuario.
FORMATOID	NUMBER(10)	Clave foránea para relación con la tabla Formato.
PUESTOCODIGO	VARCHAR2(20)	El código del puesto de trabajo.
OBSERVACIONES	VARCHAR2(600)	Las observaciones de las evaluaciones realizadas.
PLAN	VARCHAR2(600)	El plan de cambio para las evaluaciones realizadas.

Tabla 3.12 Diccionario de datos de la tabla Evaluacion

La Tabla 3.13 tiene el detalle de los campos que se emplean en la tabla DetalleEvaluacion, esta tabla es utilizada para almacenar el detalle de las evaluaciones realizadas.

Campos	Tipo	Descripción
ID	NUMBER(10)	Clave principal de la tabla.
EVALUACIONID	NUMBER(10)	Clave foránea para relación con la tabla Evaluación.
SUBFACTORID	NUMBER(10)	Clave foránea para relación con la tabla SubFactor.
NIVELEVALUACIONID	NUMBER(10)	Clave foránea para relación con la tabla NivelEvaluación.
VALOR	NUMBER(2)	Valor aplicado en la evaluación.
FACTORID	NUMBER(10)	Clave foránea para relación con la tabla Factor, para el formato de práticas.

Tabla 3.13 Diccionario de datos de la tabla DetalleEvaluacion

La Tabla 3.14 tiene el detalle de los campos que se emplean en la tabla Actividad, esta tabla es utilizada para almacenar el cronograma de actividades relacionadas a la evaluación.

Campos	Tipo	Descripción
ID	NUMBER(10)	Clave principal de la tabla.
NOMBRE	VARCHAR2(50) NOT NULL	La descripción de la actividad.
TIPOFORMATOID	NUMBER(10)	Clave foránea para relación con la tabla TipoFormato.
FECHAINICIO	DATE NOT NULL	Fecha de inicio de la actividad.
FECHAFIN	DATE NOT NULL	Fecha de finalización de la actividad.
EMPLEADOID	NUMBER(10)	Clave foránea para relación con la tabla Empleado.
USUARIOID	NUMBER(10)	Clave foránea para relación con la tabla Usuario.
LUGAR	VARCHAR2(50)	Lugar dónde ser realiza la actividad de evaluación.

Tabla 3.14 Diccionario de datos de la tabla Actividad

3.3.5 Definición de los patrones de diseño

En transcurso del desarrollo del sistema de evaluación del personal de la PUCESA, se emplearon los siguientes patrones de diseño.

3.3.5.1 Modelo Vista Controlador

Modelo Vista Controlador (MVC, por sus siglas en inglés) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El primer componente es la vista que en el sistema la constituye el archivo declarativo de ZK empleado para la interfaz gráfica de usuario. El segundo componente es el modelo, el que se encarga de acceder a la lógica de la capa de negocios, en el sistema las clases de vista. Y por último el controlador, que es el responsable de recibir los eventos de entrada desde la vista (implementación propia de ZK).

3.3.5.2 Inversión de control

Inversión de control (IoC, por sus siglas en inglés) es un concepto junto a unas técnicas de programación en las que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales, en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos o funciones. En el sistema se puede especificar respuestas

a determinados eventos que se dan durante todo el ciclo de vida de la aplicación mediante llamadas a funciones, dejando que Spring lleve a cabo las acciones de control que se requieran en el orden necesario y para el conjunto de sucesos que tengan que ocurrir. Un ejemplo de inversión de control en el sistema basado en Spring se muestra a continuación:

```
<bean id="entidadDAO" class="com.pucesa.persistence.EntidadDAOImp">
        <property name="sessionFactory">
            <ref local="sessionFactory"/>
        </property>
</bean>
```

Mediante un archivo de configuración se le indica a Spring que implementación debe utilizar para una interfaz. En el ejemplo la interfaz *entidadDAO* será atada a la implementación que se encuentra en la clase *EntidadDAOImp*. Sin embargo, todas las referencias que se realizan en el sistema son a la interfaz y no directamente a la implementación, por lo que la implementación puede ser cambiada sin ningún impacto.

3.3.5.3 Singleton

El patrón de diseño singleton (instancia única) está diseñado para delimitar la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su propósito es garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. Un ejemplo de singleton en el sistema lo constituye la clase BeanFactoryServices que devuelve un método que es inicializado una sola vez creando un punto de acceso común para el contexto de la aplicación:

```
public class BeanFactoryServices {

        public static EntidadDAO entidadDAOService(WebApp webApp) {
            ServletContext servletContext = (ServletContext) webApp
                                .getNativeContext();
            BeanFactory beanFactory = WebApplicationContextUtils

        .getRequiredWebApplicationContext(servletContext);
            return (EntidadDAO) beanFactory.getBean("entidadDAO");
        }
}
```

3.3.5.4 Método de plantilla

Aunque este es un patrón que no es utilizado directamente por el código del sistema, es la esencia de trabajo de Spring con Hibernate para la generación de las operaciones persistentes. Este es un patrón de diseño que define una estructura algorítmica en la súper clase, delegando la implementación a las subclases. Es decir, define una serie de pasos, en donde los pasos serán redefinidos en las subclases.

Usando el método de plantilla (Template Method, por su término en inglés), se define una estructura de herencia en la cual la superclase sirve de plantilla de los métodos en las subclases, evitando la repetición de código. En el siguiente código se muestra como Spring hace uso de este patrón para definir los métodos de persistencia que son comunes para todos los objetos de la aplicación, en este caso una búsqueda de uso general.

```
public List<?> retrieveAll(Class<?> clase) {
List<?> list = super.getHibernateTemplate().find("from "+clase.getName());
        return list;
```

```
}
```

Este método se encuentra ubicado en la implementación de la interfaz *EntidadDAO* que es la clase *EntidadDAOImp.*

3.3.6 Diseño de entradas

Las interfaces gráficas de usuario (GUI por sus siglas en inglés) se diseñaron con el objetivo de facilitar la manipulación del sistema por parte de los usuarios. Estas entradas utilizan un esquema de tablas HTML y de estilos en cascada (CSS por sus siglas en ingles) que son aspectos inherentes en las aplicaciones Web.

Se diseño una GUI de plantilla que incluya los aspectos visuales comunes para la aplicación, estos incluye: logotipos, menú de navegación, colores y áreas de trabajo. La figura 3.19 muestra el diseño de esta entrada.

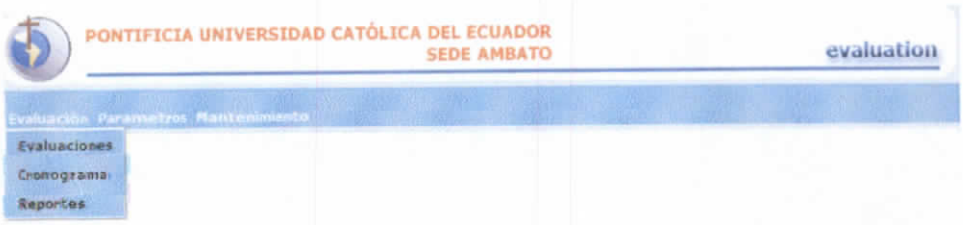


Figura 3.19 Diseño de la GUI de plantilla de la aplicación

La mayoría de entradas del sistema de evaluación del personal administrativo están constituidas por pantallas de mantenimiento, es decir las pantallas que permiten: insertar, actualizar, consultar y eliminar información. La figura 3.20 muestra el diseño básico empleado para las pantallas de mantenimiento.

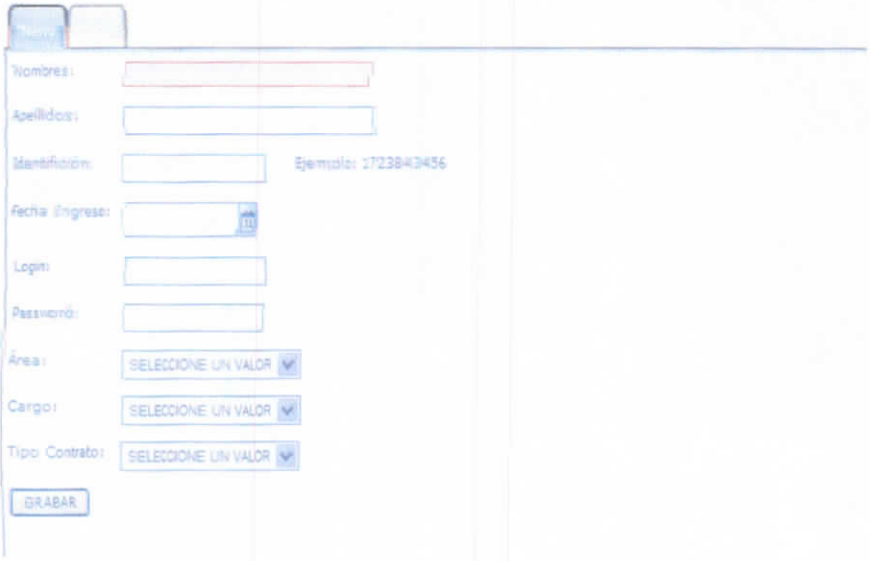


Figura 3.20 Diseño básico de una GUI de mantenimiento

Todas las entradas de mantenimiento se diseñaron con dos tabuladores, uno empleado para insertar nuevos registros, y el segundo para consulta, actualización y eliminación de registros (Ver figura 3.21).

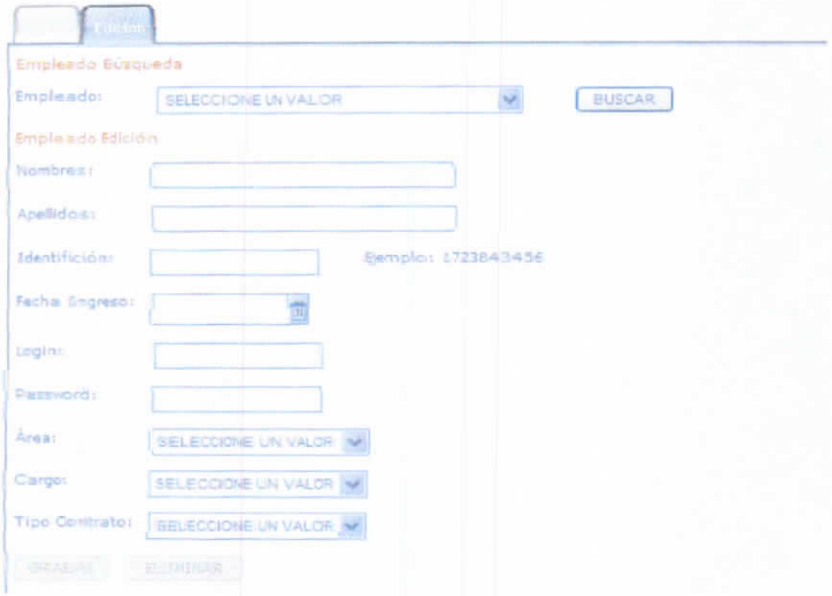


Figura 3.21 Diseño básico de una GUI de mantenimiento

3.3.7 Diseño de Salidas

Las únicas salidas del sistema de evaluación están constituidas por los reportes que fueron definidos en la etapa de análisis. A continuación se detallan estos reportes:

Reporte de evaluación de las observaciones: Incluye gráficos de barra por subfactores con las estadísticas de los resultados.

Reporte de evaluación de las pruebas (prácticas): incluye gráficos de barra por subfactores con las estadísticas de los resultados.

Reporte de los cronogramas agrupados por fecha.

Histórico de todas las evaluaciones realizadas al personal.

En la sección de anexos de este documento se puede encontrar el formato de cada uno de los reportes citados anteriormente.

3.4 Etapa de codificación

3.4.1 Configuración del proyecto Web

El primer paso que es necesario realizar antes de la codificación de la lógica de negocio, es la creación del proyecto Java. En este caso se creó un proyecto Web con la estructura estándar de Java Figura 3.19.

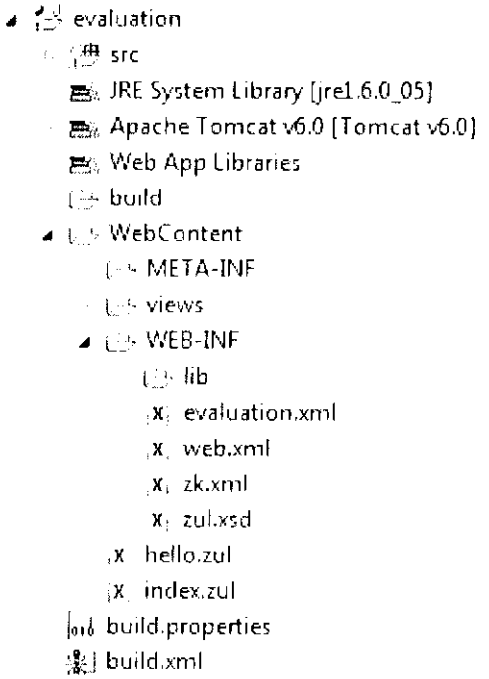


Figura 3.22 Estructura de un proyecto Web de Java

En la carpeta src se ubica todos los paquetes y clases Java que son las fuentes de la aplicación. En la carpeta WebContent se ubica todos recursos

necesarios para la aplicación Web, como por ejemplo: las librerías, los archivos descriptores de la aplicación, imágenes, paginas HTML, hojas de estilo en cascada (CSS, por sus siglas en inglés), las clases compiladas, etc. Para tareas como: compilación de clases, publicación en el servidor de aplicaciones, generación de wars y otras se emplean archivos ant (ant.xml) con asignaciones definidas en un archivo de propiedades (build.propierties). El código del archivo de propiedades es el siguiente:

```
# Ant properties for building the evaluation
appserver.home=D:/Tomcat 6.0
deploy.path=${appserver.home}/webapps
tomcat.manager.url=http://localhost/manager
tomcat.manager.username=admin
tomcat.manager.password=admin
```

Incluye las asignaciones como la ruta del servidor de aplicaciones, la dirección url del administrador de Tomcat y la información de autentificación.

El archivo ant utiliza la información que esta parametrizada en este archivo, para realizar algunas de sus tareas, a continuación se muestra todas las tareas que el archivo ant puede realizar:

```
<?xml version="1.0"?>
<project name="evaluation" basedir="." default="usage">
        <property file="build.properties" />
        <property name="src.dir" value="src" />
        <property name="web.dir" value="WebContent" />
        <property name="build.dir" value="${web.dir}/WEB-INF/classes" />
        <property name="name" value="evaluation" />
        <path id="master-classpath">
            <fileset dir="${web.dir}/WEB-INF/lib">
                <include name="*.jar" />
            </fileset>
```

```
        <!-- We need the servlet API classes:       -->
        <!--   for Tomcat 4.1 use servlet.jar       -->
        <!--   for Tomcat 5.0 use servlet-api.jar   -->
        <!--   for Other app server - check the docs -->
        <fileset dir="${appserver.home}/lib">
            <include name="servlet*.jar" />
        </fileset>
        <pathelement path="${build.dir}" />
    </path>
    <target name="usage">
        <echo message="" />
        <echo message="${name} build file" />
        <echo message="-----------------------------------" />
        <echo message="" />
        <echo message="Available targets are:" />
        <echo message="" />
        <echo message="build     --> Build the application" />
        <echo message="deploy    --> Deploy application as directory" />
        <echo message="deploywar --> Deploy application as a WAR file" />
        <echo message="install   --> Install application in Tomcat" />
        <echo message="reload    --> Reload application in Tomcat" />
        <echo message="start     --> Start Tomcat application" />
        <echo message="stop      --> Stop Tomcat application" />
        <echo message="list      --> List Tomcat applications" />
        <echo message="" />
    </target>
    <target name="build" description="Compile main source tree java files">
        <mkdir dir="${build.dir}" />
        <javac destdir="${build.dir}" target="1.5" debug="true"
deprecation="false" optimize="false" failonerror="true">
            <src path="${src.dir}" />
            <classpath refid="master-classpath" />
        </javac>
    </target>
    <target name="deploy" depends="build" description="Deploy
application">
        <copy todir="${deploy.path}/${name}" preservelastmodified="true">
            <fileset dir="${web.dir}">
                <include name="**/*.*" />
            </fileset>
```

```
            </copy>
        </target>
        <target name="deploywar" depends="build" description="Deploy application as a WAR file">
            <war destfile="${name}.war" webxml="${web.dir}/WEB-INF/web.xml">
                <fileset dir="${web.dir}">
                    <include name="**/*.*" />
                </fileset>
            </war>
            <copy todir="${deploy.path}" preservelastmodified="true">
                <fileset dir=".">
                    <include name="*.war" />
                </fileset>
            </copy>
        </target>
        <!--
================================================================= -->
        <!-- Tomcat tasks - remove these if you don't have Tomcat installed -->
        <!--
================================================================= -->
        <taskdef name="install" classname="org.apache.catalina.ant.InstallTask">
            <classpath>
                <path location="${appserver.home}/lib/catalina-ant.jar" />
            </classpath>
        </taskdef>
        <taskdef name="reload" classname="org.apache.catalina.ant.ReloadTask">
            <classpath>
                <path location="${appserver.home}/lib/catalina-ant.jar" />
            </classpath>
        </taskdef>
        <taskdef name="list" classname="org.apache.catalina.ant.ListTask">
            <classpath>
                <path location="${appserver.home}/lib/catalina-ant.jar" />
            </classpath>
        </taskdef>
        <taskdef name="start" classname="org.apache.catalina.ant.StartTask">
            <classpath>
```

```
                <path location="${appserver.home}/lib/catalina-ant.jar" />
            </classpath>
        </taskdef>
        <taskdef name="stop" classname="org.apache.catalina.ant.StopTask">
            <classpath>
                <path location="${appserver.home}/lib/catalina-ant.jar" />
            </classpath>
        </taskdef>
        <target name="install" description="Install application in Tomcat">
            <install                                    url="${tomcat.manager.url}"
username="${tomcat.manager.username}"
password="${tomcat.manager.password}" path="/${name}" war="${name}" />
        </target>
        <target name="reload" description="Reload application in Tomcat">
            <reload                                     url="${tomcat.manager.url}"
username="${tomcat.manager.username}"
password="${tomcat.manager.password}" path="/${name}" />
        </target>
        <target name="start" description="Start Tomcat application">
            <start                                      url="${tomcat.manager.url}"
username="${tomcat.manager.username}"
password="${tomcat.manager.password}" path="/${name}" />
        </target>
        <target name="stop" description="Stop Tomcat application">
            <stop                                       url="${tomcat.manager.url}"
username="${tomcat.manager.username}"
password="${tomcat.manager.password}" path="/${name}" />
        </target>
        <target name="list" description="List Tomcat applications">
            <list                                       url="${tomcat.manager.url}"
username="${tomcat.manager.username}"
password="${tomcat.manager.password}" />
        </target>
        <!-- End Tomcat tasks -->

        <target name="clean" description="Clean output directories">
            <delete>
                <fileset dir="${build.dir}">
                    <include name="**/*.class" />
                </fileset>
            </delete>
        </target>
```

```
        <target name="undeploy" description="Un-Deploy application">
            <delete>
                <fileset dir="${deploy.path}/${name}">
                    <include name="**/*.*" />
                </fileset>
            </delete>
        </target>
</project>
```

3.4.2 Configuración de ZK

La configuración de ZK se realiza mediante la creación de archivos descriptores específicos en el WebContent de la aplicación. Estos archivos definen el comportamiento y la funcionalidad de ZK en la aplicación.

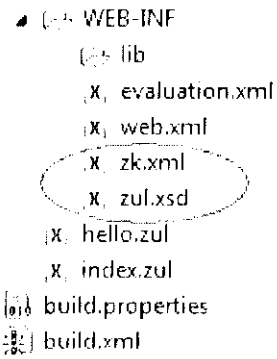


Figura 3.23 Archivos de configuración de ZK

El archivo zul.xsd es un esquema que contienen la información de los controles y propiedades de ZK, por ejemplo: las propiedades y eventos del control *button*. Esta información no es necesaria pero si es muy útil para la creación de interfaces gráficas de usuario.

El archivo zk.xml incluye todas las características de configuración de ZK, a continuación se detalla la configuración utilizada:

```
<?xml version="1.0" encoding="UTF-8"?>
<zk>
<language-mapping>
        <language-name>xml</language-name>
        <extension>svg</extension>
    </language-mapping>
    <language-mapping>
        <language-name>xml</language-name>
        <extension>xml2html</extension>
    </language-mapping>
<session-config>
        <timer-as-inactive>true</timer-as-inactive>
</session-config>
<richlet>
        <richlet-name>Test</richlet-name>
        <richlet-class>org.zkoss.zkdemo.test.TestRichlet</richlet-class>
        <!-- Any number of initial parameters.
        <init-param>
            <param-name>any</param-name>
            <param-value>any</param-value>
        </init-param>
        -->
    </richlet>
    <richlet-mapping>
        <richlet-name>Test</richlet-name>
        <url-pattern>/test/*</url-pattern>
    </richlet-mapping>
<device-config>
        <device-type>ajax</device-type>
        <timeout-uri>/timeout.zul</timeout-uri>
</device-config>
<listener>
        <description>[Optional] Mointor the statistic</description>
        <listener-class>org.zkoss.zk.ui.util.Statistic</listener-class>
</listener>
</zk>
```

3.4.3 Configuración de Spring

La configuración de Spring se realiza en el archivo de contexto de la aplicación, básicamente lo que se escribe son los beans con los diferentes mapeos. La primera configuración de Spring consiste en la creación de un bean que defina el driver que se emplea para la conexión de base de datos Oracle:

```
<bean id="myDataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName">
            <value>oracle.jdbc.driver.OracleDriver</value>
        </property>
        <property name="url">
            <value>jdbc:oracle:thin:@localhost:1521:jupiter</value>
        </property>
        <property name="username">
            <value>evaluation</value>
        </property>
        <property name="password">
            <value>sa</value>
        </property>
    </bean>
```

En este bean se asocia la información del driver de Oracle, la ruta de la base, el usuario y la clave se emplean para la conexión con el motor de base de datos.

Posteriormente, se realiza la configuración del ORM a emplearse, que para el sistema es Hibernate, además se indica que se trabaja vía anotaciones evitando el uso de archivos de mapeo hbm:

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
        <property name="dataSource">
            <ref local="myDataSource" />
        </property>
        <property name="annotatedClasses">
            <list>
                <value>com.pucesa.domain.Departamento</value>
                <value>com.pucesa.domain.Cargo</value>
                <value>com.pucesa.domain.Entidad</value>
                <value>com.pucesa.domain.Empleado</value>
                <value>com.pucesa.domain.Usuario</value>
                <value>com.pucesa.domain.TipoFormato</value>
                <value>com.pucesa.domain.Formato</value>
                <value>com.pucesa.domain.Factor</value>
                <value>com.pucesa.domain.SubFactor</value>

        <value>com.pucesa.domain.NivelEvaluacion</value>
                <value>com.pucesa.domain.Evaluacion</value>

        <value>com.pucesa.domain.DetalleEvaluacion</value>
                <value>com.pucesa.domain.Actividad</value>
            </list>
        </property>
        <property name="annotatedPackages">
            <list>
                <value>com.pucesa.domain</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop
key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop
key="hibernate.transaction.auto_close_session">after_transaction</prop>

            </props>
        </property>
    </bean>
```

En este bean se incluya el listado de todas las clases persistentes, también se indica el paquete en el que estas clases se encuentran ubicadas y el dialecto de Oracle que se emplea.

Luego a través de la inyección de dependencia se le indica a Spring que implementación debe utilizar para las operaciones de persistencia:

```
<bean id="entidadDAOTarget" class="com.pucesa.persistence.EntidadDAOImp">
        <property name="sessionFactory">
            <ref local="sessionFactory"/>
        </property>
    </bean>
```

Además, este bean hace referencia al bean del *sessionFactory* que se encuentra detallado en el código anterior.

Los siguientes beans realizan la configuración de la transaccionalidad de Hibernate a través de Spring, se definen los interceptores con los respectivos puntos de finalización:

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

    <bean id="hibernateInterceptor"
        class="org.springframework.orm.hibernate3.HibernateInterceptor">
        <property name="sessionFactory">
            <ref bean="sessionFactory"/>
        </property>
    </bean>
```

```
<bean id="hibernateTransactionInterceptor"
class="org.springframework.transaction.interceptor.TransactionInterceptor">
<property name="transactionManager"
ref="hibernateTransactionManager"/>
<property name="transactionAttributeSource">
<value>
com.pucesa.persistence.EntidadDAOImp.*=PROPAGATION_REQUIRED
</value>
</property>
</bean>
```

Por último, no es necesario tener código explicito de transaccionalidad en el código de negocios, para conseguir esto se utiliza la característica de programación orientada a aspectos de Spring, esto permite disponer de toda la funcionalidad de transaccionalidad en forma complementaria sin que sea necesario realizar llamadas a métodos transaccionales:

```
<bean id="entidadDAO"
class="org.springframework.aop.framework.ProxyFactoryBean">
<property name="proxyInterfaces">
<value>
com.pucesa.ipersistence.EntidadDAO
</value>
</property>
<property name="target">
<ref local="entidadDAOTarget"/>
</property>
<property name="interceptorNames">
<list>
<value>hibernateInterceptor</value>
<value>hibernateTransactionInterceptor</value>
</list>
</property>
</bean>
```

Para poder emplear la funcionalidad de la interfaz expuesta en Spring es necesario utilizar el *BeanFactory* indicándole el nombre del bean que se desea utilizar, para eso se emplea la clase *BeanFactoryServices:*

```
package com.pucesa.services;
import javax.servlet.ServletContext;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.web.context.support.WebApplicationContextUtils;
import org.zkoss.zk.ui.WebApp;
import com.pucesa.ipersistence.EntidadDAO;

/**
 * @author Alex
 * This singleton class includes all BeanFactory and ServletContext
 * services
 */
public class BeanFactoryServices {

        public static EntidadDAO entidadDAOService(WebApp webApp) {
            ServletContext servletContext = (ServletContext) webApp
                        .getNativeContext();
            BeanFactory beanFactory = WebApplicationContextUtils

        .getRequiredWebApplicationContext(servletContext);
            return (EntidadDAO) beanFactory.getBean("entidadDAO");
        }
}
```

Esta clase devuelve gracias al contexto de la aplicación la implementación que se desee de la interfaz *EntidadDAO*, en este caso Spring ata esa implementación que se haya definida en el bean *entidadDAO.*

La interfaz tiene la firma de todos los métodos persistentes, los mismos que pueden ser utilizados para cualquier tipo de objeto:

```
Departamento departamento = (Departamento) entidadDAO.retrieve(
```

```
Departamento.class, 1));
```

En el método anterior se utiliza el método *retrieve* para recuperar un departamento especifico mediante su id. El método *retrive* es tan general que puede ser utilizado para cualquier otro objeto.

```
Cargo cargo = (Cargo) entidadDAO.retrieve(Cargo.class, 900);
```

Al igual que existe un método *retrive* general, existen métodos generales para cada una de las operaciones de persistencia (insertar, actualizar, eliminar, búsquedas generales):

```
package com.pucesa.ipersistence;
import java.util.List;
/**
 * This interface includes persistence methods signed
 * @author Alex Roldán
 *
 */

public interface EntidadDAO {

        /**
         * Method to save an Object
         * @param obj - Object
         * @return String
         */
        public String save(Object obj);

        /**
         * Method to update an Object
         * @param obj - Object
         */
        public void update(Object obj);

        /**
```

```
     * Methodo to delete an Object
     * @param obj - Object
     */
    public void delete(Object obj);

    /**
     * Methodo to retrieve an Object
     * @param clase - class to retrieve
     * @return Object
     */
    public Object retrieve(Class<?> clase, Integer id);

    /**
     * Method to retrieve all instances of an object
     * @param clase - class to retrieve
     * @return List
     */
    public List<?> retrieveAll(Class<?> clase);
}
```

3.4.4 Clases de negocio

Las clases de negocio están integradas en su mayoría por las clases persistentes, estas clases utilizan anotaciones para indicarle a Hibernate como deben ser persistidas. A continuación solo se escribe el código de la clase de Evaluación. Los campos y métodos del resto de clases ya fueron detallados en el diagrama de clases anteriormente en este documento.

```
package com.pucesa.domain;
import java.sql.Date;
import java.util.HashSet;
import java.util.Set;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
```

```
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;

/**
 * This class represent a Evaluacion Object
 *
 * @author Alex Roldán
 */

@Entity
@javax.persistence.SequenceGenerator(name = "SEQENTIDAD", sequenceName
= "SEQENTIDAD")
@Table(name = "Evaluacion")

public class Evaluacion implements java.io.Serializable {

        private static final long serialVersionUID = -7407652518293458286L;

        private Integer id;
        private Date fecha;
        private String puestoCodigo;
        private String observaciones;
        private String plan;
        private Empleado empleado;
        private Usuario usuario;
        private Formato formato;
        private Set<DetalleEvaluacion> detalleEvaluaciones = new
HashSet<DetalleEvaluacion>(0);

        public Evaluacion() {
            super();
        }

        public Evaluacion(Date fecha, String puestoCodigo, String
observaciones,
                String plan, Empleado empleado, Usuario usuario, Formato
formato) {
            super();
```

```
        this.fecha = fecha;
        this.puestoCodigo = puestoCodigo;
        this.observaciones = observaciones;
        this.plan = plan;
        this.empleado = empleado;
        this.usuario = usuario;
        this.formato = formato;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"SEQENTIDAD")
    @Column(name = "id")
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    @Column(name = "fecha")
    public Date getFecha() {
        return fecha;
    }

    public void setFecha(Date fecha) {
        this.fecha = fecha;
    }

    @Column(name = "puestoCodigo")
    public String getPuestoCodigo() {
        return puestoCodigo;
    }

    public void setPuestoCodigo(String puestoCodigo) {
        this.puestoCodigo = puestoCodigo;
    }
```

```
@Column(name = "observaciones")
public String getObservaciones() {
    return observaciones;
}

public void setObservaciones(String observaciones) {
    this.observaciones = observaciones;
}

@Column(name = "plan")
public String getPlan() {
    return plan;
}

public void setPlan(String plan) {
    this.plan = plan;
}

@ManyToOne
@JoinColumn(name = "empleadoId")
public Empleado getEmpleado() {
    return empleado;
}

public void setEmpleado(Empleado empleado) {
    this.empleado = empleado;
}

@ManyToOne
@JoinColumn(name = "usuarioId")
public Usuario getUsuario() {
    return usuario;
}

public void setUsuario(Usuario usuario) {
    this.usuario = usuario;
}

@ManyToOne
```

```
    @JoinColumn(name = "formatoId")
    public Formato getFormato() {
        return formato;
    }

    public void setFormato(Formato formato) {
        this.formato = formato;
    }

    @OneToMany(mappedBy = "evaluacion")
    @JoinColumn(name = "id")
    public Set<DetalleEvaluacion> getDetalleEvaluaciones() {
        return detalleEvaluaciones;
    }

    public void setDetalleEvaluaciones(Set<DetalleEvaluacion>
detalleEvaluaciones) {
        this.detalleEvaluaciones = detalleEvaluaciones;
    }

}
```

3.4.5 Etapa de mantenimiento

El éxito del mantenimiento de un sistema se fundamenta el en hecho de que tan fácil es implementar nuevas funcionalidades o requerimientos. Dentro de este contexto, se puede corroborar que el sistema de evaluación del personal administrativo de la PUCESA, aunque haya sido planificado solo para la evaluación del personal del Área de Secretaría de la Universidad, está diseñado para cambiar y soportar el proceso de evaluación del resto de áreas de la universidad.

Además, la elaboración de los formatos de evaluación son totalmente fáciles de administrar en el sistema, permitiendo que estos formatos puedan ser: modificados, eliminados e incluso crear nuevos formatos para tareas de evaluación.

Adicionalmente, con toda la arquitectura que se empleo en la elaboración del sistema de evaluación del personal administrativo de la PUCESA, se consiguieron algunas características que facilitan el mantenimiento de la aplicación considerablemente, entre las características se pueden mencionar:

El diseño del sistema basado en tres capas facilita realizar procesos de mantenimientos especializados en cada capa, sin que el resto de capas se vean afectadas. Por ejemplo: se podría crear una nueva capa de presentación que este enfocada a dispositivos móviles (este un en ejemplo y por lo tal no se refleja en el ámbito del proyecto), y al tener esa independencia entre capas, no es necesario alterar ni la capa de datos ni la capa de negocios para que el sistema trabaje en dispositivos móviles.

El uso de Spring, que a su vez implica directamente el empleo de una serie de patrones de diseño, posibilita el realizar cambios en la aplicación que normalmente sería de gran impacto sino se utilizara Spring. Por ejemplo: el cambio del ORM empleado, del motor de base de datos, del modelo de presentación. Incluso la misma lógica de negocio puede verse influenciada positivamente al cambio con Spring.

Con Spring, también se consigue delegar tareas que no son de interés para el desarrollador, permitiendo que el desarrollador se enfoque en la lógica del negocio que es realmente su interés. Spring es un framework que incluye una gran cantidad de funcionalidad prefabricada (transacciones, seguridad, mensajería, etc.) que no necesita ser programada, ni mucho menos modificada lo que finalmente facilita el mantenimiento.

CAPITULO IV

DESARROLLO DE LA APLICACIÓN

4.1 Verificación de la hipótesis

Partiendo de la hipótesis planteada en el primer capítulo: "El estudio del framework de desarrollo Spring y su empleo práctico, permite la construcción de una aplicación Web para evaluación del personal de la PUCESA que sea escalable, reutilizable y que no exija complicados procesos de desarrollo".

Se demuestra la hipótesis mediante el método lógico Modus Ponendo Ponens.

A= Variable Independiente

B= Variable Dependiente

A= El estudio del framework de desarrollo Spring y su empleo práctico.

B= La construcción de una aplicación Web para evaluación del personal de la PUCESA que sea escalable, reutilizable y que no exija complicados procesos de desarrollo.

$$A \rightarrow B$$

$$A$$

$$B$$

Esto quiere decir:

Que el estudio del framework de desarrollo Spring y su empleo práctico permitió la construcción de una aplicación Web para evaluación del personal de la PUCESA que sea escalable, reutilizable y que no involucró complicados procesos de desarrollo.

4.2 Validación de los resultados

4.3 Conclusiones

Con la elaboración del sistema de evaluación del personal administrativo del Área de Secretaría de la PUCESA, se aplicaron y reforzaron los conceptos estudiados sobre el desarrollo de aplicaciones con Spring, que permite construir aplicaciones escalables, fáciles de mantener y de reutilizar.

El empleo de Spring requiere un alto nivel de aprendizaje y fundamentos, por lo que este documento puede ser una buena base inicial para los profesores, alumnos de la universidad, y desarrolladores en general que quieran incursionar en el desarrollo de soluciones empresariales con Spring.

El sistema desarrollado permite llevar un control de las evaluaciones realizadas al personal administrativo de la Universidad por el Área de Recursos Humanos. A pesar de que el ámbito del sistema era la evaluación del personal del Área de Secretaría, el sistema se puede parametrizar para permitir la evaluación del resto de áreas. Además, la aplicación reúne las características propias que se consiguen al desarrollar con Spring: escalabilidad, facilidad de mantener, reutilizar y testear.

Spring es framework liviano que se distribuye en un archivo jar de aproximadamente 2,5MB, este archivo jar es fácilmente instalable en cualquier proyecto y al ser de poco tamaño evita la sobrecarga de la aplicación mejorando la escalabilidad. Adicionalmente, Spring soporta la programación orientada a aspectos, la cual permite al desarrollador enfocarse en la programación de la lógica del negocio, delegando otras

responsabilidades que no tienen que ver con la lógica de negocio (como la transaccionalidad) a este framework.

4.4 Recomendaciones

Para facilitar el entendimiento de los temas relacionados con el estudio de Spring, se recomienda una lectura progresiva de este documento. Ya que algunos tópicos de este trabajo hacen referencia a temas explicados con anterioridad en este mismo documento.

Se sugiere que las llamadas a métodos o servicios expuestos por las clases sean realizadas a través de interfaces, esto permite ocultar la implementación, la cual puede ser inyectada utilizando Spring. Con esta sugerencia las aplicaciones pueden son más fáciles de mantener, escalar y testear.

La complejidad de Spring radica en que se debe tener una sólida base de fundamentos de diseño orientado a objetos, por lo que se recomienda, antes de empezar a utilizar Spring, estudiar ampliamente los tópicos de este tema y específicamente los patrones de diseño.

Para una adecuada utilización del sistema de evaluación del personal administrativo de la PUCESA, es necesario realizar una parametrización de todas las condiciones de evaluación, como por ejemplo: formatos, subformatos, niveles de evaluación, áreas, empleados y usuarios.

Normalmente las versiones de Spring son compatibles hacia atrás, por lo que reemplazar en un proyecto una versión de Spring por otra superior no

debería representar mayor problema. Sin embargo, en ciertas funcionalidades las versiones difieren por lo que se recomienda leer las notas de reléase de cada nueva versión.

En la mayoría de escenarios es recomendable utilizar la programación orientada a aspectos. Sin embargo, algunas veces es necesario, de acuerdo a la complejidad del negocio, no delegar las responsabilidades de transaccionabilidad a la programación orientada a aspectos de Spring. Sobre todo en aquellos casos donde se requiere tener un control de la transaccionalidad de la aplicación.

ABSTRACT DE LA DISERTACION DE GRADO, REVISADO POR LA ESCUELA DE LENGUAS Y LINGUISTICA DE LA PUCESA.

NOMBRES Y APELLIDOS DEL ESTUDIANTE: JAIRO ALEXANDER ROLDAN AGUDELO

UNIDAD ACADEMICA: ESCUELA DE INGENIERIA EN SISTEMAS

TEMA: "ESTUDIO DEL FRAMEWORK DE DESARROLLO SPRING CON UN ENFOQUE PRACTICO BASADO EN EL DESARROLLO EN JAVA DE UNA APLICACIÓN WEB PARA EVALUACIÓN DEL PERSONAL DE LA PUCESA"

RESUMEN

Actualmente los programadores de software deben desarrollar complejas aplicaciones empresariales que faciliten la rentabilización y puedan crecer conforme a las necesidades del negocio. Razón por la cual es muy importante relacionar frameworks como Spring que permitan a los desarrolladores construir este tipo de aplicaciones.

Spring es un framework de código abierto, creado para solventar el desarrollo de aplicaciones empresariales. Además, Spring es un framework completo que maneja todos los aspectos concernientes al desarrollo de este tipo de aplicaciones. Con Spring se desarrolló un sistema para evaluación del personal de la PUCESA que reúne muchas de las características del desarrollo de aplicaciones hoy en día.

Este sistema provee un sólido modelo de programación y configuración de código y buenas prácticas de programación como la programación de interfaces en lugar de la programación de clases. Este framework también facilita el diseño orientado a objetos, el intercambio y la disminución en la dependencia de código entre objetos. El sistema Evaluador del Personal de la PUCESA, desarrollado con Spring, posee un Mapeo Objeto-Relacional (ORM) con sus siglas en Inglés que permite trabajar con cualquier Sistema de Base de Datos sin necesidad de aprender lenguaje SQL. El sistema Evaluador de Personal Administrativo de la PUCESA también tiene una estructura de vistas hechas en JSP siguiendo el patrón de diseño Modelo Vista Controlador (MVC) que es el más

En conclusión: El sistema Evaluador del Personal Administrativo de la PUCESA es una aplicación empresarial desarrollada con Spring, este sistema es escalable y su código puede ser reutilizado y configurado fácilmente.

ABSTRACT

Nowadays, software programmers need to develop complex enterprise applications that grow with business necessities. Consequently, it is extremely essential to look for frameworks such as Spring that support developers to build these software programs.

Spring is an open source framework, created to address the complexity of enterprise application development. Furthermore, Spring is an extended framework that holds all aspects about enterprise application development. With Spring, a Staff Evaluator Application System for the PUCESA, that comprises several enterprise development features, can be developed.

This application provides a consistent programming model, code reuse and good programming practice, such as programming to interfaces rather than classes. This framework also aims to facilitate Object Oriented design, pluggability and loosely coupled application code. The Staff Evaluator Application System for the PUCESA, developed with Spring, provides an Object-relational mapping (ORM) that allows working with any Data Base System without using a SQL language. The Staff Evaluator Application System for the PUCESA also has a multiple-layers structure (based in Spring) that allows software configuration management.

In conclusion, the Staff Evaluator Application System for the PUCESA system is an enterprise application developed with Spring framework. This application is scalable and its code could be reused and configured easily.

Firma Estudiante
Nro. Matrícula 949

CERTIFICACION

Una vez revisado el "Abstract" de la Disertación de Grado, con el tema:

"ESTUDIO DEL FRAMEWORK DE DESARROLLO SPRING CON UN ENFOQUE PRACTICO BASADO EN EL DESARROLLO EN JAVA DE UNA APLICACIÓN WEB PARA EVALUACIÓN DEL PERSONAL DE LA PUCESA" presentado por el señor Jairo Alexander Roldán Agudelo, previo a la obtención del título de Ingeniero en Sistemas y Computación, certifico que el mismo guarda concordancia con el contenido del texto en español.

Lic. Mariela Pérez
DIRECTORA ELL

PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR SEDE AMBATO
DIRECCION
ESCUELA DE LENGUAS Y LINGÜISTICA

CERTIFICACIÓN

La Dirección Administrativa y RRHH de la Pontificia Universidad Católica del Ecuador Sede Ambato, certifica que:

La tesis de grado: "Estudio del framework de desarrollo spring con un enfoque práctico basado en el desarrollo en Java de una aplicación Web para evaluación del personal de la PUCESA", elaborada por el Sr. Jairo Alexander Roldán Agudelo, con CC 0400830634, egresado de la Escuela de Ingeniería en Sistemas, está en funcionamiento y cumple con el objetivo de automatizar con la aplicación Web el proceso de evaluación del desempeño del personal administrativo de la PUCESA.

Es todo cuanto puedo certificar en honor a la verdad, autorizando al interesado hacer uso de este documento, el uso que estimare conveniente.

Ambato abril 3, 2009

Ing. Rosa Palacios Gavilanes
Directora Administrativa y RRHH

DIRECCION ADMINISTRATIVA Y RR.HH

Ing. Laura Rivera
Asistente de la Dirección Administrativa y RRHH

PONTIFICIA UNIVERSIDAD CATOLICA DEL ECUADOR
SEDE AMBATO
DIRECCIÓN ADMINISTRATIVA Y RRHH

Av. Manuelita Sáenz s/n
Sector El Tropezón
Apartado Postal No.18-01662
Telf: 593 3 2 414 604 ext. 109-110
Fax: 593 3 2 411 868 ext. 105
rosy@pucesa.edu.ec
lrivera@pucesa.edu.ec
Ambato - Ecuador
www.pucesa.edu.ec

BIBLIOGRAFÍA

[6] ECKEL, Bruce: *Thinking in Java*. 3ra edición. New Jersey: Prentice Hall, 2003. 1151 p. ISBN 0-13-027363-5.

[3] HEMRAJANI, Anil: *Java Development with Spring, Hibernate and Eclipse*. 1ra edición. Estados Unidos de América: John Wiley & Sons, 2006. 360 p. ISBN-10 0-672-32896-8.

[2] JOHNSON, Rod; HOELLER, Juergen; ARENDSEN, Alef: *Professional Java Development with the Spring Framework*. 1ra edición. Indianápolis, IL: Wiley Publishing, 2005. 672 p. ISBN 0764574833.

[4] NEWARD, Ted: *Effective Enterprise Java*. 1ra edición. Boston, MA: Pearson Education, 2006. 496 p. ISBN 0-321-13000-6.

[5] ROMAN, Ed: *Mastering Enterprise JavaBeans*. 2da edición. Canadá: John Wiley & Sons, 2002. 621 p. ISBN 0-471-41711-4.

[1] WALLS, Craig: Spring in Action. 2da edición. Estados Unidos de América: Manning Publications Co, 2008. 765 p. ISBN 1-933988-13-4.

GLOSARIO

AJAX: Asynchronous JavaScript and XML, nombre original en inglés.

AOP: Programación orientada a aspectos, aspect-oriented programming.

Apache Tomcat: Servidor de aplicaciones de Internet para Java.

CodeBehind: Mecanismo que permite atar una Clase Java a una interfaz gráfica de usuario.

CSS: Hoja de estilos en cascada, Cascading Style Sheets.

DAO: Objeto de acceso a datos, Data Access Object.

DI: Inyección de dependencia, Depency Injection.

DML: Lenguaje de manipulación de datos, Data Manipulation Languaje.

Eclipse: IDE de desarrollo que se puede emplear para proyectos Java.

EJB: Enterprise JavaBeans, nombre original en inglés.

Factory: Clases administradoras de servicios.

HQL: Lenguaje para consultas de Hibernate, Hibernate Query Language.

IOC: Patrón de diseño inversion de control, Inversion Control.

JCA: J2EE Connector Architecture, nombre original en inglés.

JDBC: Java Database Controller, nombre original en inglés.

JMS: Java Message Service, nombre original en inglés.

JMX: Java Management Extensions, nombre original en inglés.

JNDI: Java Naming and Directory Interface, nombre original en inglés.

JSF: Java Server Faces, nombre original en inglés.

JTA: Java Transacction API, nombre original en inglés.

MVC: Patrón de diseño modelador vista/controlador, Model View Controller.

Opernsouce: Código abierto de cualquier arquitectura.

Oracle: Motor de Base de datos.

ORM: Mapeo Objeto/Entidad, Object-relational mapping.

POJO: Objetos simples y viejos de Java, Plain Old Java Object.

RIA: Aplicaciones ricas de Internet, Rich Internet Application.

RPC: Llamadas a procedimientos remotos, Remote Procedure Call.

Singleton: Patrón de diseño que crea una sola instancia que es compartida.

SQL: Lenguaje estructurado de consulta, Structured Query Language.

Template Method: Metodo de plantilla, patrón de diseño.

UML: Lenguaje unificado de modelamiento, Unified Model Languaje.

XML: Lenguaje de marcado extendido, Extensible Markup Language.

ANEXOS

ANEXO 1

CUESTIONARIO DE LEVANTAMIENTO DE REQUERIMIENTOS

Cuestionario para levantamiento de requierimento del sistema de evaluación de la PUCESA

Dirigido: Ing. Rosita Palacios.

Autor: Jairo Roldán

Introducción

Describa brevemente las necesidades de un sistema de evaluación.

Preguntas

1. ¿Qué áreas de la Universidad utilizarán el sistema?

2. ¿Cuáles son los propósitos principales del sistema?

3. ¿Qué beneficios se espera conseguir con la elaboración del sistema?

4. ¿Cuáles son los problemas actuales que se espera solucionar con el sistema?

5. ¿Quién o quienes son las personas encargadas de realizar las encuestas y de añadirlas al sistema?

6. ¿Quién o quienes son las personas encargadas de registrar o llenar las encuestas?

7. ¿Quién o quienes son las personas que serán evaluadas por el sistema?

8. Se podría definir todos los usuarios del sistema y que rol tendrán en el mismo.

9. ¿Existe una clasificación para las personas que serán evaluadas, por ejemplo: profesores, personal administrativo, etc.?

10. ¿Existe una clasificación para las personas que llenarán las encuestas, por ejemplo: alumnos, becarios, oyentes, etc.?

11. ¿Existe una clasificación especial para las encuestas, por ejemplo: encuestas por departamento, encuestas por personal, encuestar por alumnos y semestres, etc.?

12. A más de las encuestas que otra información es requerida de registrarse en el sistema.

13. ¿Qué información es requerida de consulta en el sistema?

14. A más de la cuantificación de las encuestas para evaluación del personal ¿Debe existir otras operaciones o cálculos que deba realizar el sistema?

15. Se podría especificar los reportes, estadísticas, o informes que se desean obtener del sistema.

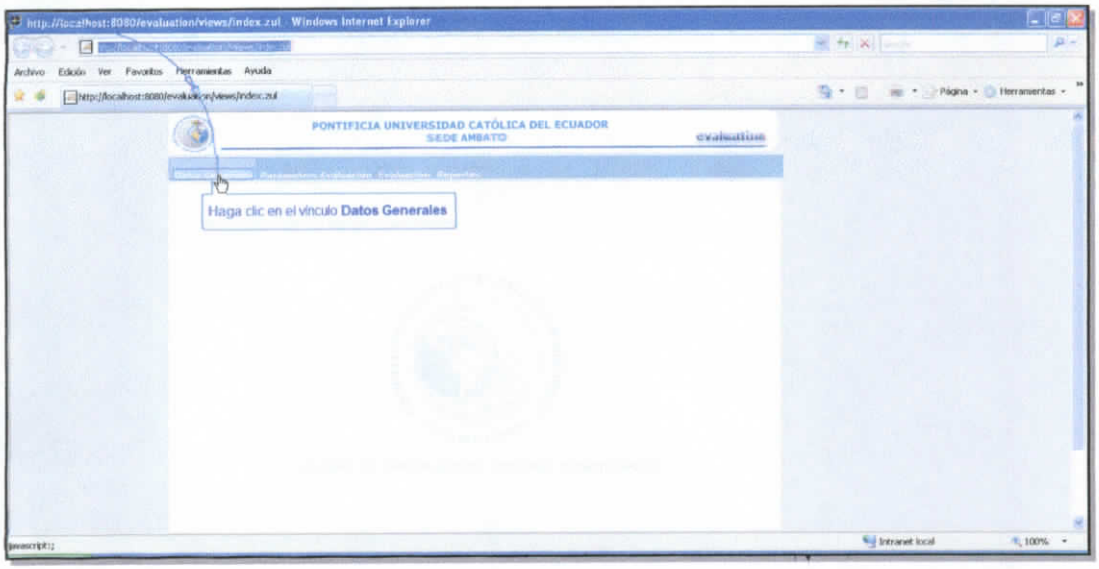
16. Licencias con las que contamos para base de datos, sistema operativo, servidor y otros recursos.

ANEXO 2

GUÍA DE USUARIO DEL SISTEMA DE EVALUACIÓN DEL PERSONAL ADMINISTRATIVO DE LA PUCESA

1. *Sección de Datos Generales*

La sección de Datos Generales de la aplicación permite realizar la parametrización inicial de la información que es necesaria para el resto de procesos del sistema de evaluación.



Existen diferentes hipervínculos (llamados también celdas dentro de este documento), que se encuentran dentro de un menú, que pueden ser seleccionados, todas las páginas de esta sección tienen una misma estructura de inserción, actualización y eliminación.

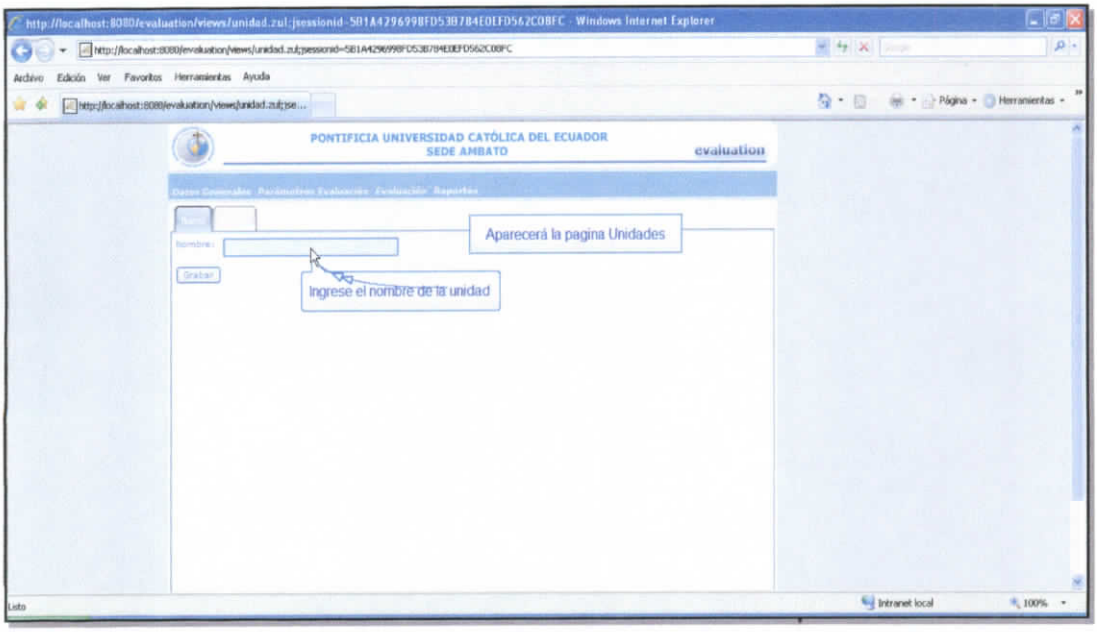
1.1 Páginas de unidades

Esta página permite administrar las unidades de la universidad, las cuales serán utilizadas en procesos posteriores del sistema, como por ejemplo: Definición de instrumentos de evaluación y en la definición de las áreas de la universidad.



Esta página está constituida por dos tabuladores: el tabulador **nuevo** que permite insertar una nueva unidad; y el tabulador **edición** que permite actualizar, eliminar y consultar una unidad específica. Esta es la misma estructura se emplea en todas las páginas de mantenimiento y administración del sistema.

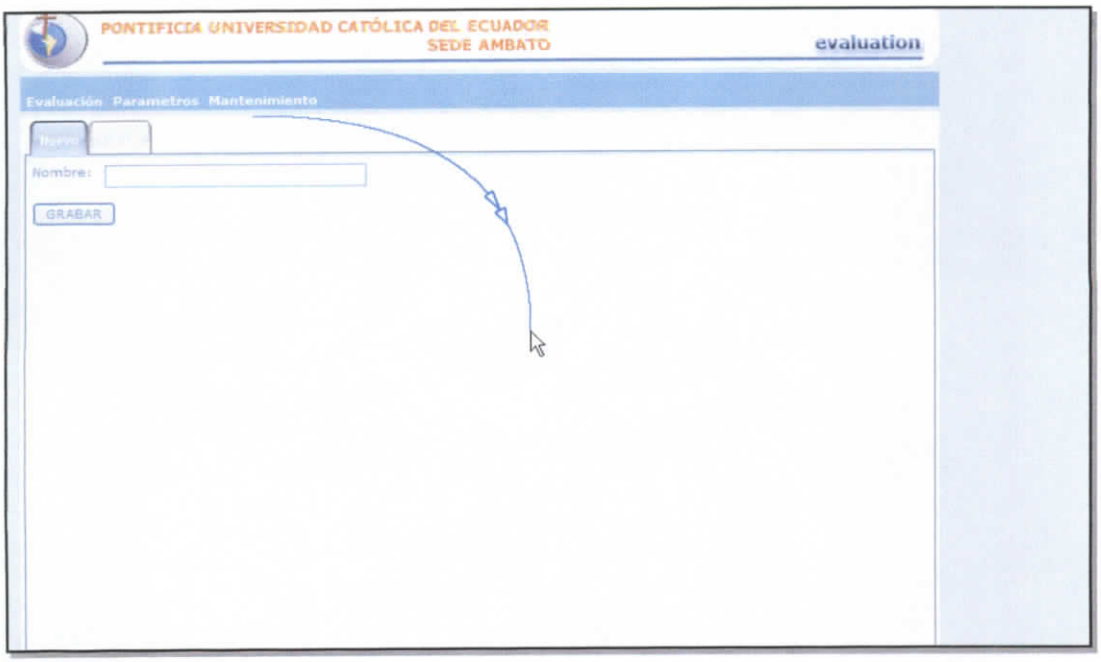
Para grabar una nueva unidad, se ingresa el nombre de la nueva unidad en el cuadro de texto y se procede a dar clic en el botón de grabar:



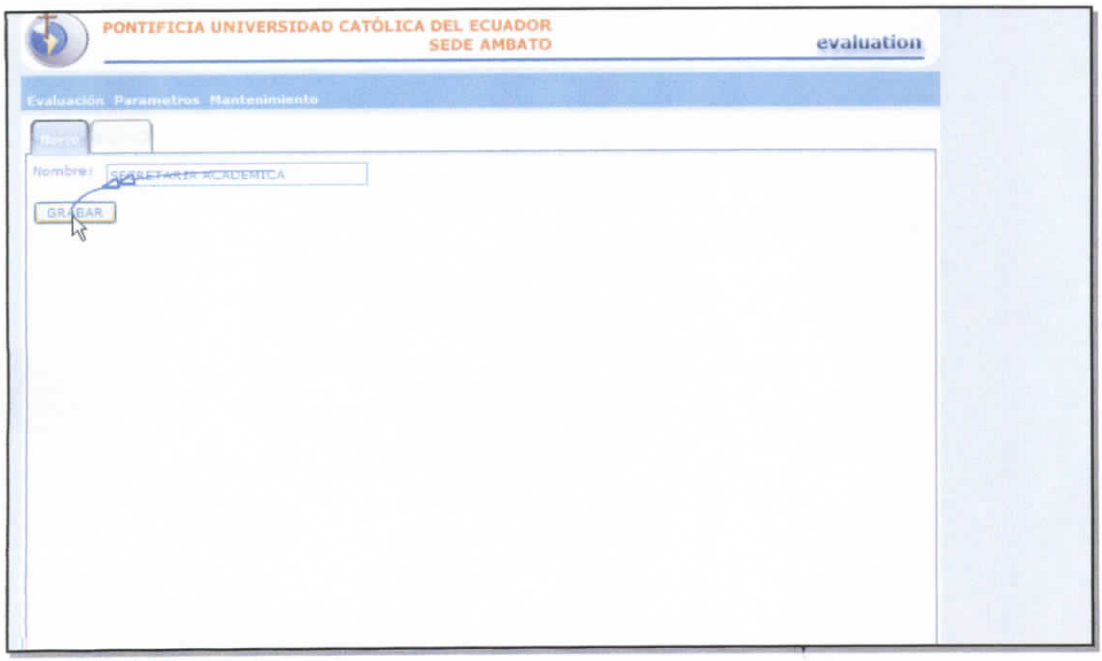
1.2 Página de áreas

Esta página permite ingresar nuevas áreas de la universidad, las cuales serán utilizadas en procesos posteriores del sistema, como por ejemplo: Definición de instrumentos de evaluación, ingreso de empleado, entre otros.

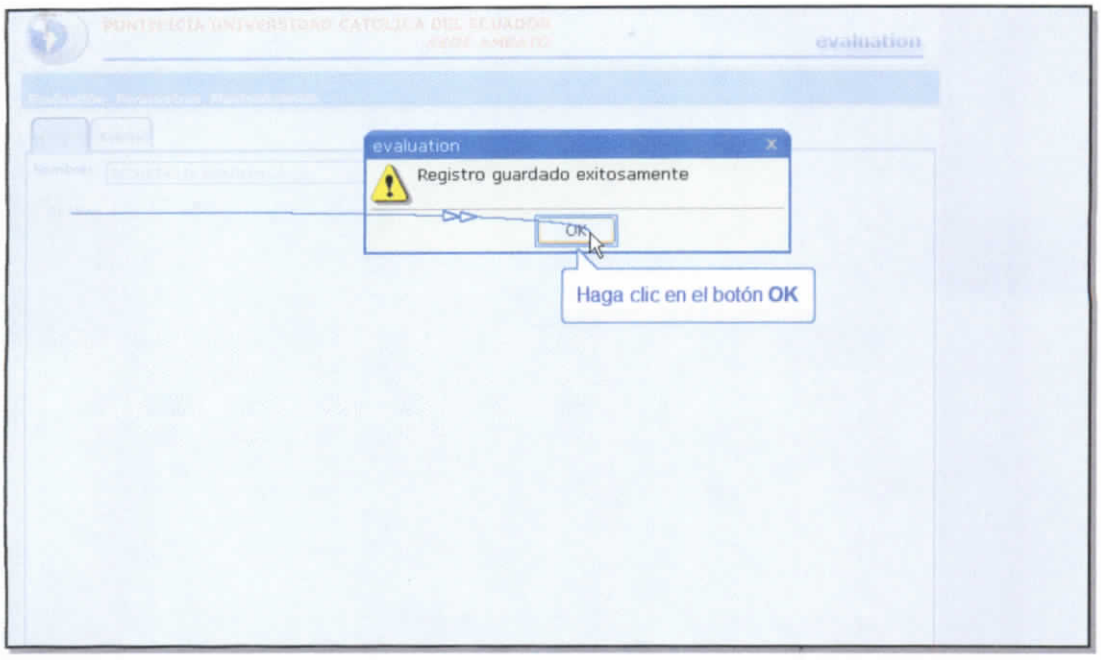
Posteriormente a la selección de la celda de área, se visualiza la página de gestión de área, está constituida por dos tabuladores: el tabulador **nuevo** que permite insertar una nueva área; y el tabulador **edición** que permite actualizar, eliminar y consultar un área específica. Esta misma estructura se emplea en todas las páginas de mantenimiento y administración del sistema.



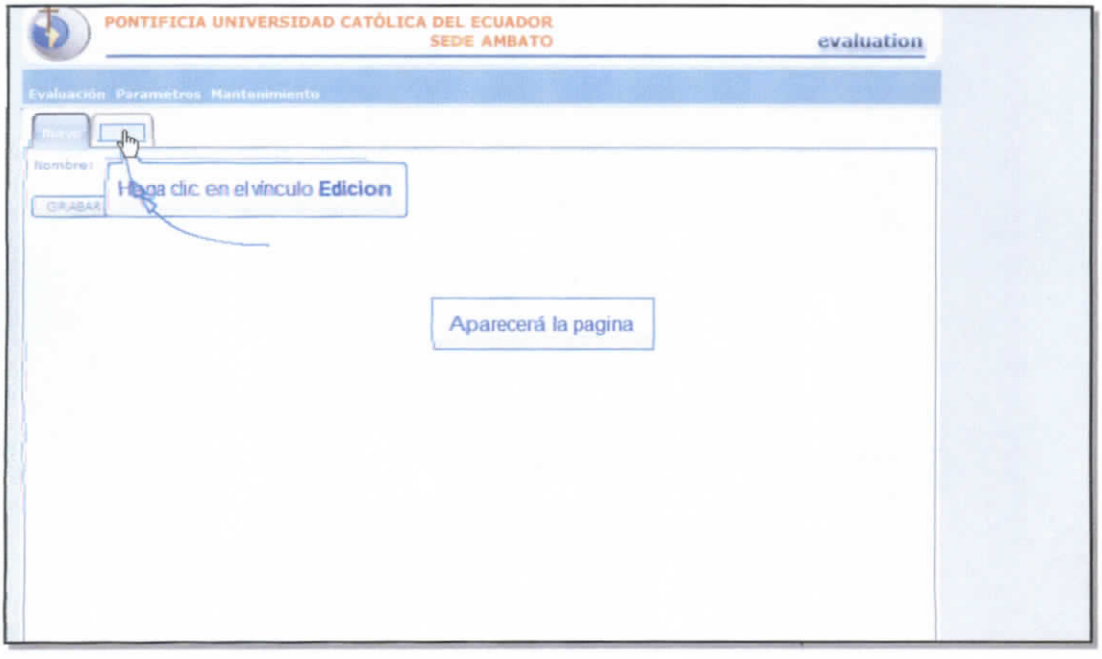
Para grabar una nueva área, se ingresa el nombre de la nueva área en el cuadro de texto y se procede a dar clic en el botón de grabar:



En el caso que el proceso de grabado del área se haya realizado con éxito, se despliega un mensaje de confirmación exitosa.

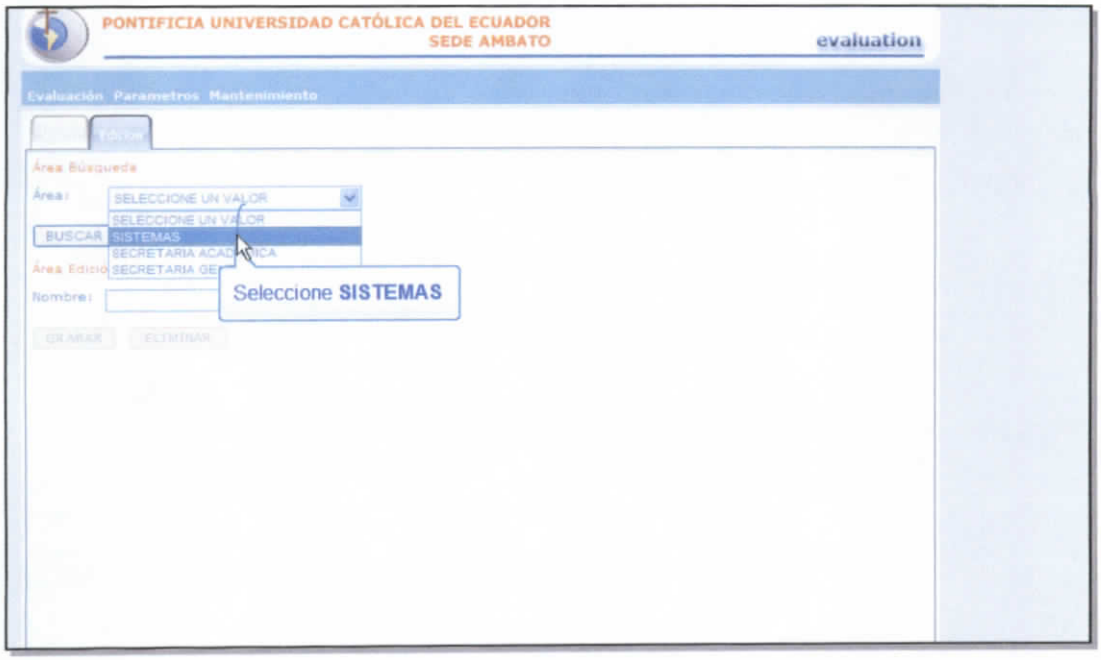


En el caso que sea necesario actualizar (eliminar o consultar) un área que se haya ingresado previamente, es necesario volver a ingresar a menú de navegación y dar clic en la celda de área.

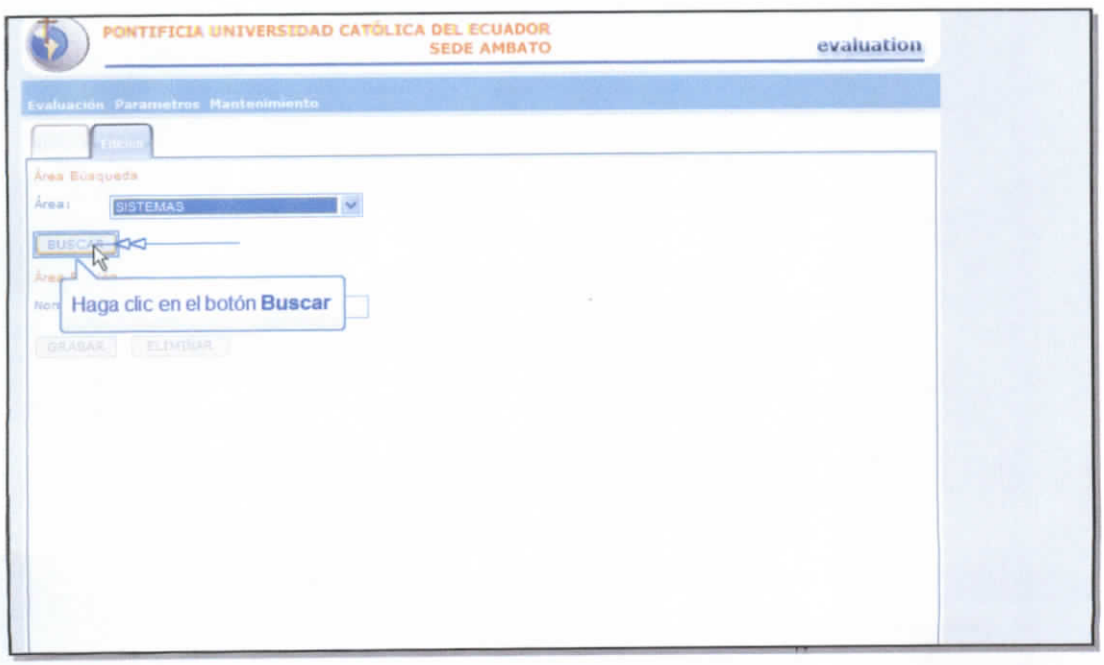


Posteriormente a dar clic en el tabulador de edición aparecerá el área de edición, el sistema permite intercambiar las áreas de inserción y edición en cualquier momento, solo es necesario dar clic en el tabulador del área que se desee

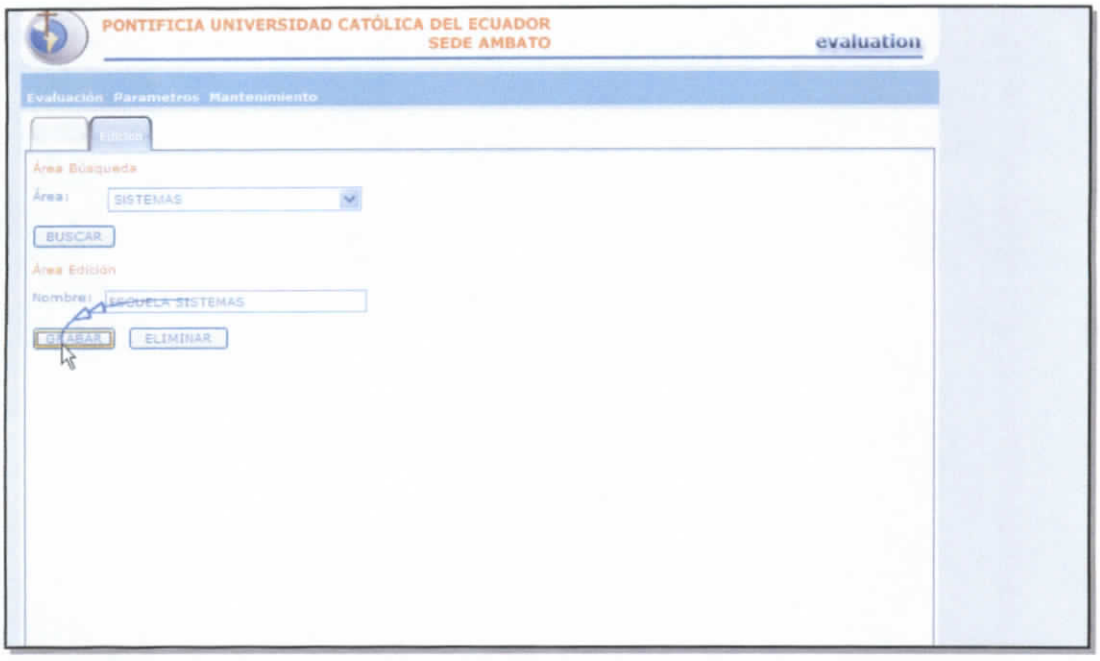
desplegar. Toda área de edición en cualquier página incluye un combo de consulta que permite seleccionar el registro que se desea actualizar o eliminar.



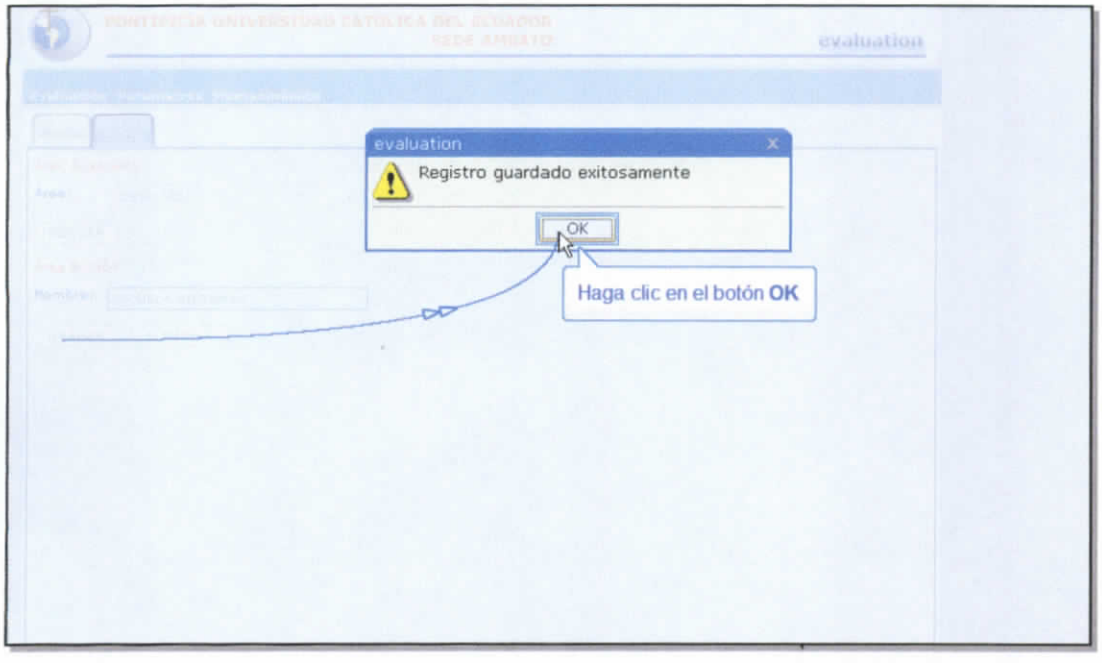
Posteriormente a la selección del registro es necesario dar clic en el botón de búsqueda para acceder a la posibilidad de modificar sus campos.



Se realizan la modificación del campo que se quiere actualizar y posteriormente se procede a grabar (o clic en eliminar si es la operación deseada).

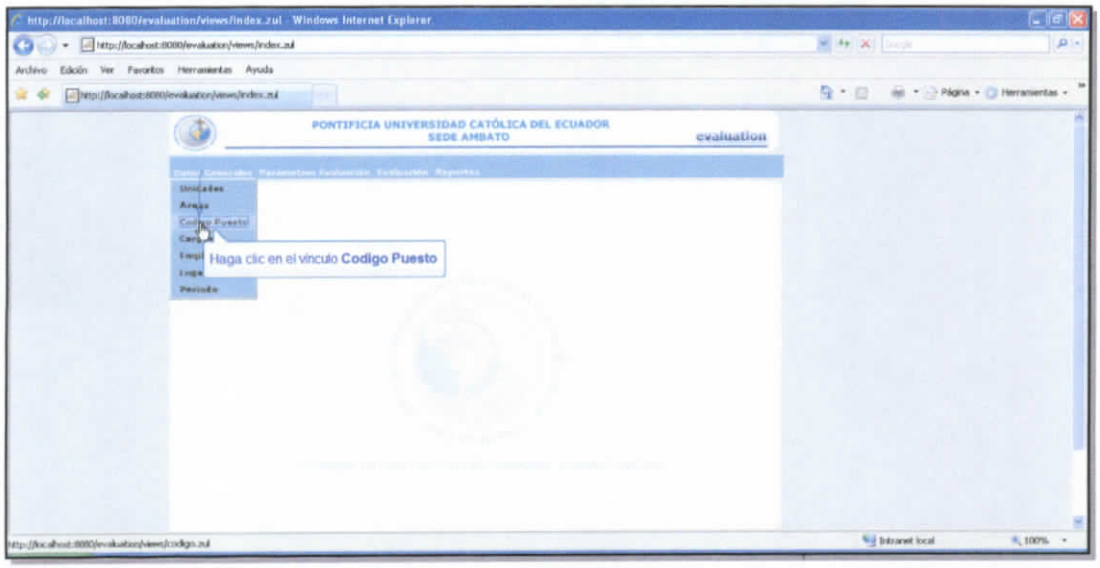


Se despliega el mensaje de confirmación ya explicado.



1.3 Página de código de puesto

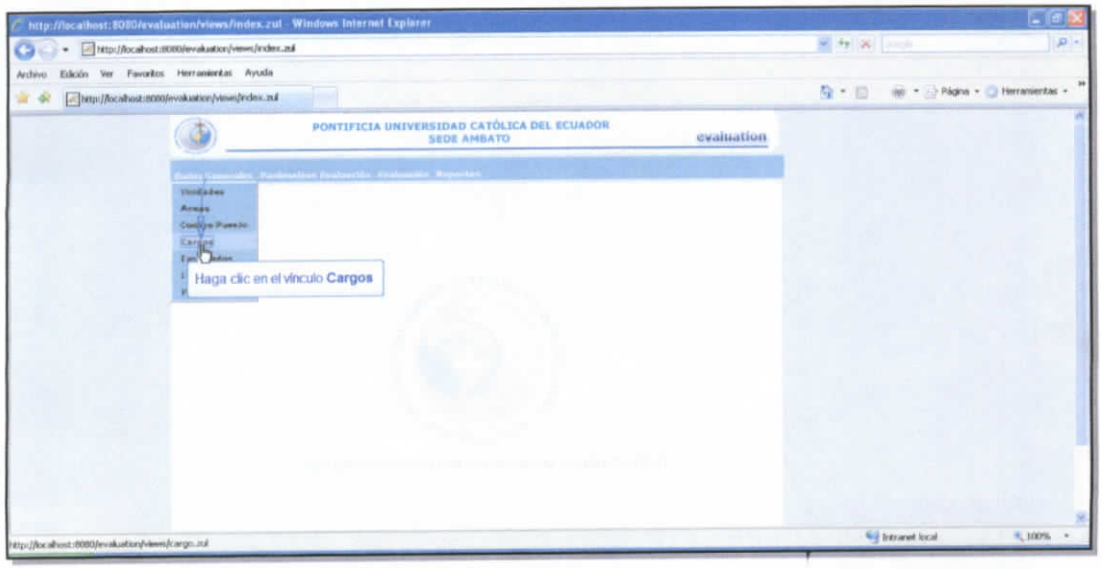
Esta página se utiliza para la parametrización de los códigos que se emplearán al momento de ingresar los cargos. Para acceder a esta página se debe ingresar al menú de mantenimiento y seleccionar la celda de código de puesto.



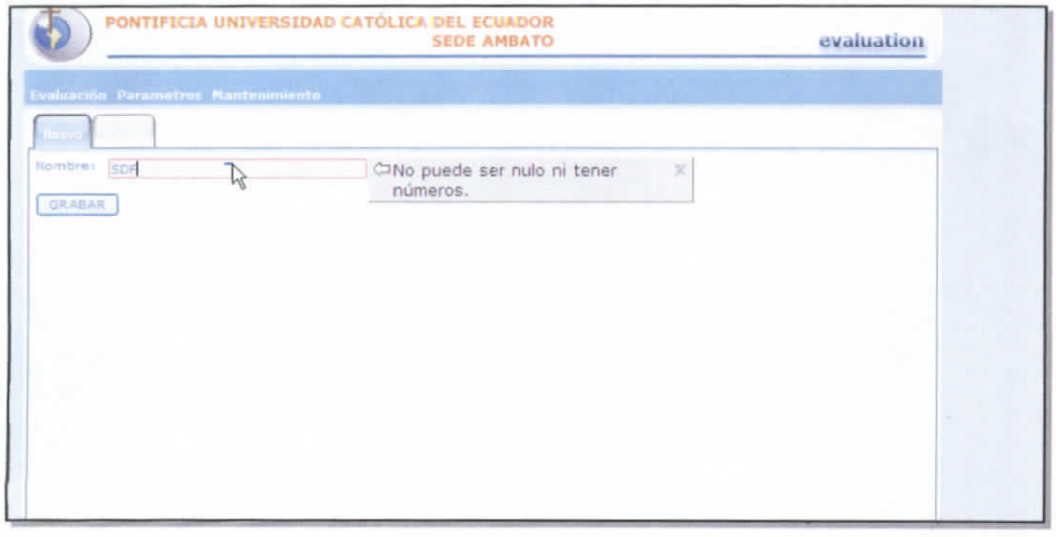
La página de códigos de puesto tiene el mismo funcionamiento que la página de áreas que se menciono anteriormente, incluso este funcionamiento se extiende a todas las páginas de parametrización del sistema de evaluación.

1.4 Página de cargo

Esta página se utiliza para la parametrización de los cargos que se emplearán al momento de ingresar los empleados. Para acceder a esta página se debe ingresar al menú de mantenimiento y seleccionar la celda de cargos.

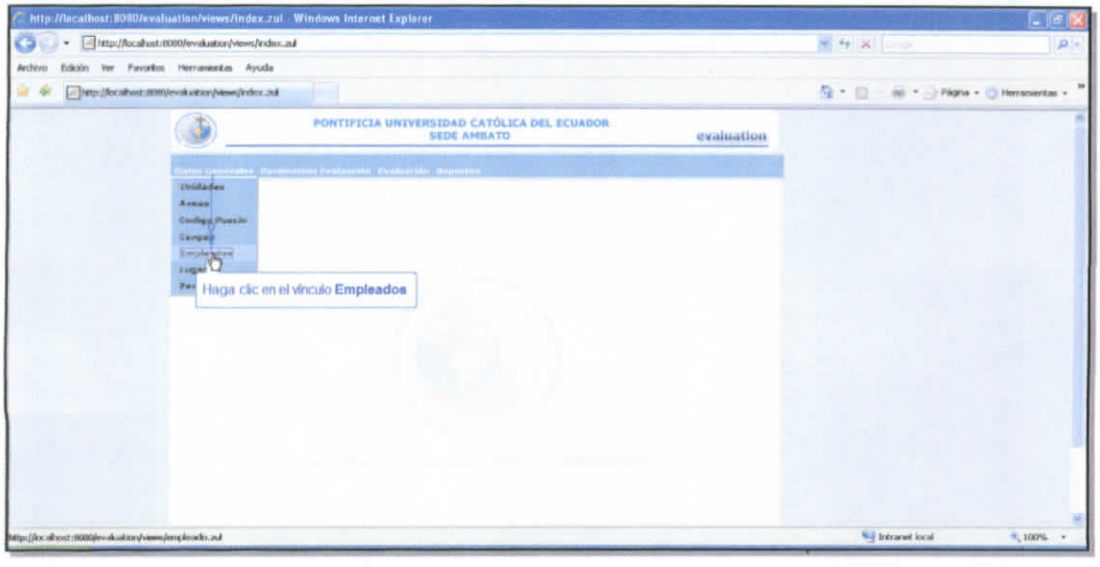


Algunas páginas tienen cuadros de texto con validación, por ejemplo: en la página de cargos el nombre no puede contener números o ser nulo. En caso que la validación no se cumpla se despliegan mensajes de error indicando el problema. Hasta que la validación no haya sido superada ningún otro proceso puede ser realizado.

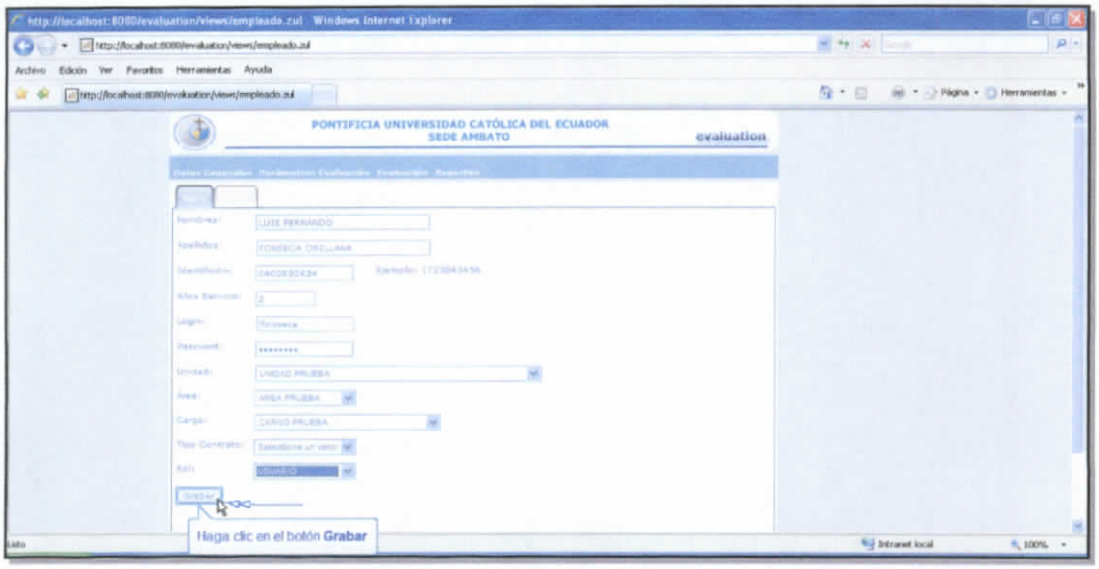


1.5 Página de empleados

Esta página se utiliza para la parametrización de los empleados que se emplearán en el proceso de evaluación. Para acceder a esta página se debe ingresar al menú de mantenimiento y seleccionar la celda de empleados.



Se ingresa la información del empleado, el cargo y la unidad. Procesos de inserción, actualización y eliminación son similares a los explicados en las páginas anteriores.



En esta página las áreas que se puedan utilizar dependen de la unidad seleccionada previamente. Adicionalmente, se puede manejar dos roles dentro del sistema que se describen a continuación:

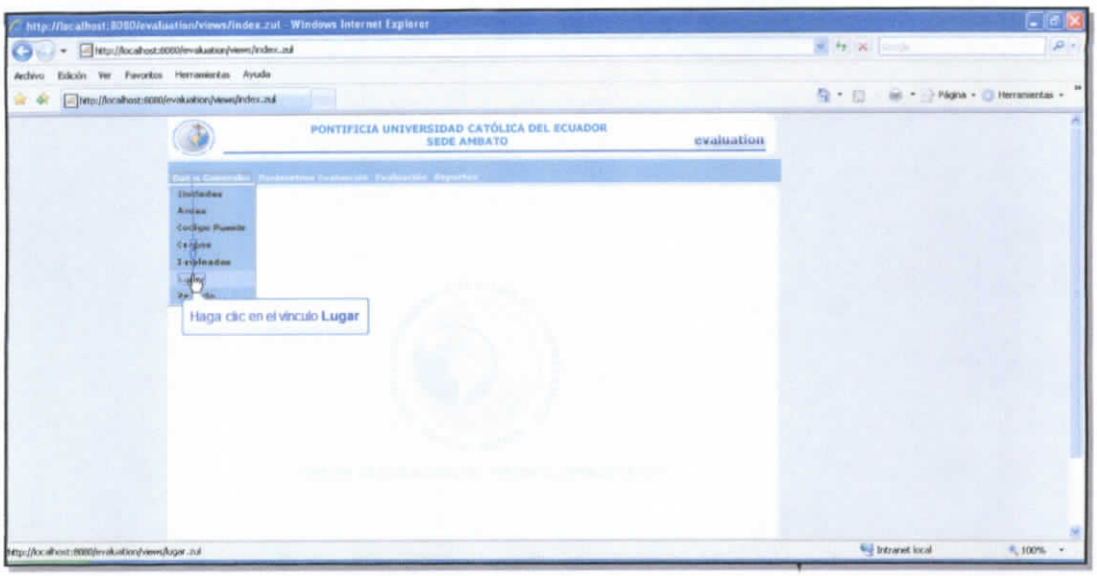
Administrador: Constituye el personal que se encargará de la parametrización del sistema y de otras actividades de administración, será integrado por el personal del área de Dirección Administrativa y RRHH.

Usuario: Constituye el personal que solamente puede realizar evaluaciones y puede ser evaluado, evaluador o ambos dentro del sistema.

1.6 Página de lugar

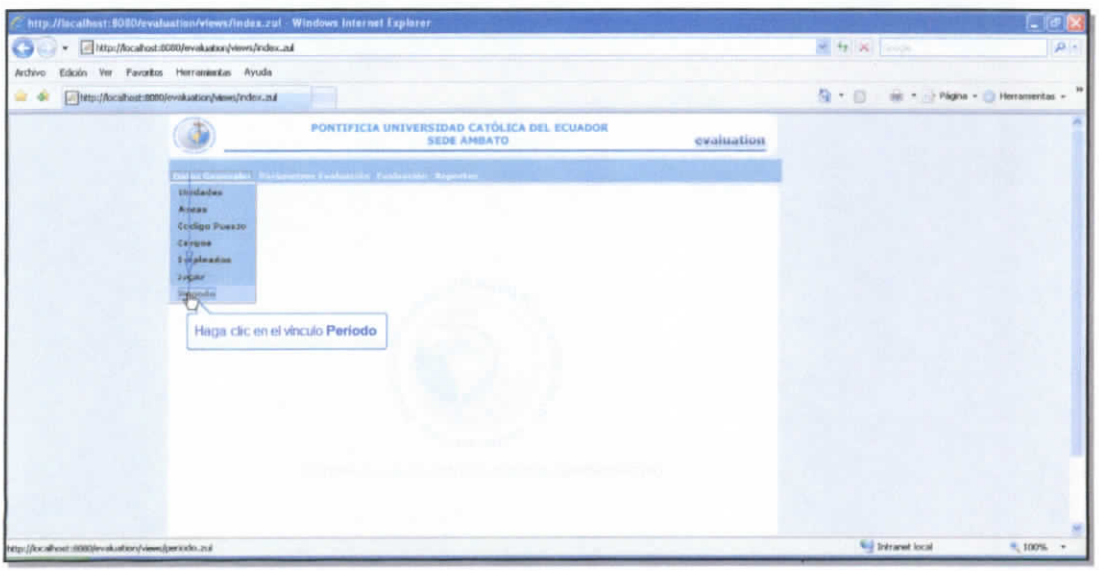
Esta página se utiliza para la parametrización de los lugares de puesto que se emplearán en el cronograma de evaluaciones. Para acceder a esta página se debe ingresar al menú de mantenimiento y seleccionar la celda de lugar.

Procesos de inserción, actualización y eliminación son similares a los explicados en las páginas anteriores.



1.7 Página de periodo

Esta página se utiliza para la parametrización de los periodos (semestres), las evaluaciones se realizan en un periodo específico. Para acceder a esta página se debe ingresar al menú de mantenimiento y seleccionar la celda de periodo.



Procesos de inserción, actualización y eliminación son similares a los explicados en las páginas anteriores.

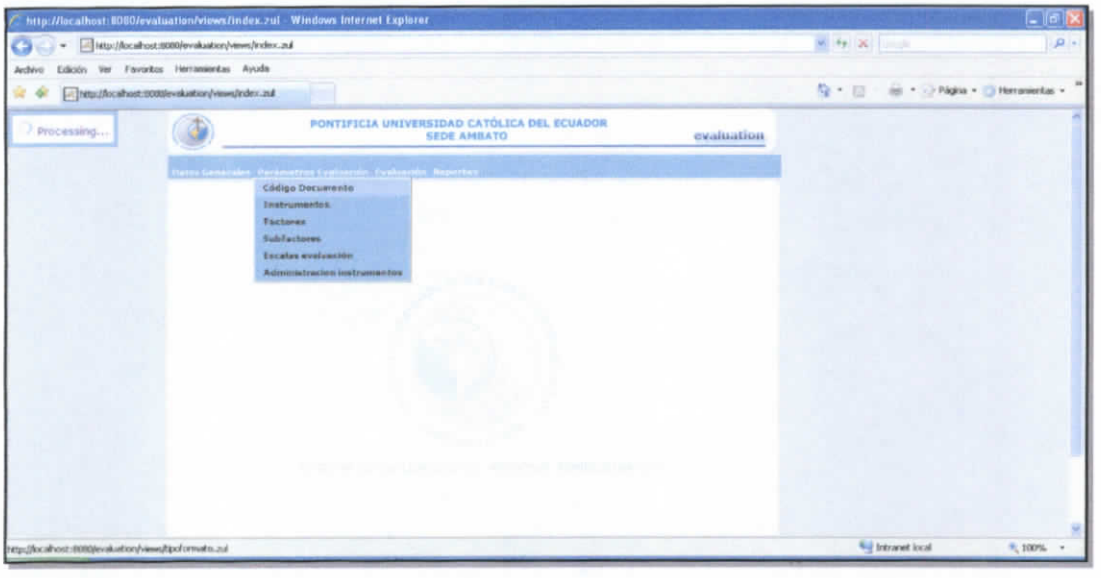
2. Sección de parámetros de evaluación

La sección de parámetros de evaluación de la aplicación es utilizada para definir todos los aspectos de las evaluaciones, esto incluye: instrumentos, escalas de calificación, instrumento, factores y subfactores.

2.1 Página de código instrumento

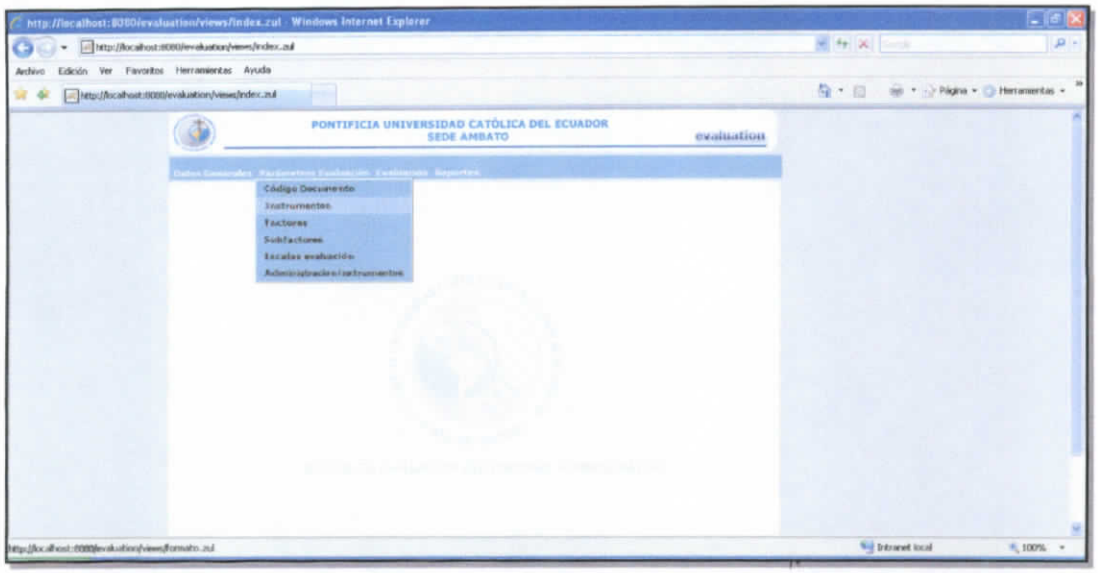
Esta página se utiliza para ingresar el código de los instrumentos de evaluación. Para acceder a esta página se debe ingresar al menú de parámetros de evaluación y seleccionar la celda de código de documento.

Procesos de inserción, actualización y eliminación son similares a los explicados en las páginas anteriores.



2.2 Página de instrumentos

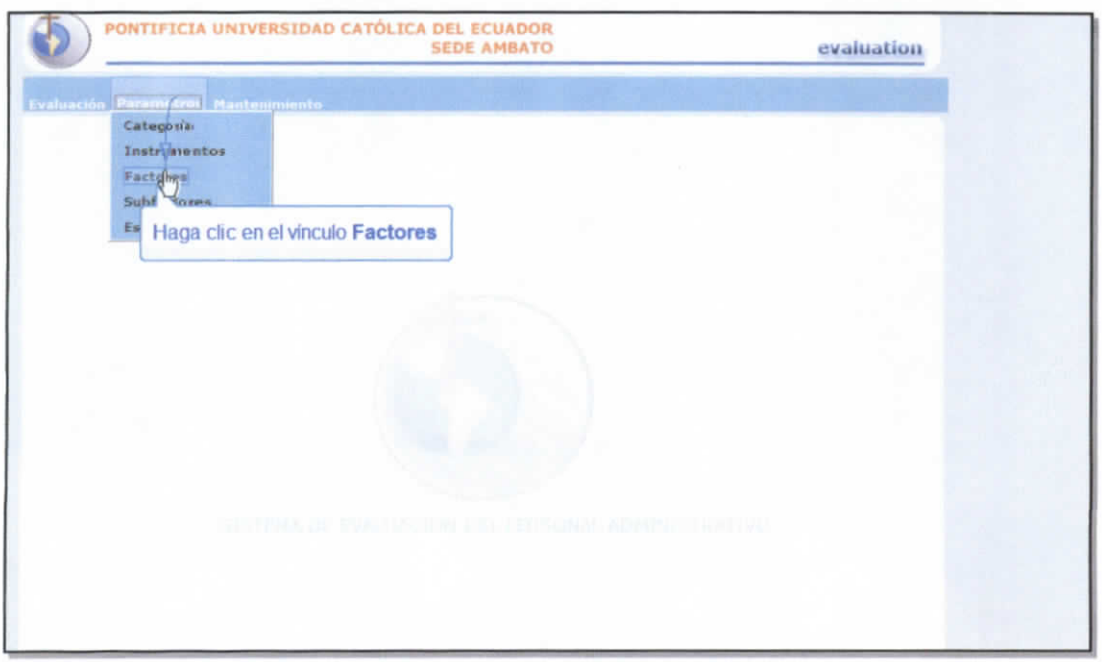
Esta página se utiliza para nombrar los instrumentos (documentos de evaluación) que se emplean en los procesos de evaluación. Para acceder a esta página se debe ingresar al menú de parámetros y seleccionar la celda de instrumentos.



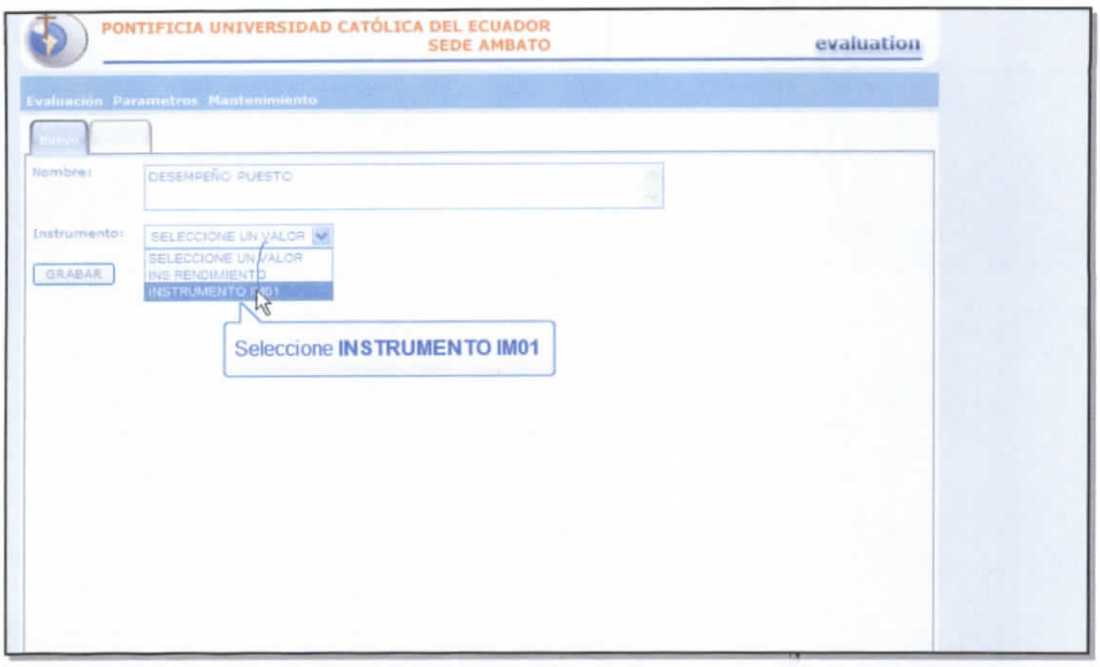
Procesos de inserción, actualización y eliminación son similares a los explicados en las páginas anteriores.

2.3 Página de Factores

Esta página se utiliza para ingresar los factores de evaluación. Para acceder a esta página se debe ingresar al menú de parámetros y seleccionar la celda de factores.

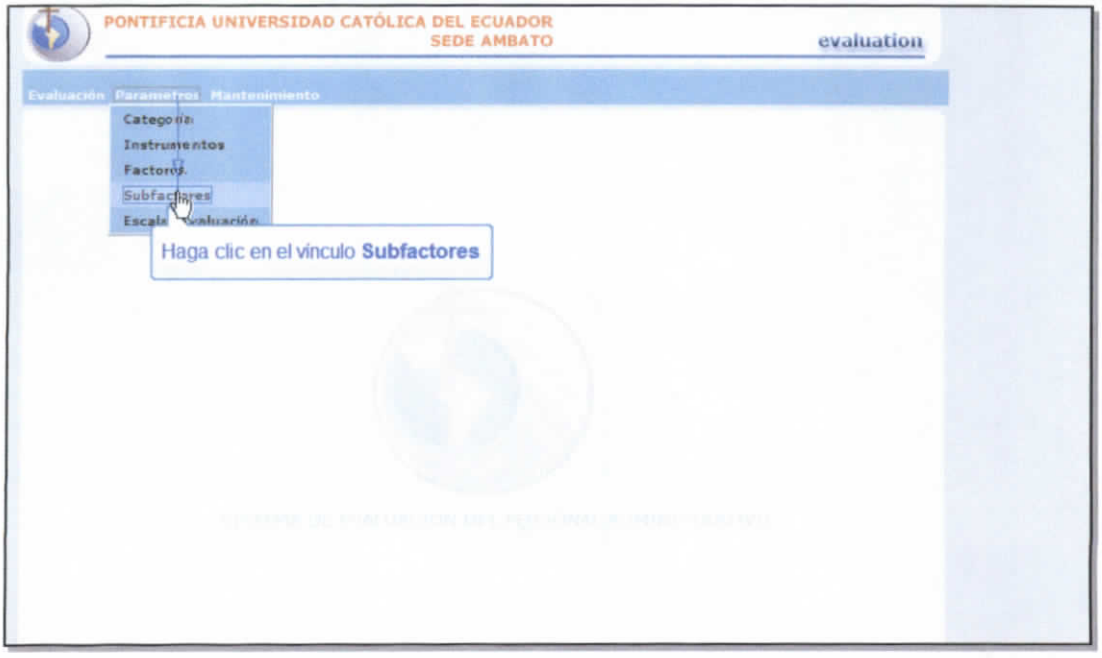


Procesos de inserción, actualización y eliminación son similares a los explicados en las páginas anteriores. Cada factor pertenece a un instrumento.

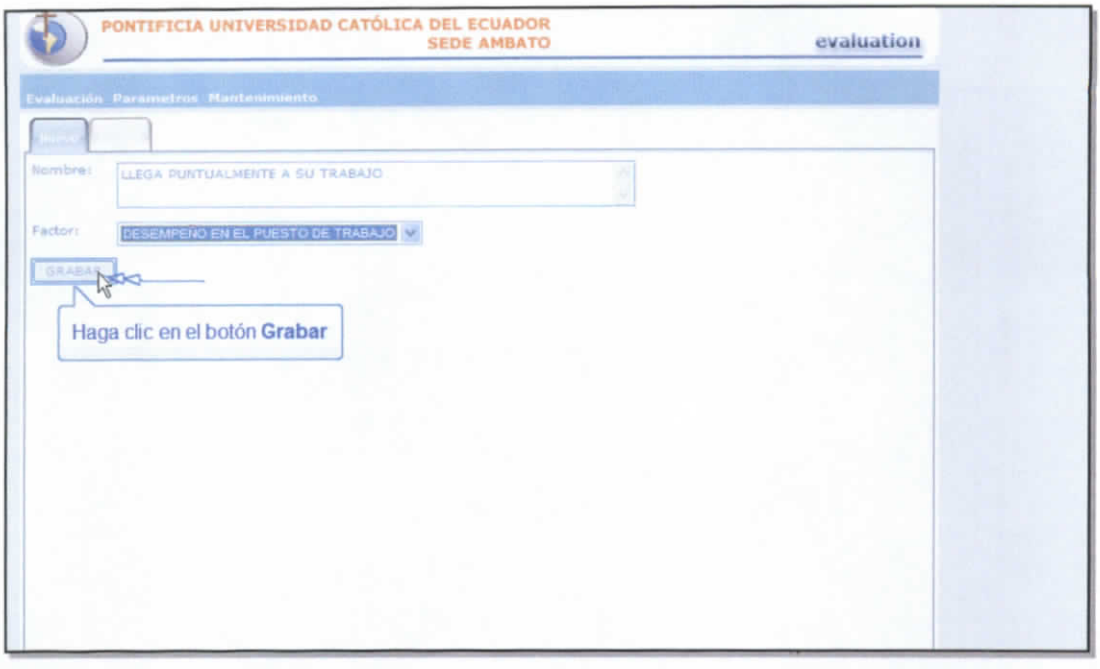


2.4 Página de subfactores

Esta página se utiliza para ingresar los subfactores de evaluación. Para acceder a esta página se debe ingresar al menú de parámetros y seleccionar la celda de subfactores.

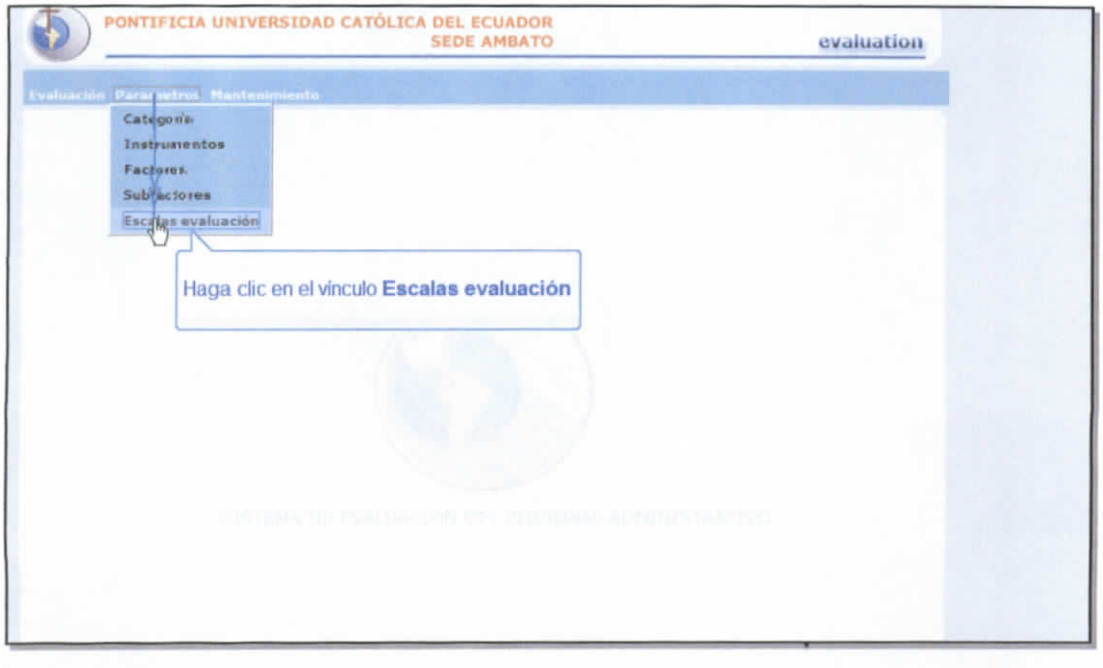


Procesos de inserción, actualización y eliminación son similares a los explicados en las páginas anteriores. Cada subfactor está asociado a un factor.

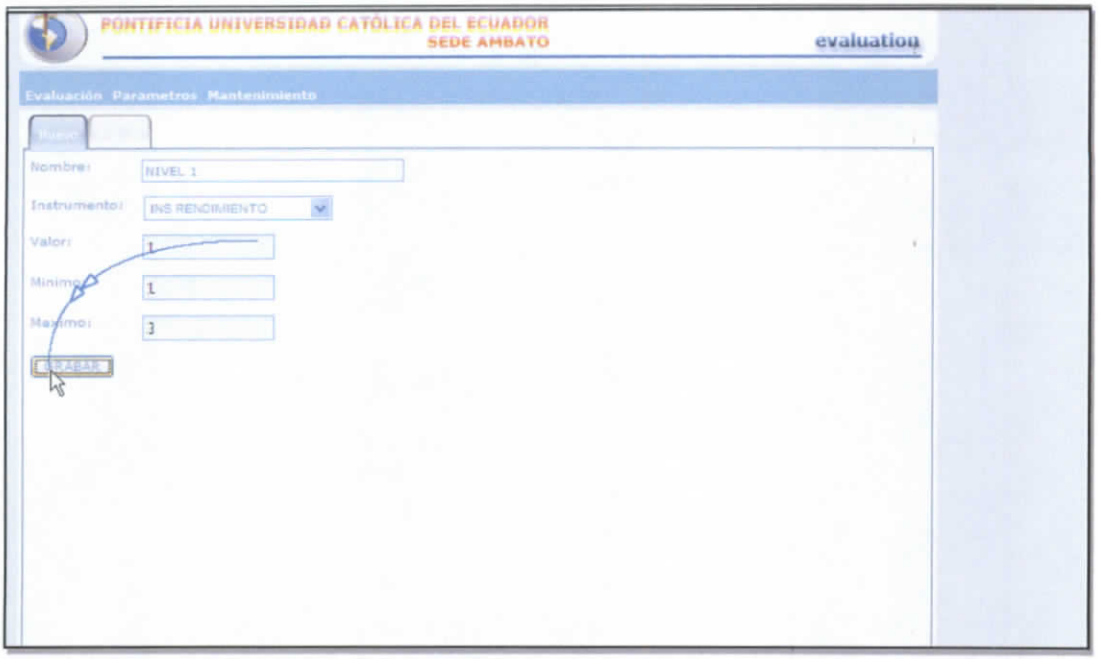


2.5 Página escalas de evaluación

Esta página se utiliza para ingresar los niveles de evaluación que se emplean en un instrumento, pueden haber diferentes escalas de evaluación para diferentes instrumentos. Para acceder a esta página se debe ingresar al menú de parámetros y seleccionar la celda de escalas de evaluación.

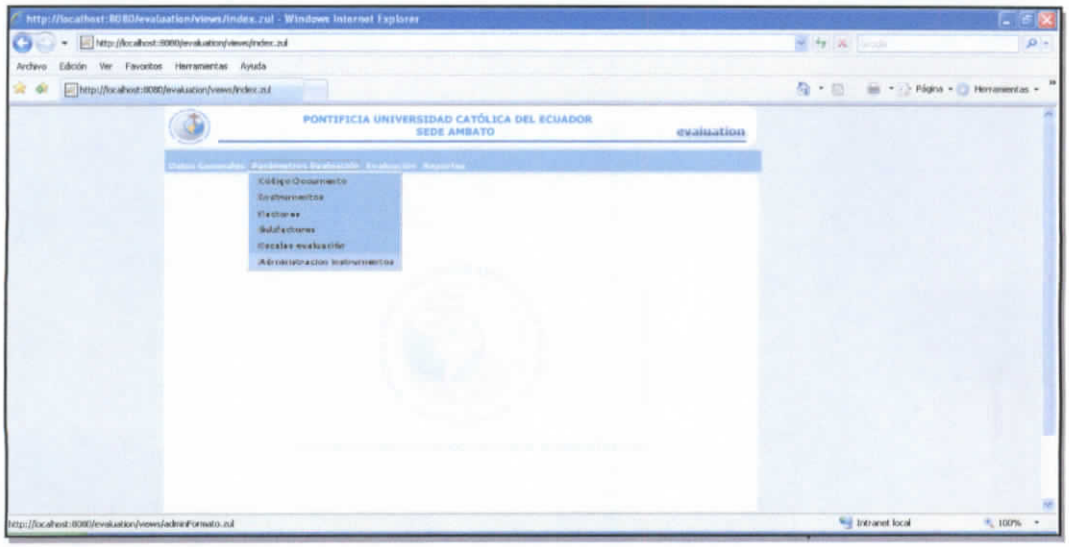


Procesos de inserción, actualización y eliminación son similares a los explicados en las páginas anteriores.

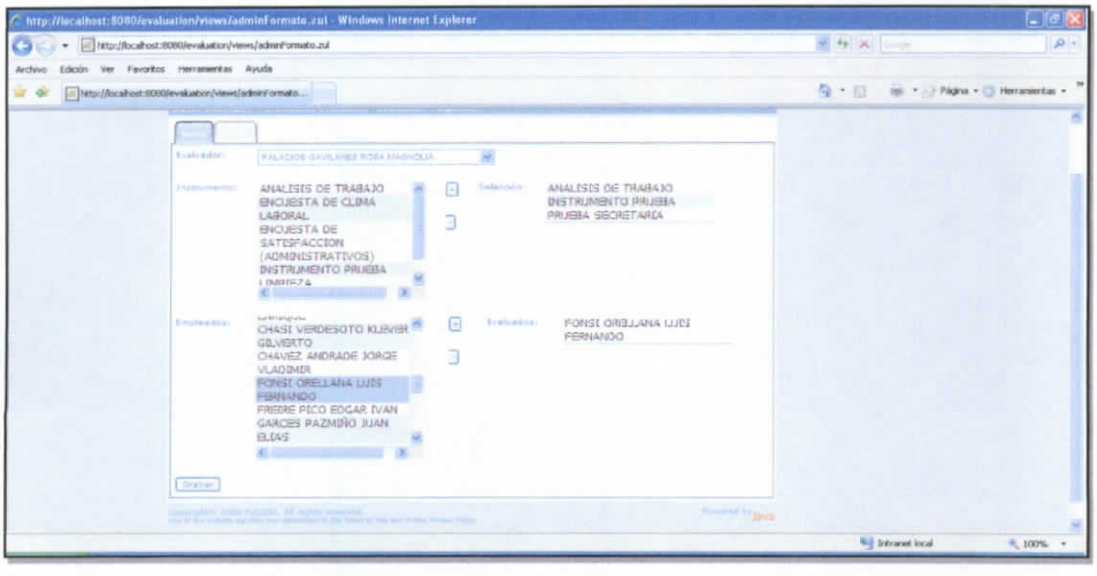


2.6 Página administración de documentos de evaluación (instrumentos)

Esta página se utiliza para administrar los instrumentos de evaluación. Los instrumentos se deben asociar a un evaluado (empleado a ser evaluado) y a un evaluador (empleado a realizar la evaluación). Para acceder a esta página se debe ingresar al menú de administración instrumentos.



Se debe seleccionar en el lado izquierdo de la lista de instrumentos los instrumentos que se quieren aplicar, se selecciona uno a uno cada instrumento y se da clic en el botón de adición (+) para añadirlo a la lista de instrumentos seleccionados al lado derecho. En el caso que requiera quitar de la lista de selección un instrumento se debe seleccionarlo aquí y dar clic en el botón de remoción (-).



El mismo proceso que se realiza con los instrumentos se aplica con los empleados que se deseen asociar. La lista de empleados puede ser de gran tamaño por lo que el sistema no muestra todos los empleado solo visualiza los veinte primeros (ordenados alfabéticamente). En caso que se requiera seleccionar un empleado no mostrado inicialmente, se debe dar clic en los botones inferiores de la lista que permite ir a los siguientes veinte empleados.

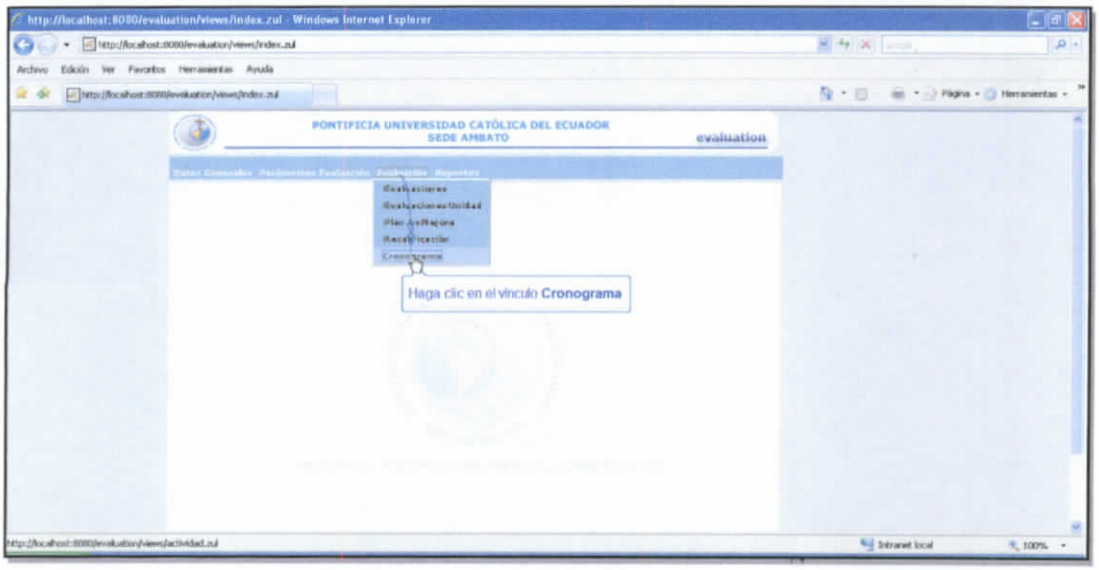
También, dentro de esta pantalla de administración, es necesario seleccionar la persona que se encargará de realizar la evaluación (evaluador) de los instrumentos seleccionados a los empleados asociados (seleccionados). Por último, el proceso termina con el clic en el botón de grabar. Procesos de edición, similares a los de las otras pantallas de mantenimiento explicadas con anterioridad.

3. Sección de evaluación

La sección de evaluación de la aplicación es utilizada para crear las evaluaciones, además permite realizar los cronogramas de evaluaciones y recalificaciones.

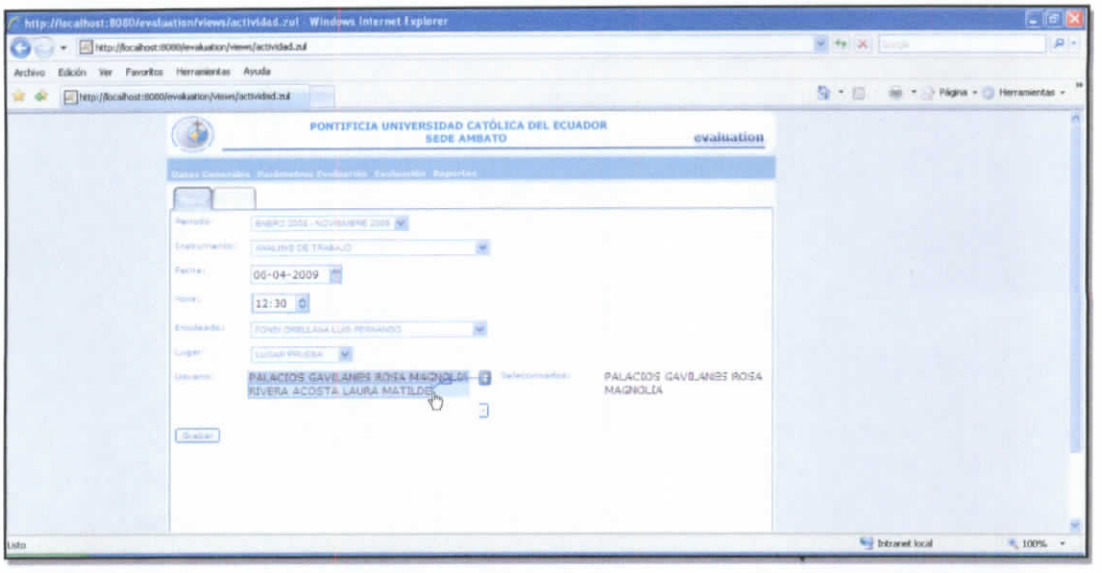
3.1 Página de cronograma

Esta página se utiliza para ingresar las actividades de evaluación y sus respectivas fechas y lugares en las que tendrán lugar. Para acceder a esta página se debe ingresar al menú de evaluación y seleccionar la celda de cronograma.



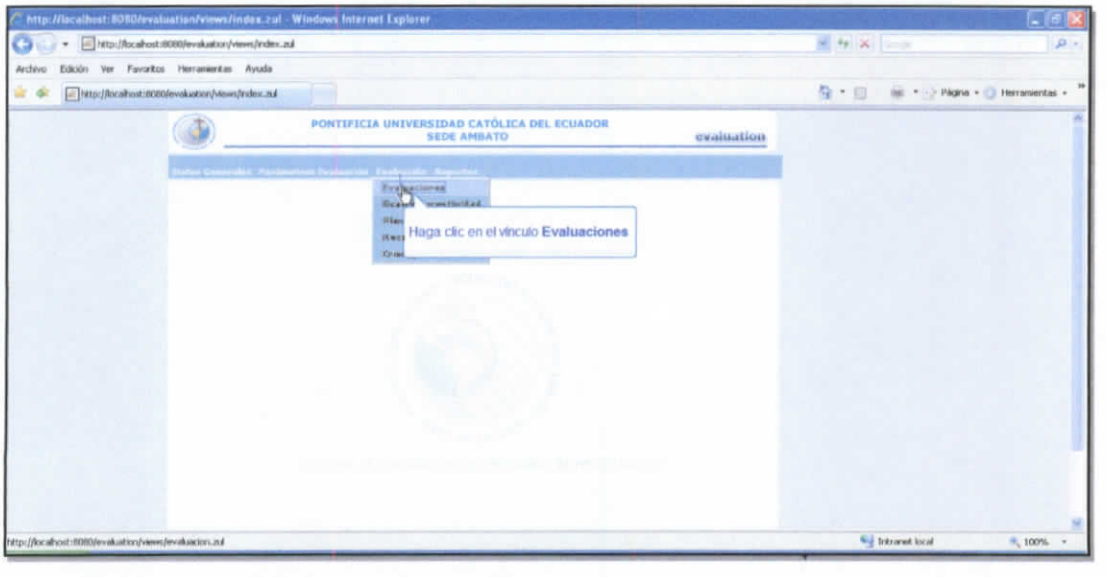
Procesos de inserción, actualización y eliminación son similares a los explicados en las páginas anteriores. Se deben incluir: el periodo (semestre), el equipo evaluador (quienes realizan la evaluación), el empleado (quien es evaluado), lugar y selección del instrumento a aplicarse.

El equipo evaluador (usuario) pueden ser varias personas, por lo que se puede seleccionar dos o más empleados que tengan rol de administrador. El proceso es similar el explicado en la selección de instrumentos en la pantalla de administración de instrumentos.

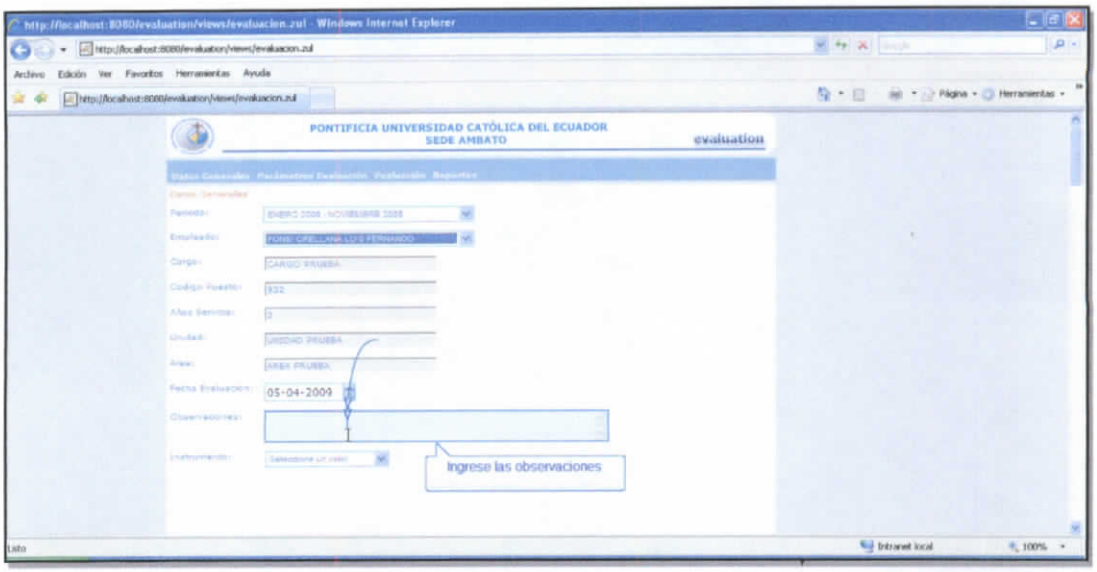


3.2 Página de evaluación

Esta página se utiliza para ingresar nuevas evaluaciones. Para acceder a esta página se debe ingresar al menú de evaluación y seleccionar la celda de evaluaciones.

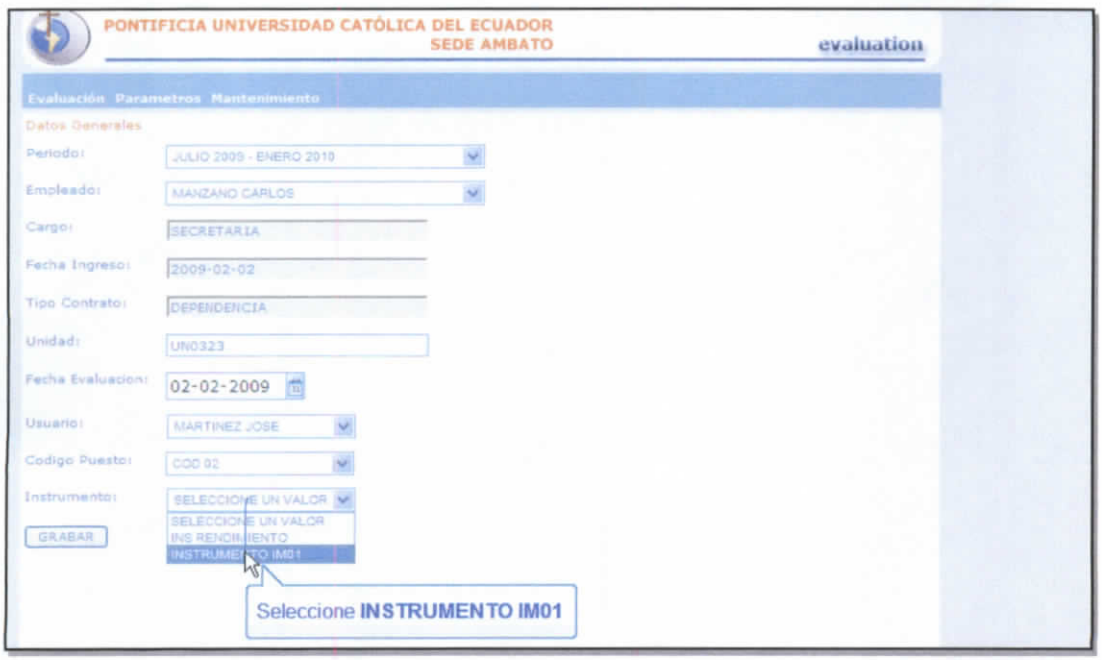


Se selecciona el periodo (semestre) de evaluación, el empleado que se va evaluar, el sistema automáticamente muestra la información detallada el empleado.

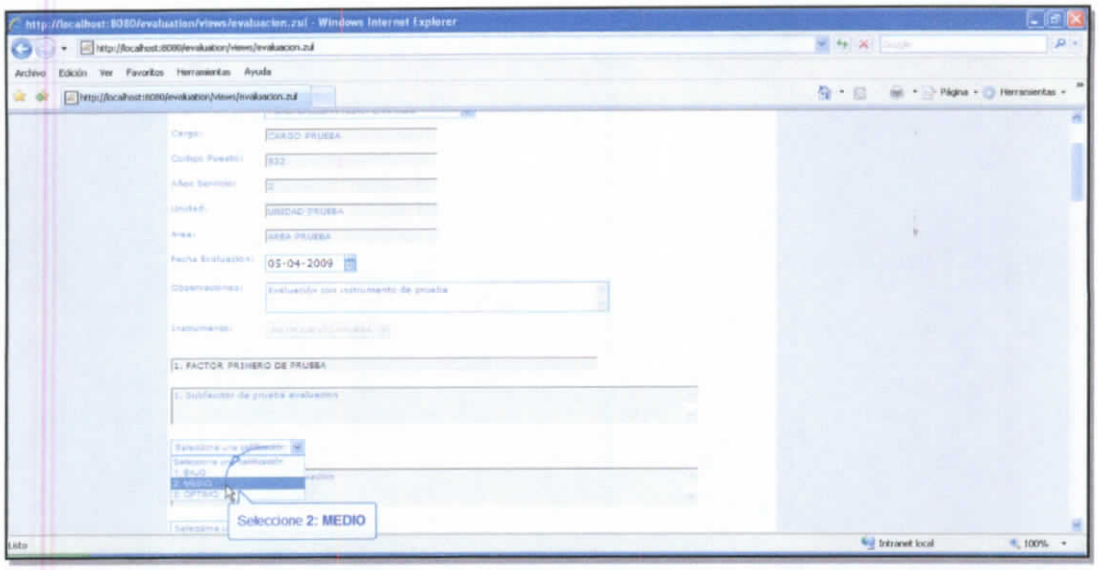


Se debe seleccionar el usuario quien realiza la evaluación, y el resto de datos necesarios.

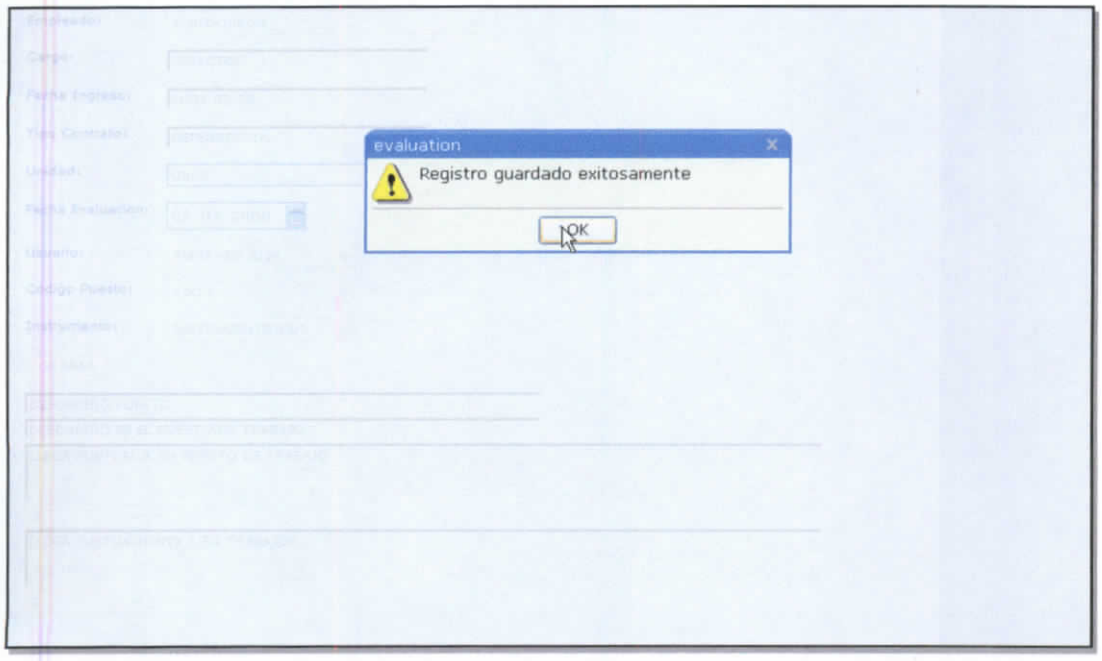
La página de evaluación es dinámica y forma las preguntas (formatos) de acuerdo al instrumento que se seleccione.



Automáticamente se forman los textos de formato y escalas (niveles de evaluación) para el instrumento seleccionado. Esta página también se emplea para todo tipo de evaluación (prácticas, desempeño, clima laboral) a excepción de servicios de unidad.

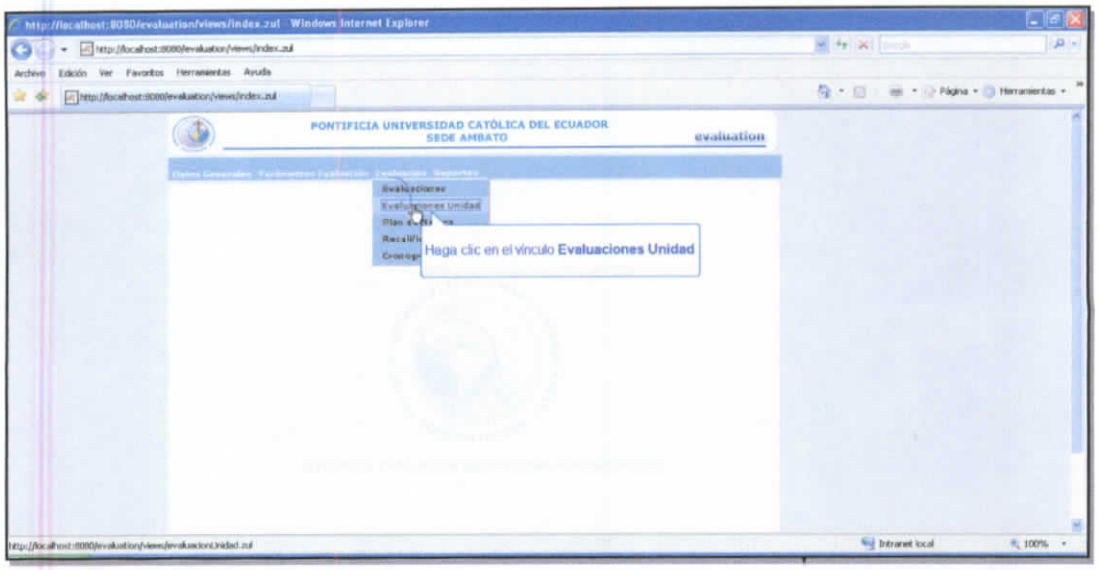


Se selecciona para cada subfactor la calificación otorgada dentro del nivel de evaluación y se procede a grabar la cotización.

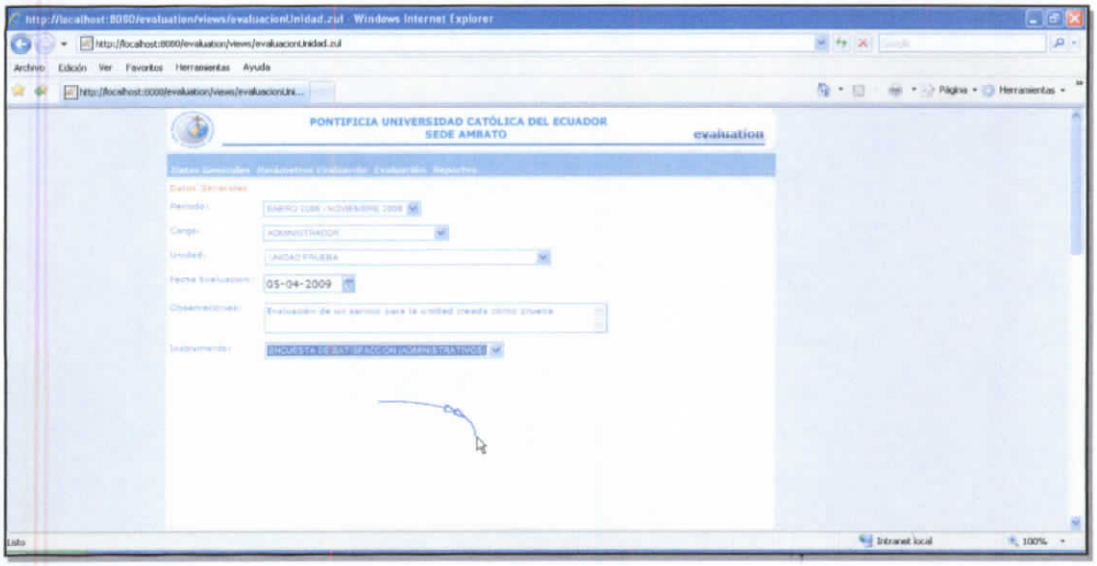


3.3 Página de evaluación unidad

Esta página se utiliza para evaluar los servicios de una unidad. Para acceder a esta página se debe ingresar al menú de evaluación y seleccionar la celda de evaluación unidad.



En esta página se debe seleccionar el periodo, la unidad a la que se va evaluar, el cargo que realiza la evaluación y el instrumento de evaluación de servicio que se va aplicar.

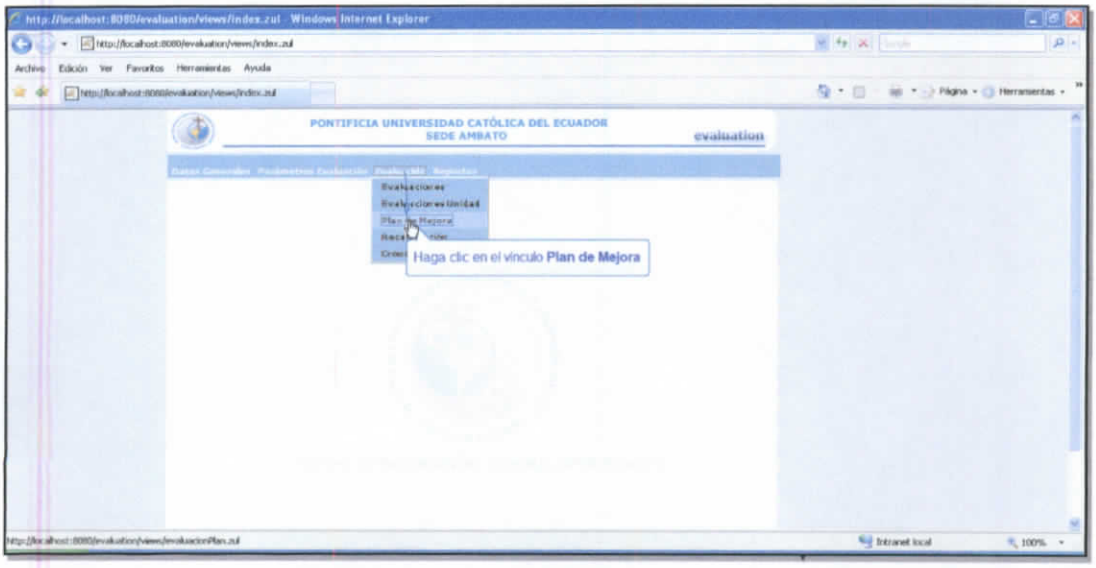


Después, de la selección del instrumento, el sistema generará dinámicamente las preguntas acordes al instrumento seleccionado así como también las escalas o niveles de evaluación referentes a este.

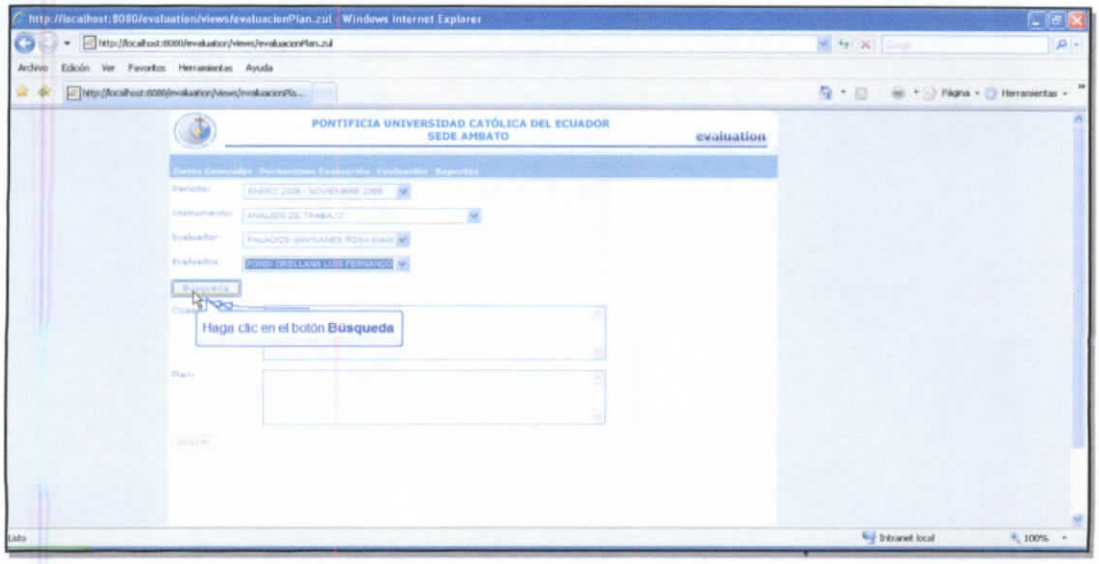
Una vez seleccionado el instrumento, ninguna pantalla de evaluación permite modificarlo, en el caso que sea necesario seleccionar otro instrumento, es necesario recargar la página ya sea a través de menú o el explorador de Internet.

3.4 Página de plan mejora

Esta página se utiliza para ingresar después del resultado de una evaluación, el plan de mejora propuesto por el evaluado, bajo la supervisión del evaluador. Para acceder a esta página se debe ingresar al menú de evaluación y seleccionar la celda de plan mejora.



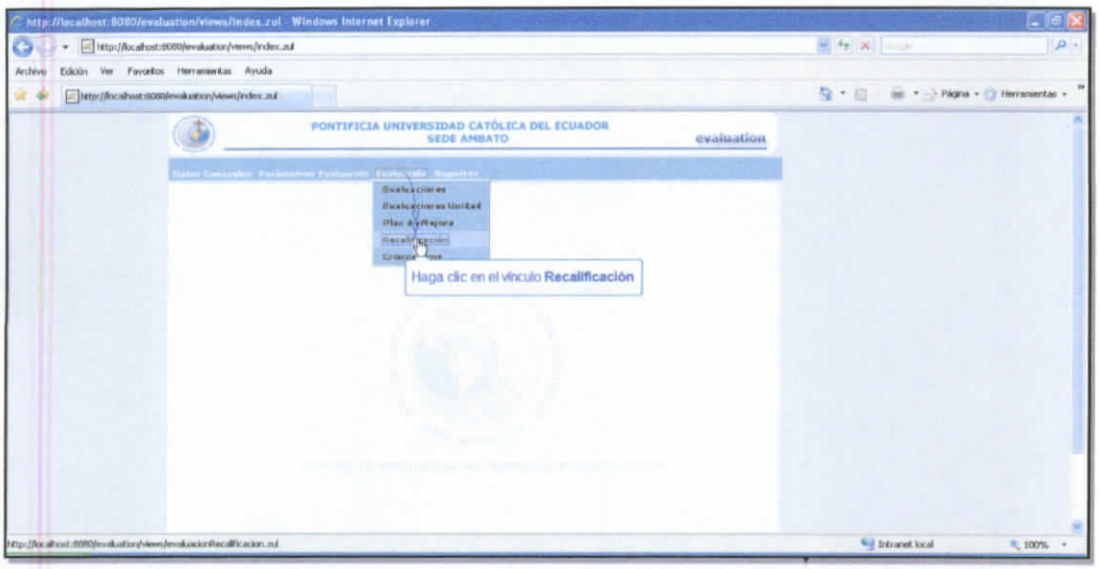
En esta pantalla se debe ingresar los datos de la evaluación y dar clic en el botón de búsqueda para poder cargar el plan de mejora.



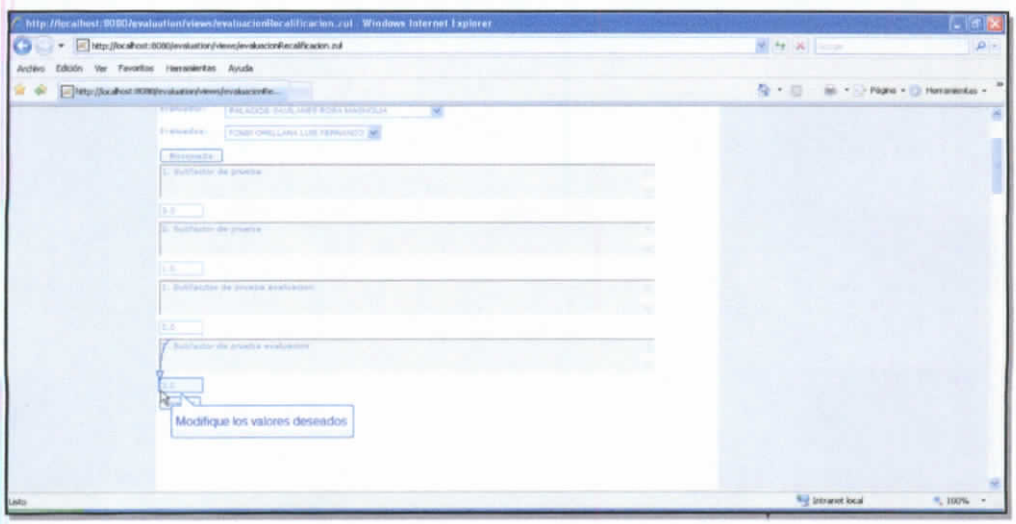
Una vez cargado el plan de mejora se procede a guardarlo.

3.5 Página de recalificación

Esta página se utiliza para modificar los resultados de una evaluación realizadas previamente, este proceso puede ser realizado únicamente por un administrador del sistema. Para acceder a esta página se debe ingresar al menú de evaluación y seleccionar la celda de recalificación.



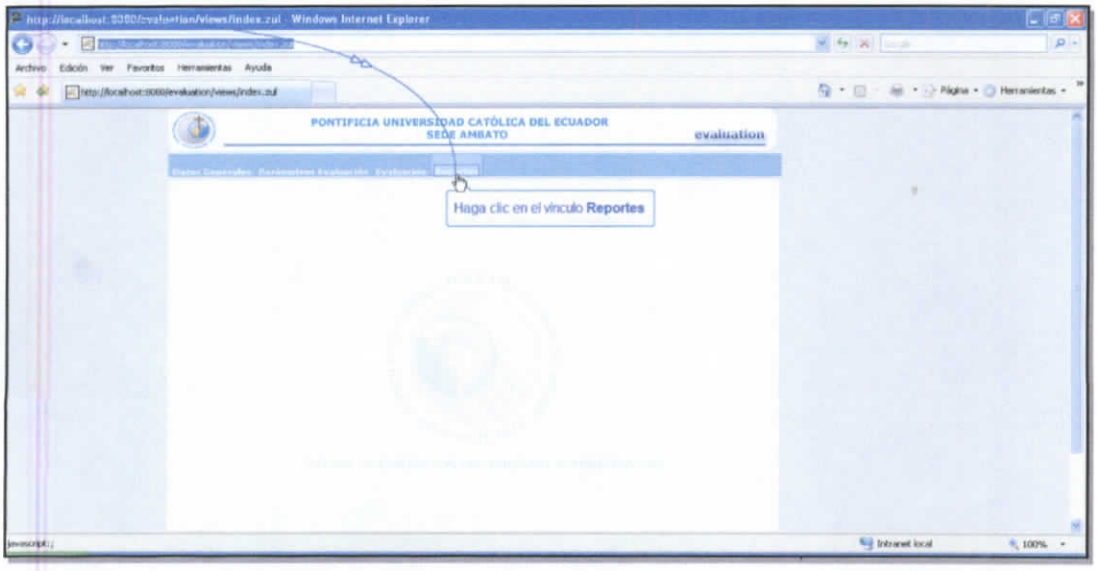
En esta pantalla se debe ingresar los datos de la evaluación y dar clic en el botón de búsqueda para poder cargar los resultados previos de la evaluación.



Los cambios se realizan editando los valores ya que no se dispone de los controles de selección.

4. Sección de reportes

La sección de reportes de la aplicación contiene todos los reportes que dispone el sistema, para la consulta de cualquier reporte es necesario acceder al menú principal a la última opción.

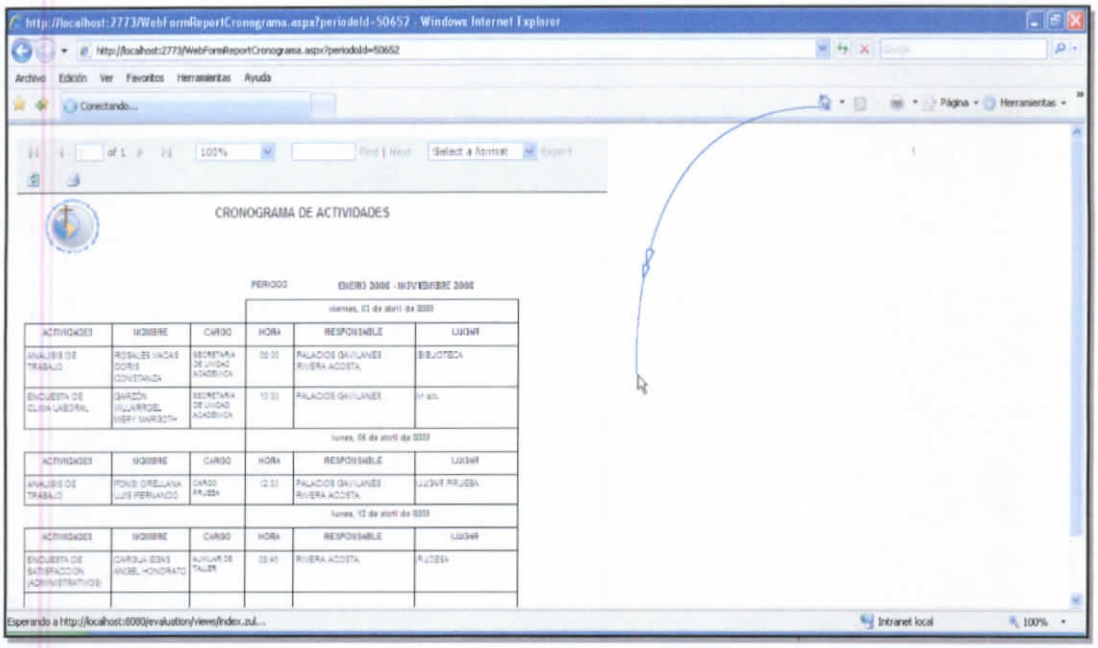


Para ejecutar cualquier reporte se requiere los siguientes parámetros:

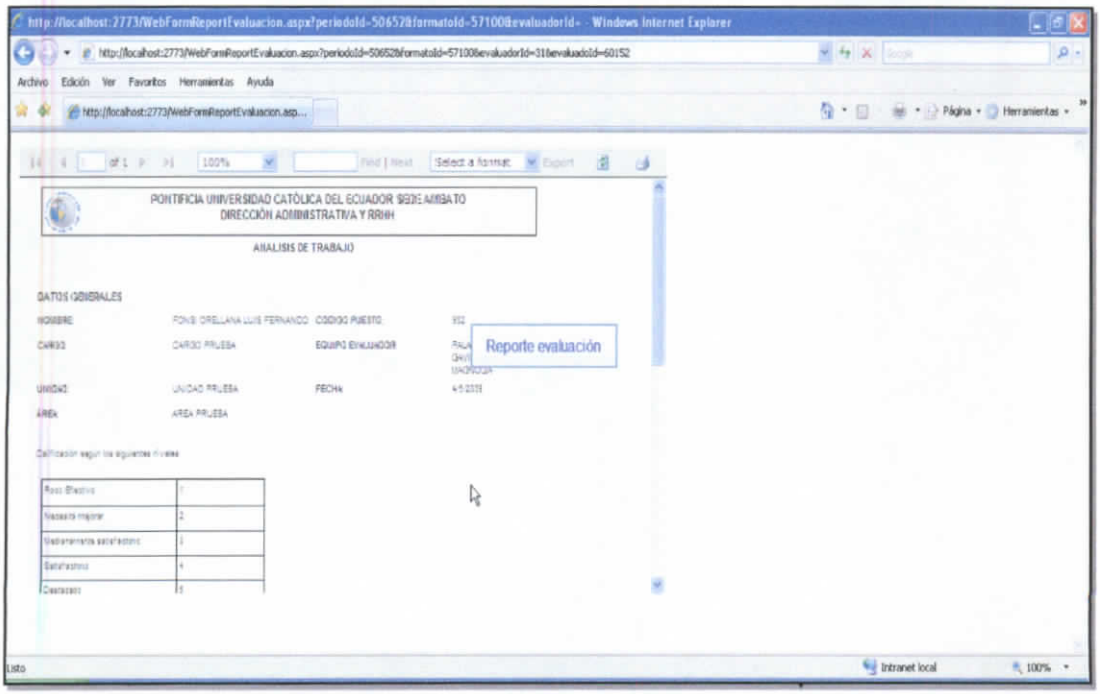
- Periodo: El periodo en la que se realizó una evaluación.
- Instrumento: El instrumento empleado en la evaluación.
- Evaluador: La persona que realizo la evaluación.
- Evaluado: La persona evaluada.

El sistema incluye los siguientes reportes:

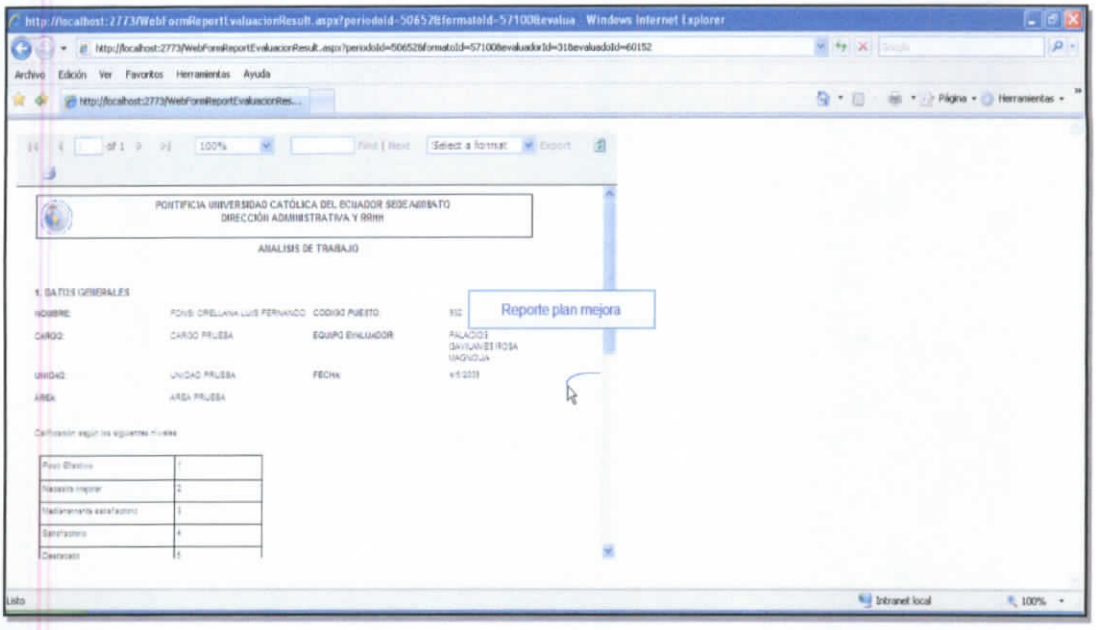
Reporte de cronograma:



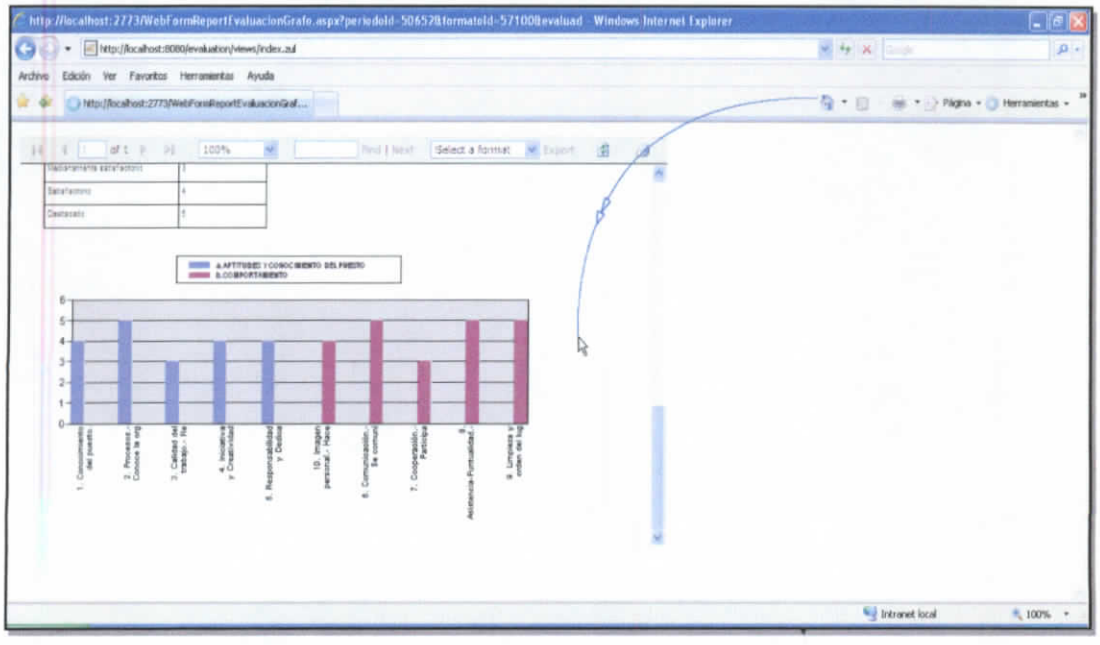
Reporte resultado evaluación detallado



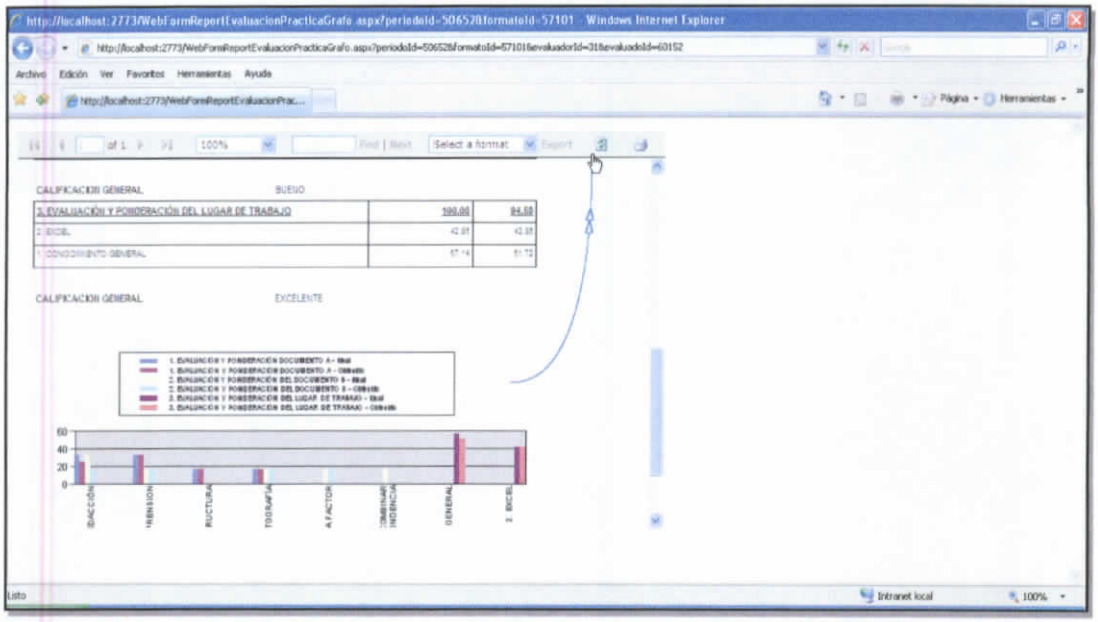
Reporte plan de mejora



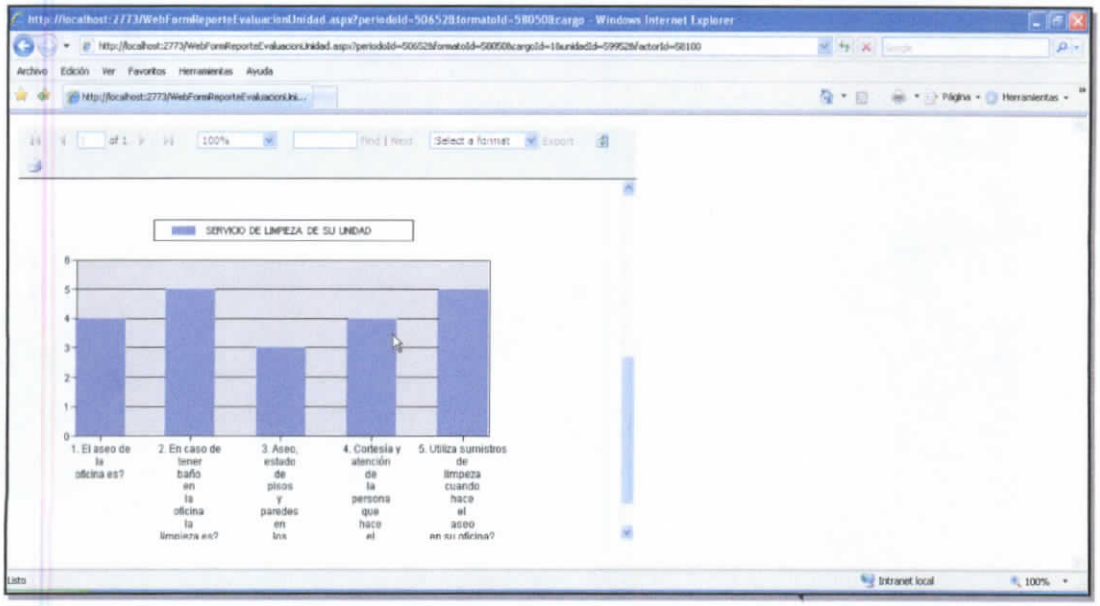
Reporte resultados estadísticos



Reporte evaluación práctica



Reporte evaluación servicio unidad detallado



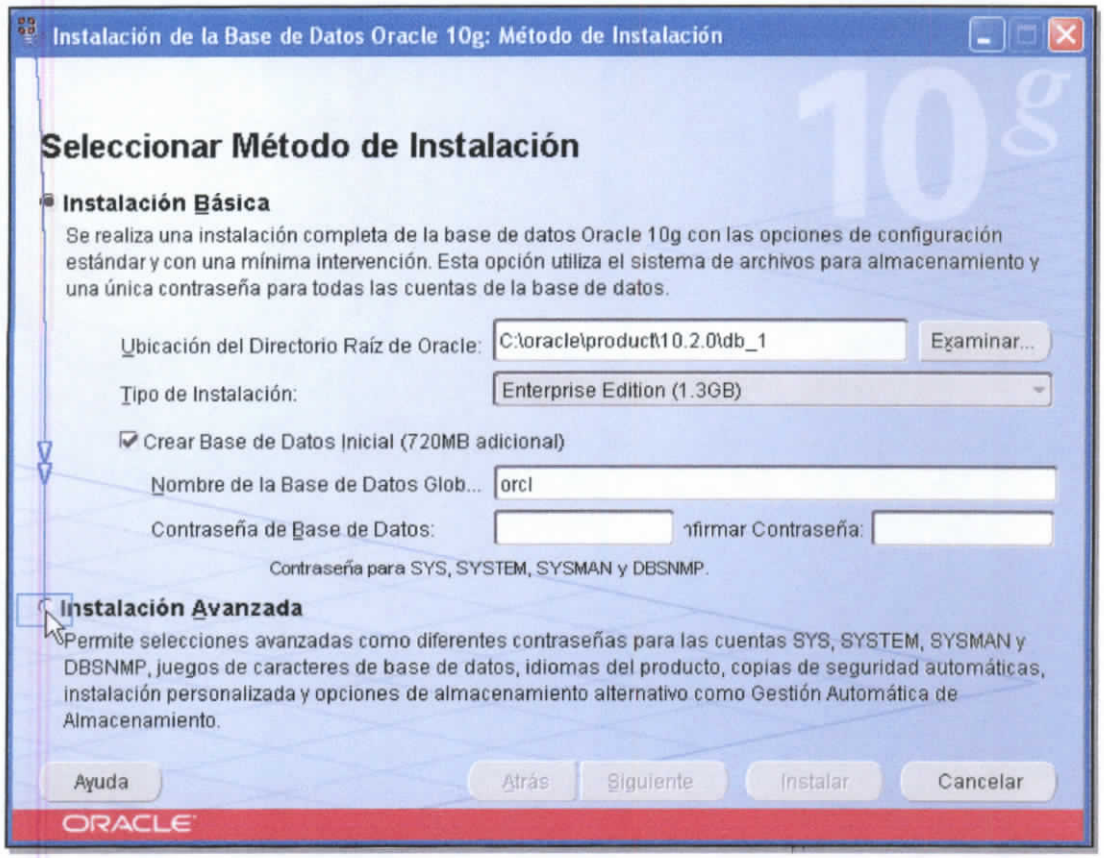
ANEXO 3

GUÍA DE INSTALACIÓN DEL SISTEMA DE EVALUACIÓN DEL PERSONAL ADMINISTRATIVO DE LA PUCESA

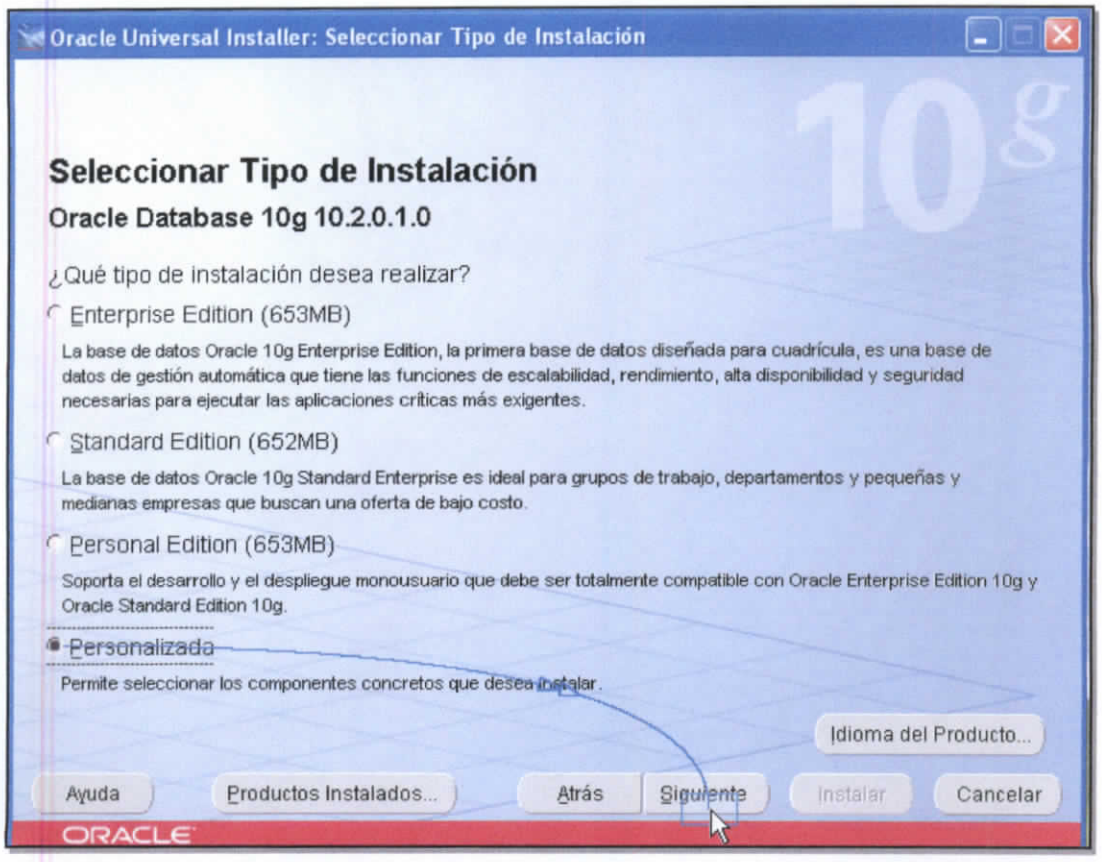
1. Instalación de Oracle

Para la implementación del sistema de evaluación del personal administrativo de la PUCESA, se requiere instalar un servidor Oracle que incluya el esquema de la base necesario para el funcionamiento de la aplicación. A continuación se detallan los pasos necesarios para esta instalación:

En la primera pantalla de selección se selecciona instalación avanzada.



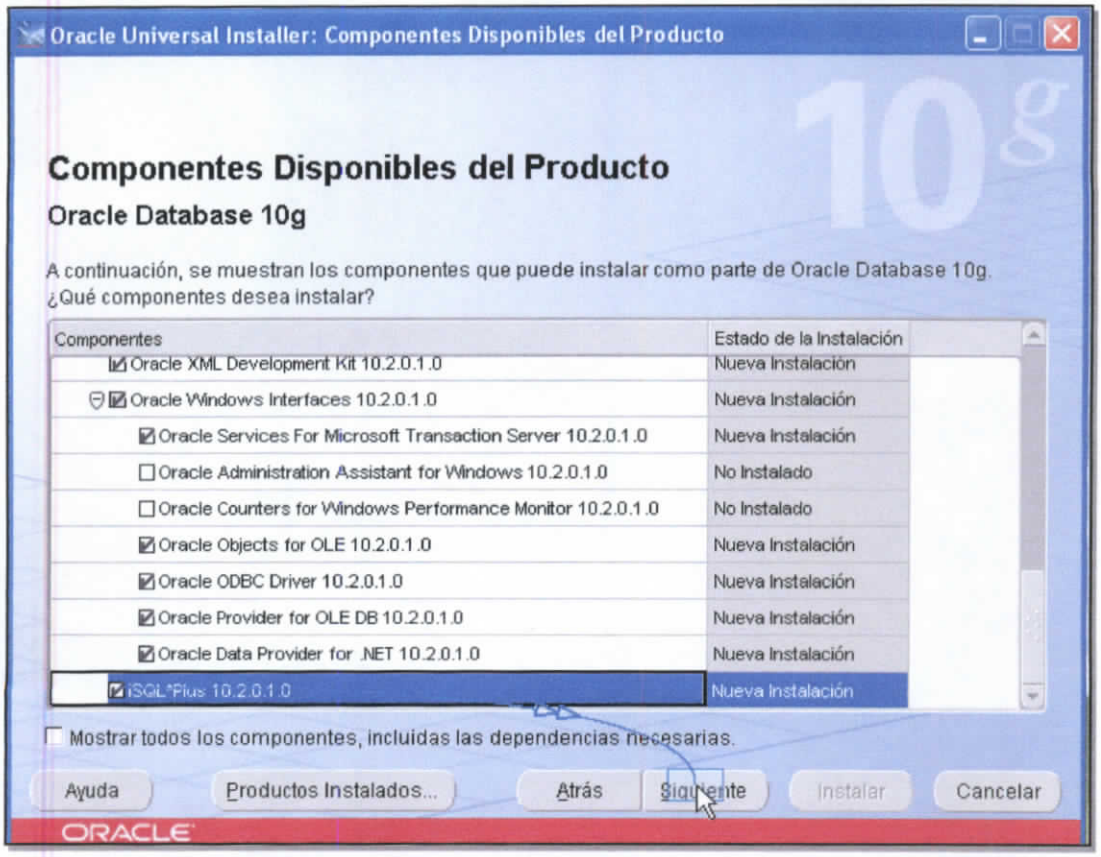
En el tipo de instalación se selecciona personalizada.



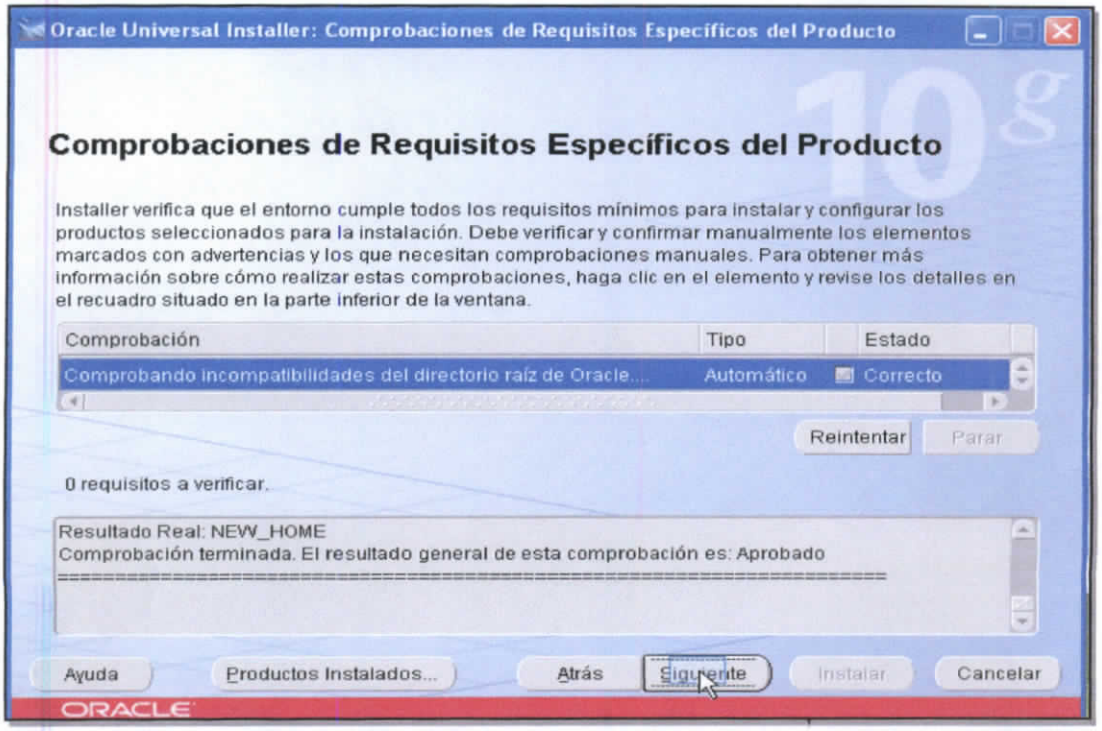
Se especifica el directorio donde residirá el servidor.



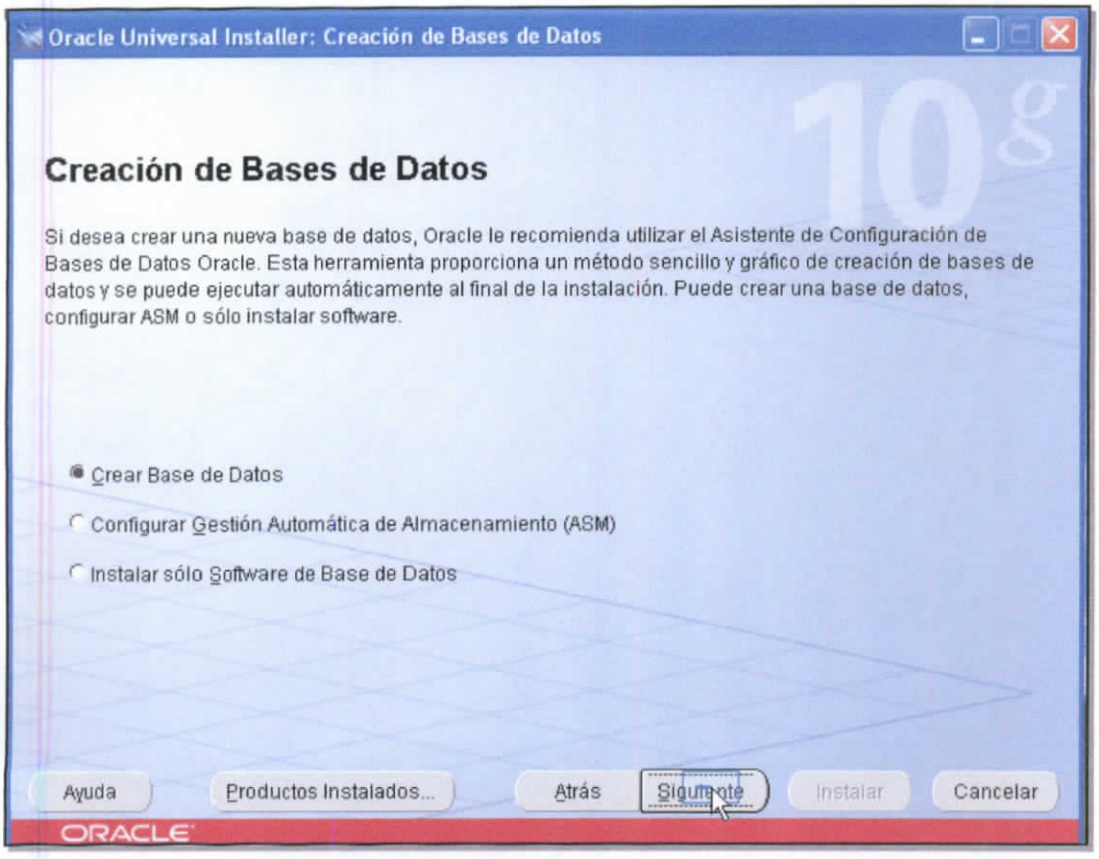
Se selecciona los componentes que se desean instalar. A más de los de defecto se debe seleccionar Windows Service Interfaz e ISQL plus para administración.



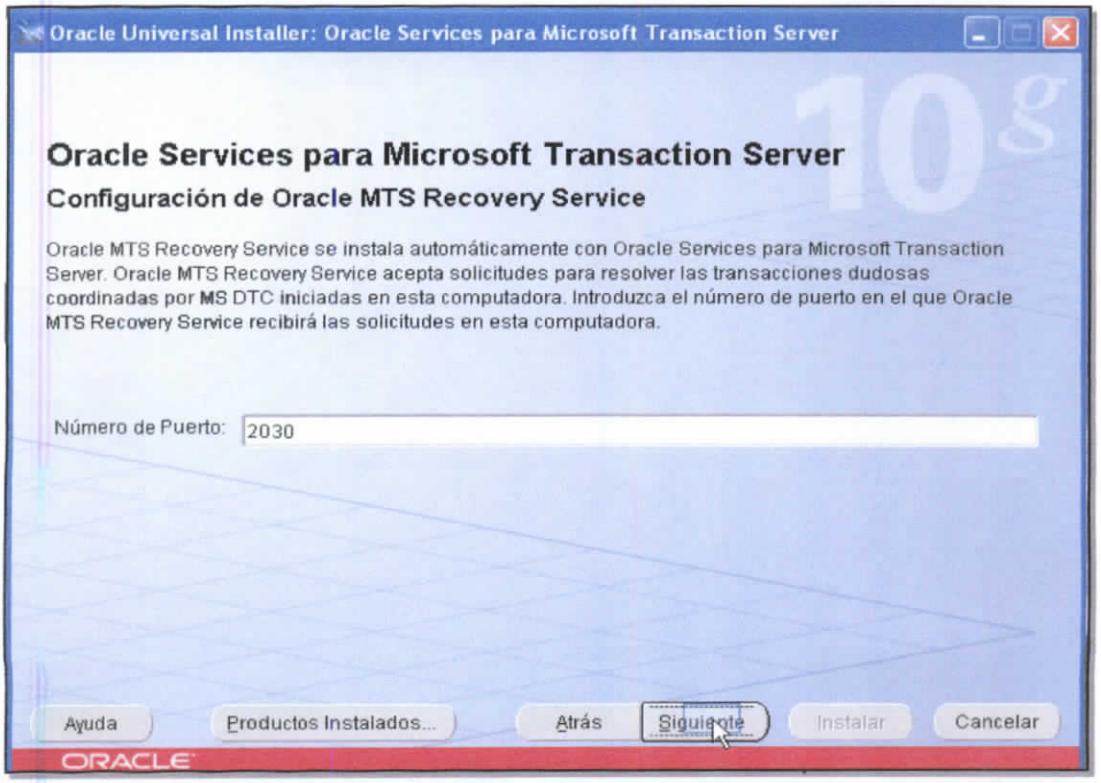
Oracle valida que los requerimientos de instalación sean correctos.



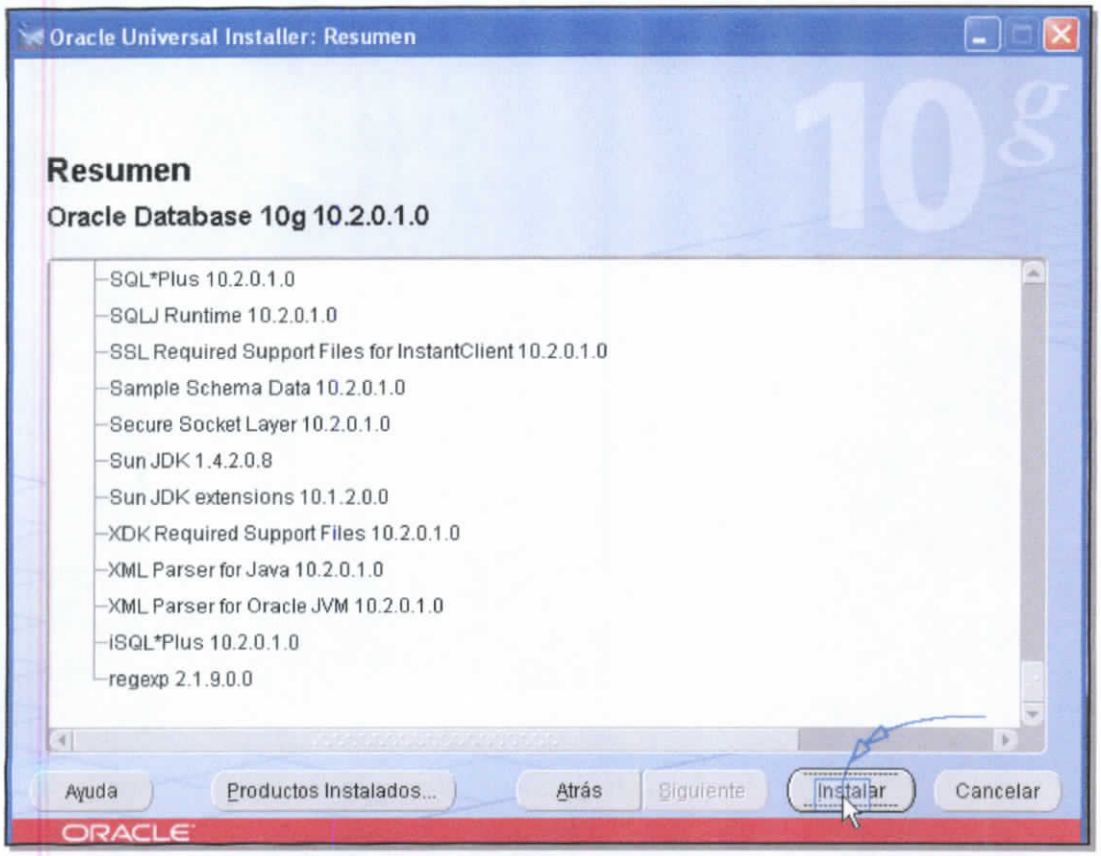
Se debe seleccionar creación de base de datos.



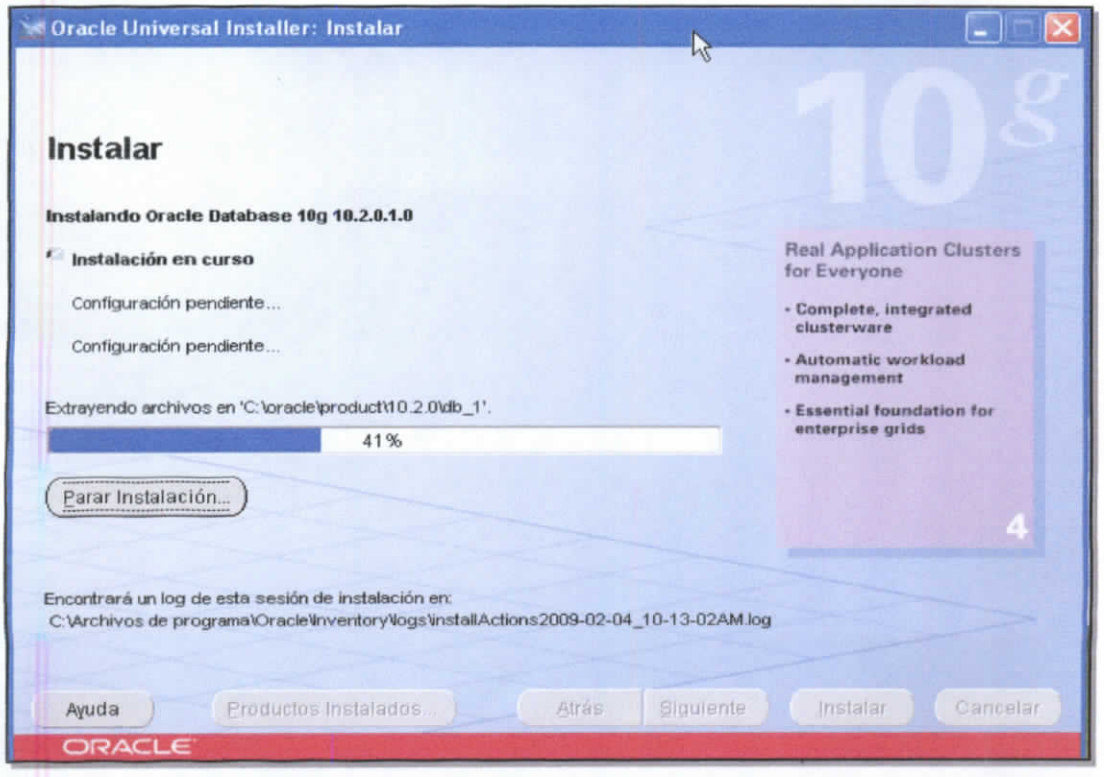
Se confirma que el puerto MTS sea el 2030



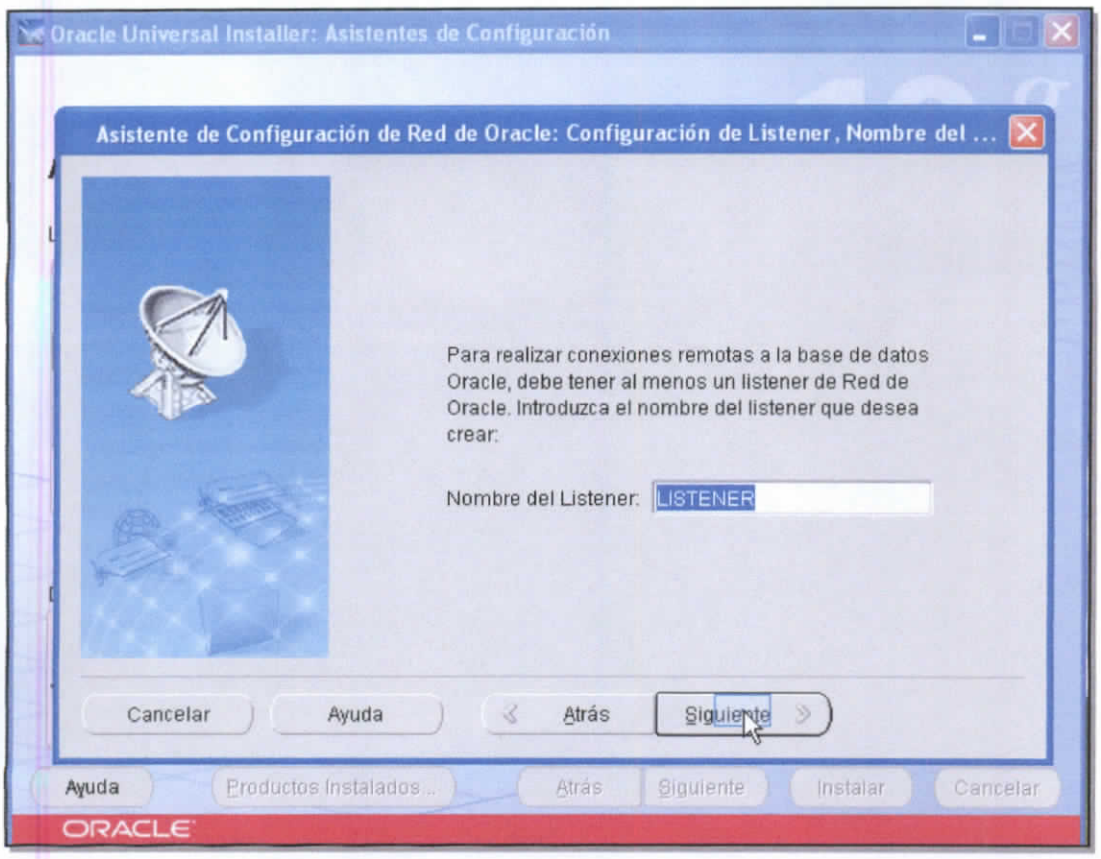
Se procede a la instalación.



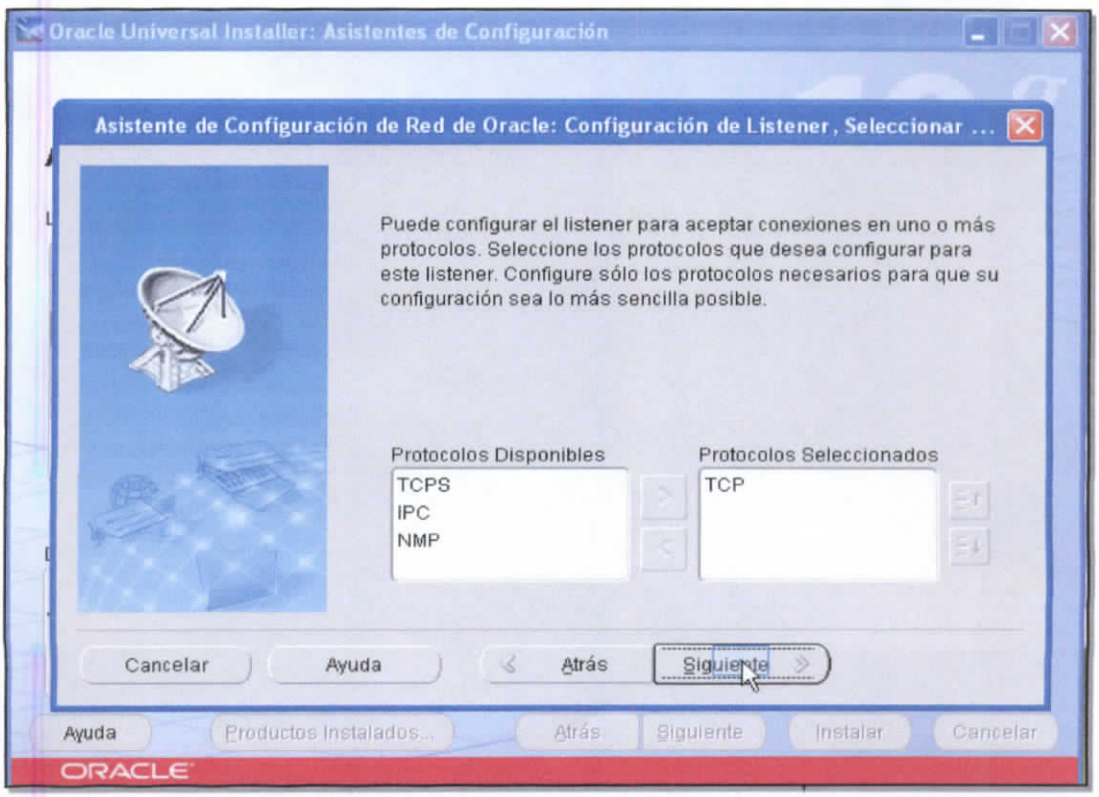
Proceso de instalación iniciado.



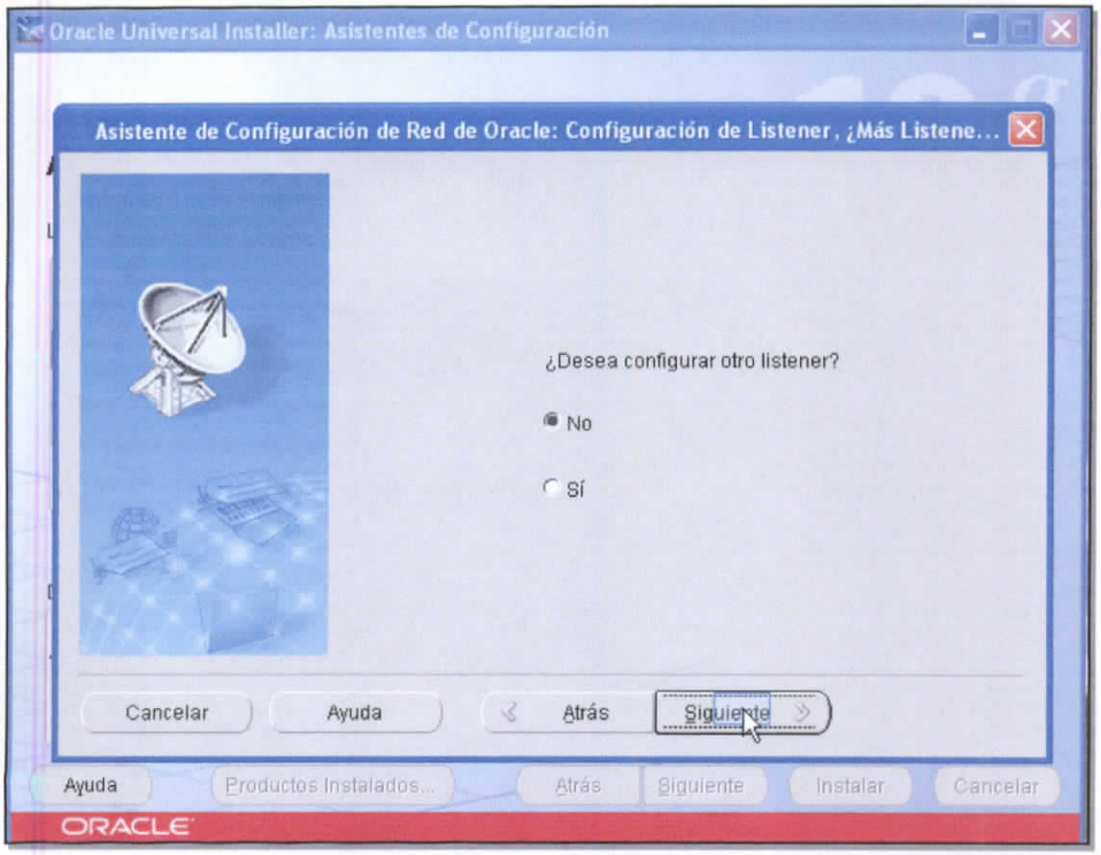
Se procede a la configuración del Listener (Configuración por defecto).



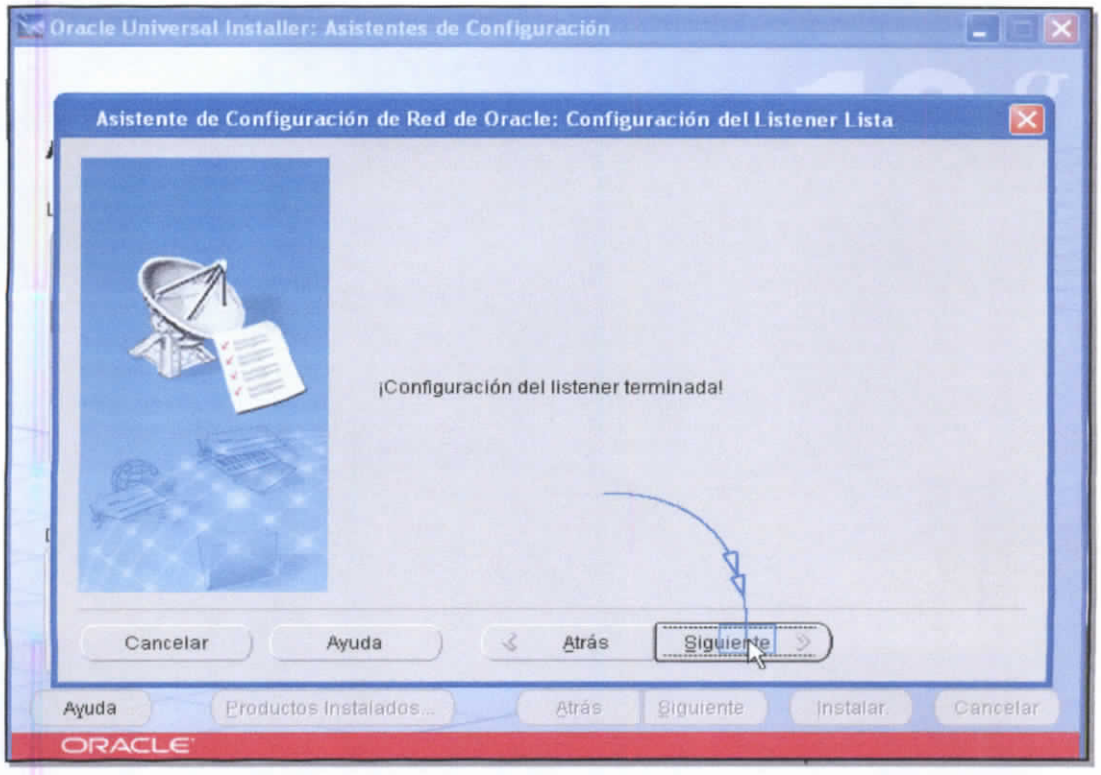
Se confirma por defecto protocolo TCP.



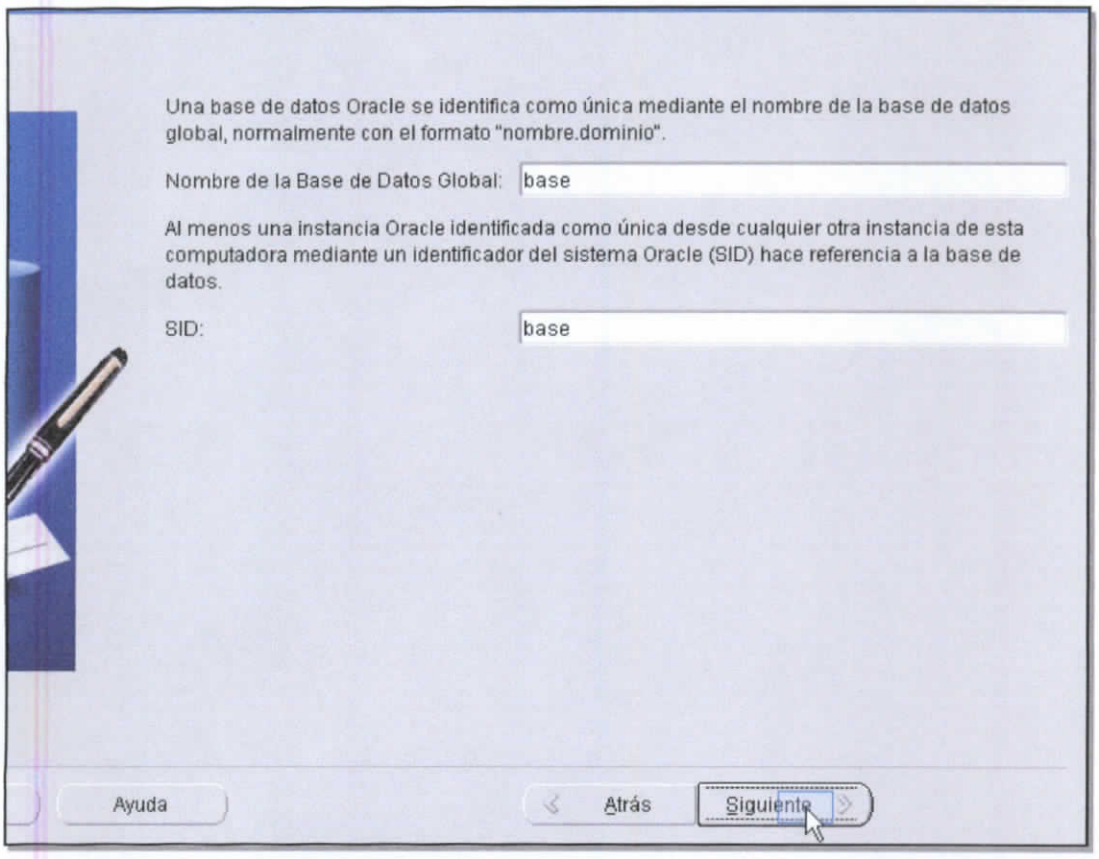
Se confirma puerto por defecto.



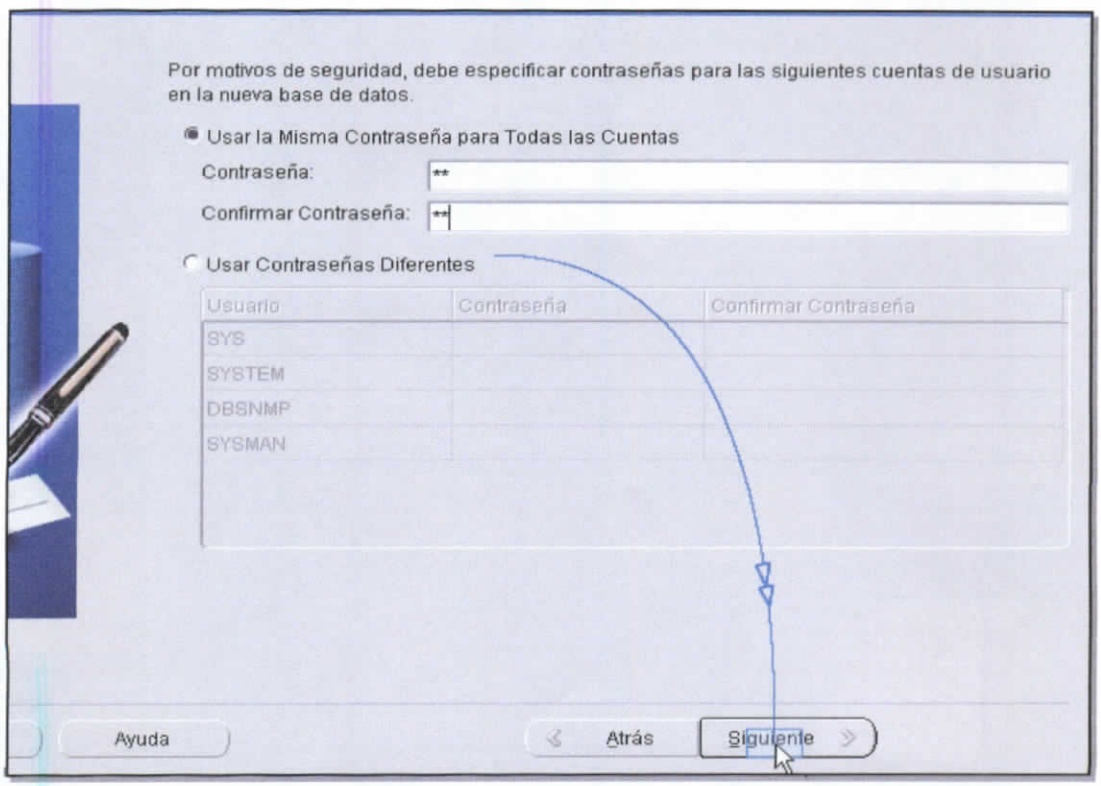
Se finaliza con la configuración del Listener.



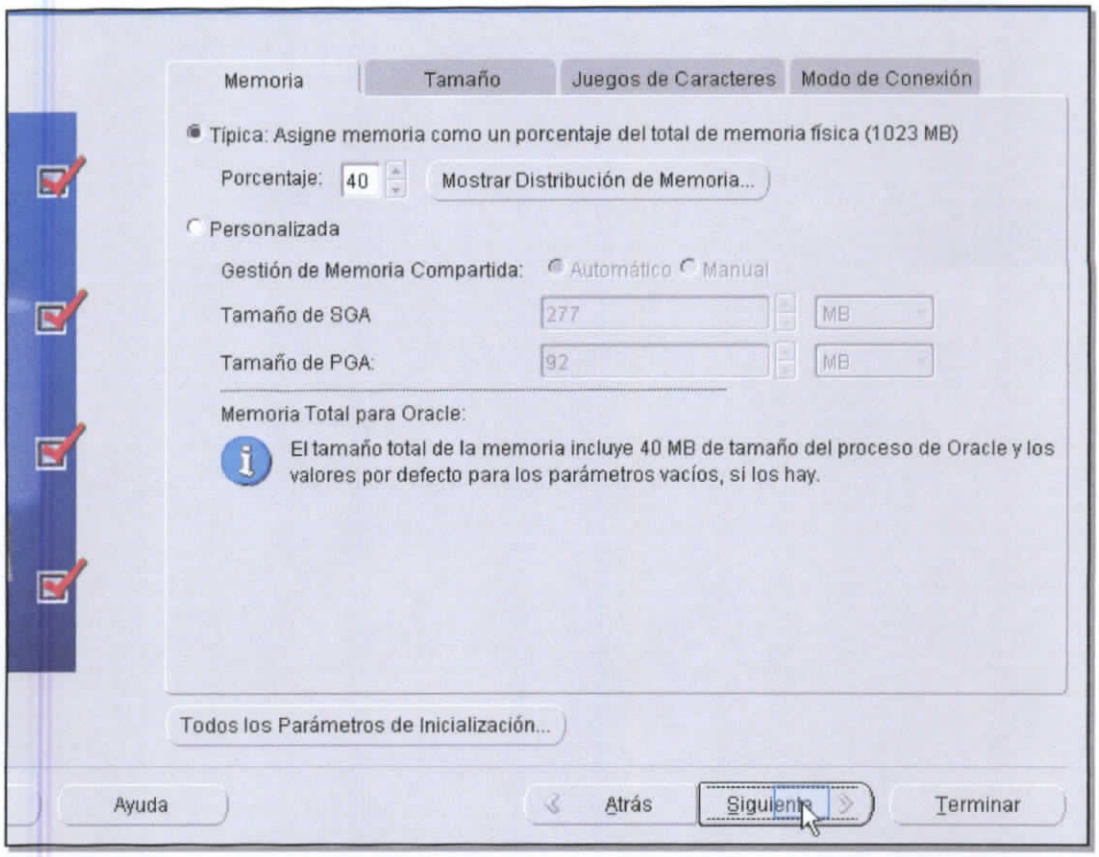
Aparece asistente para creación de base de datos.



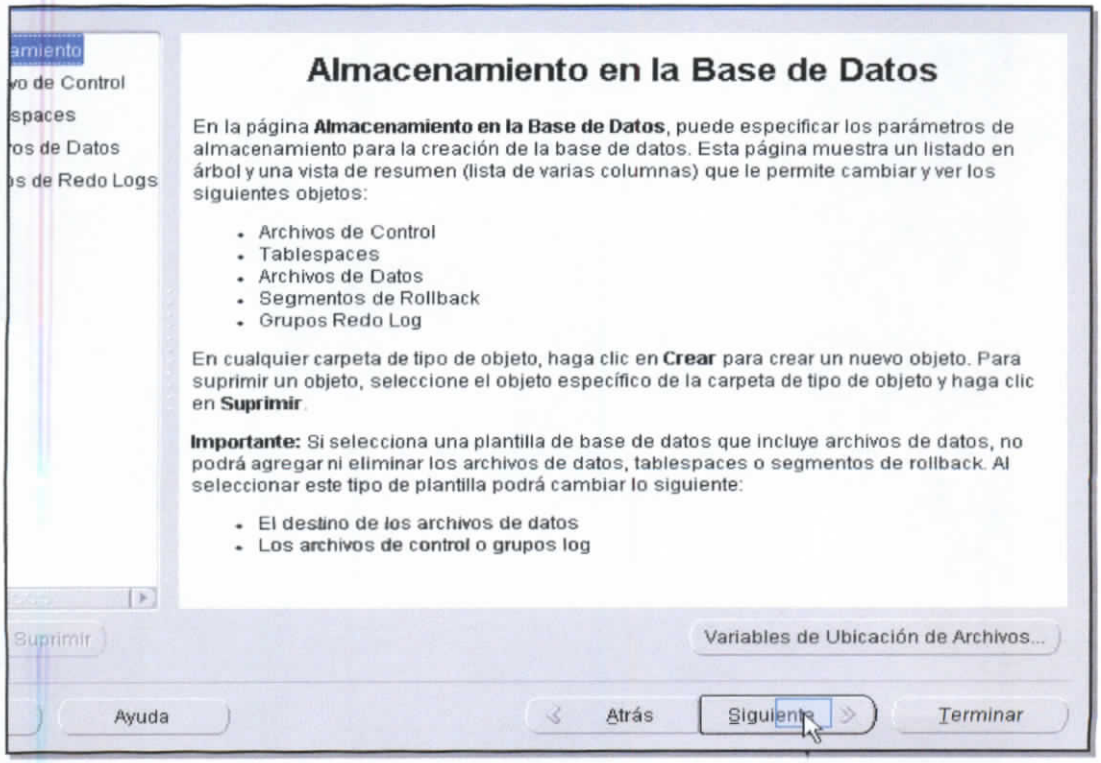
Se especifica contraseñas para usuarios (misma contraseña).



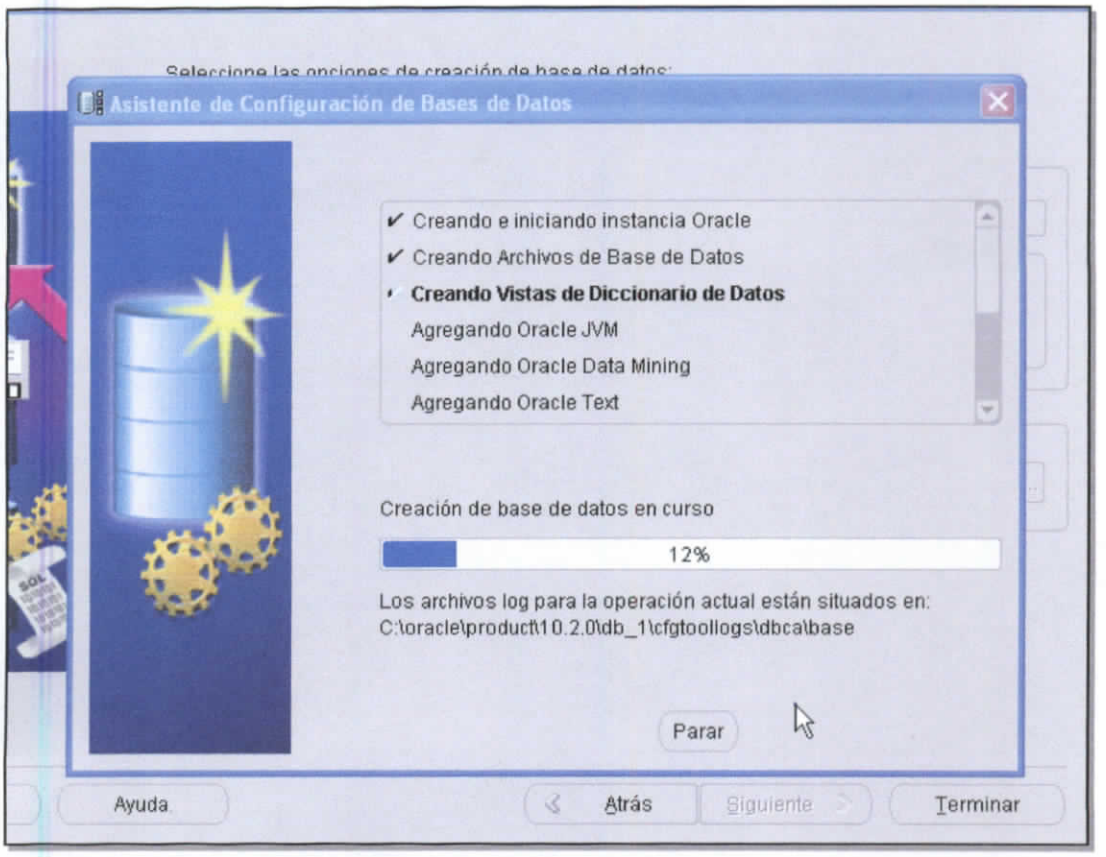
Se configura la base con los valores por defecto.



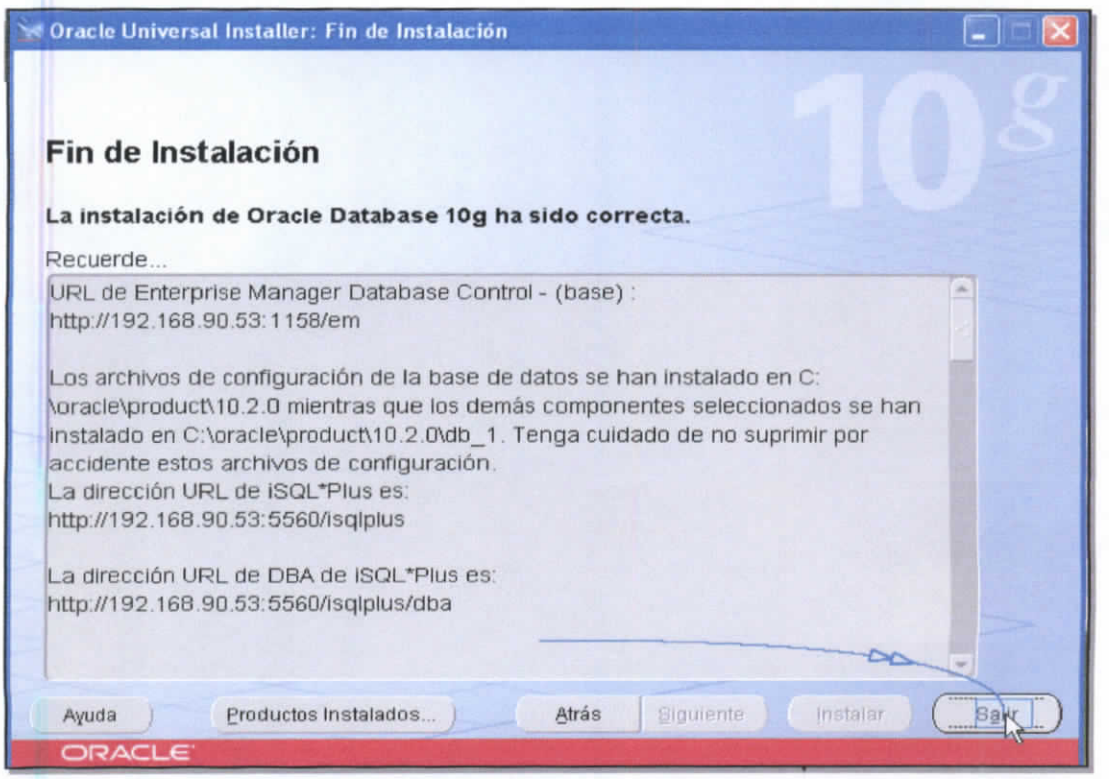
Se confirma la creación de la base.



Se inicia la creación de la base de datos.



Se finaliza con la instalación.



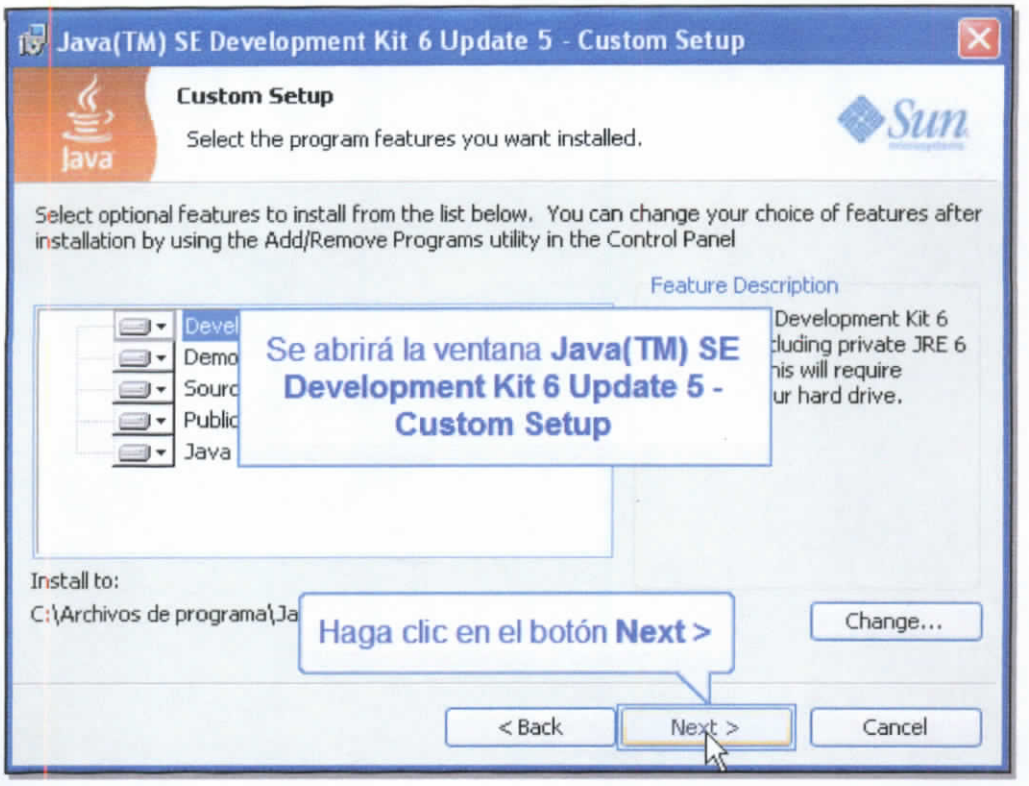
2. Instalación de Java

En esta fase se realiza la instalación de las máquinas virtuales de Java y del SDK necesario como ambiente de ejecución del sistema de evaluación del personal administrativo de la PUCESA. A continuación se detallan los pasos necesarios para esta instalación:

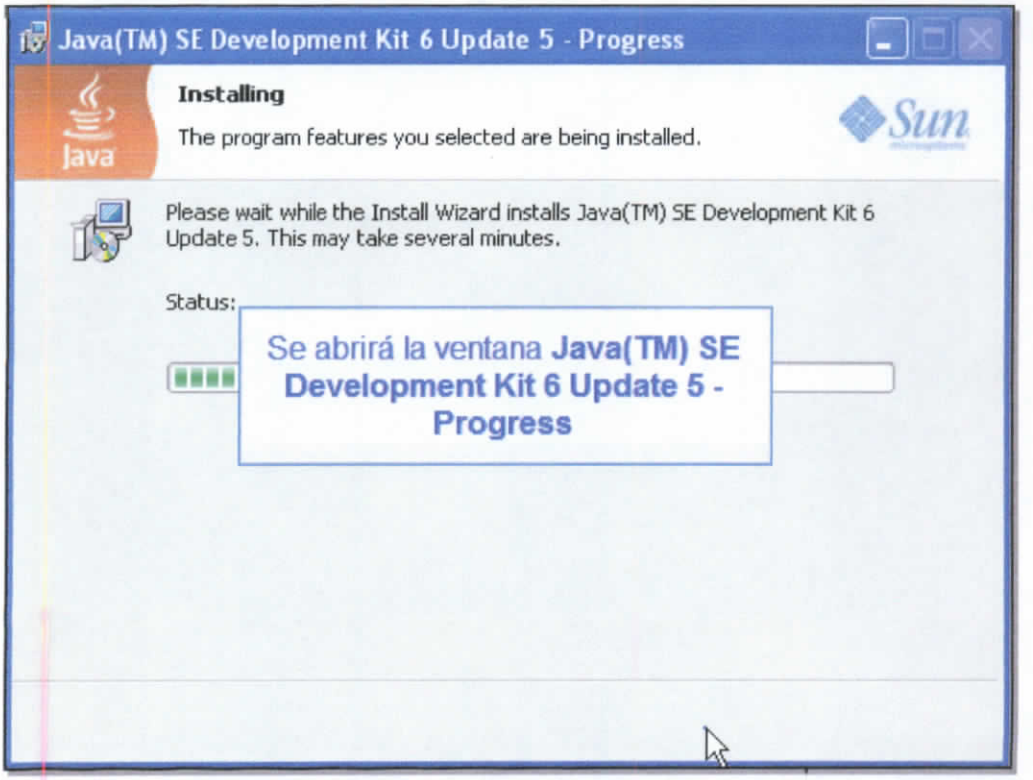
Se ejecuta el instalador contenido en el CD (jdk-6u5-windows-i586-p.exe).



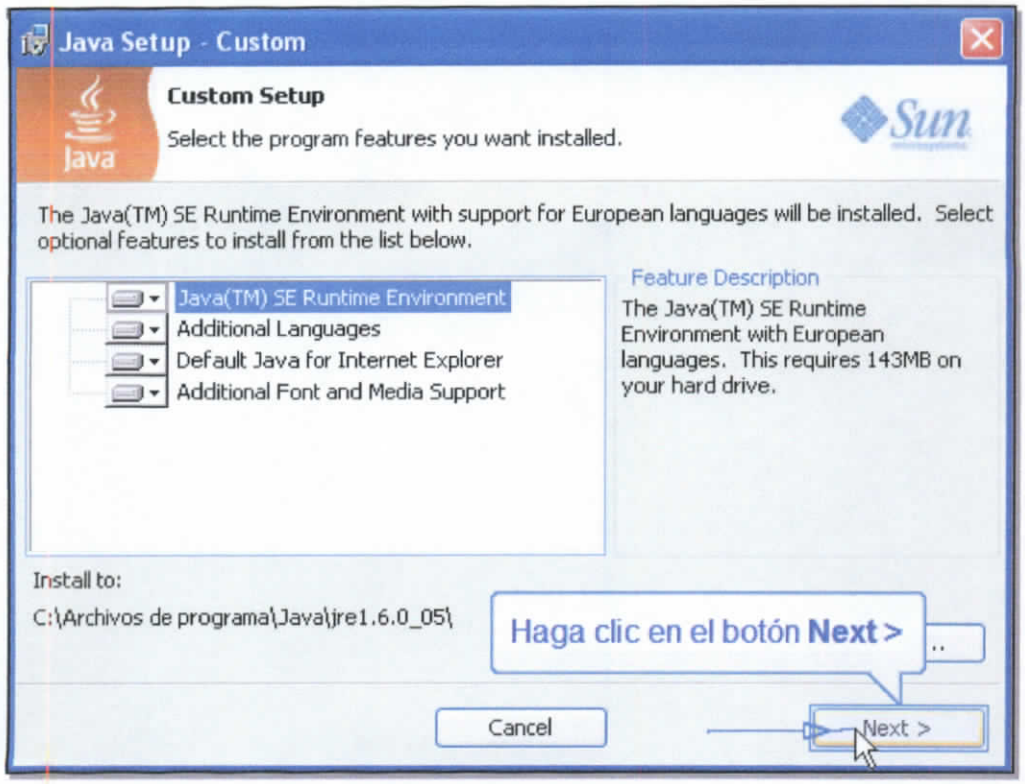
Se confirman las herramientas que se instalan por defecto.



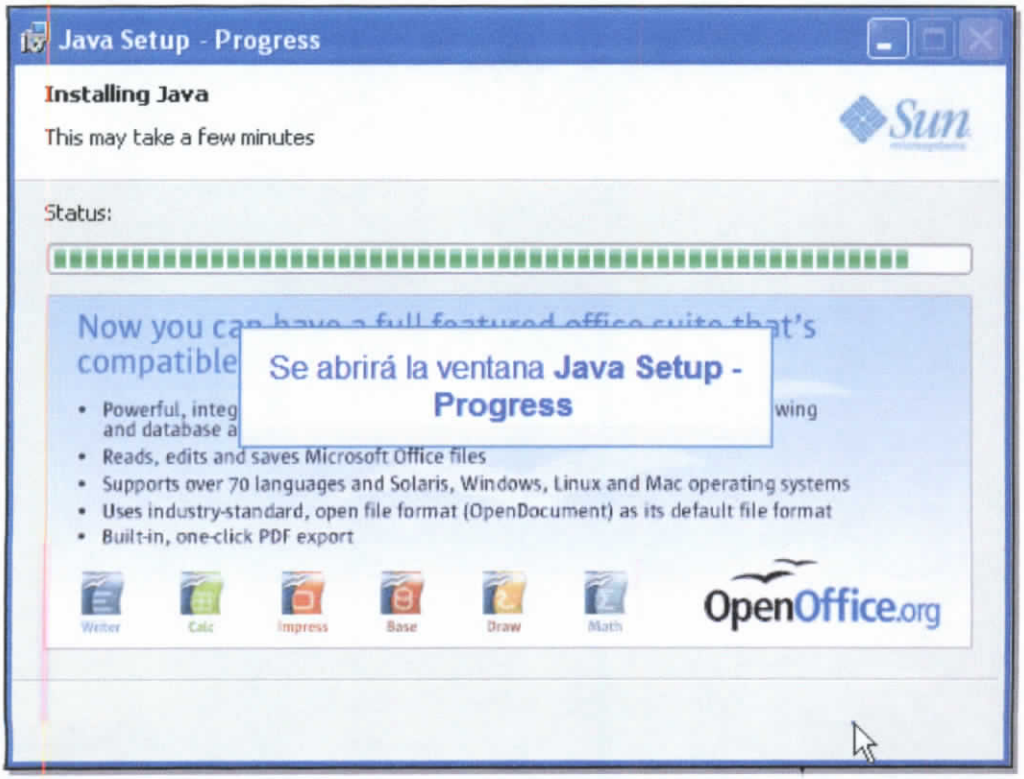
Inicia la instalación.



Se confirmas las herramientas por defecto.



Continúa con el proceso de instalación.



El proceso de instalación es finalizado exitosamente.



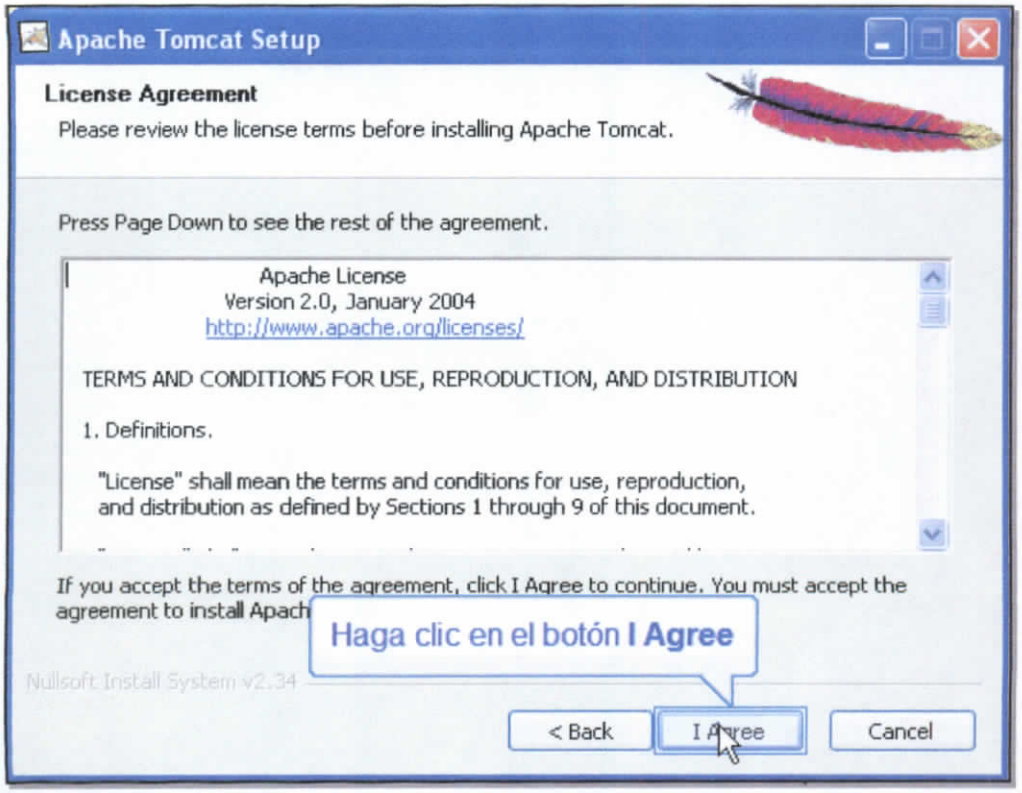
3. Instalación de Tomcat

El sistema de evaluación del personal administrativo de la PUCESA, al tratarse de una aplicación Web, requiere de un servidor de aplicaciones que realice el hosting de la misma, para este efecto se utiliza Tomcat con la versión 6.0. A continuación se detallan los pasos de instalación.

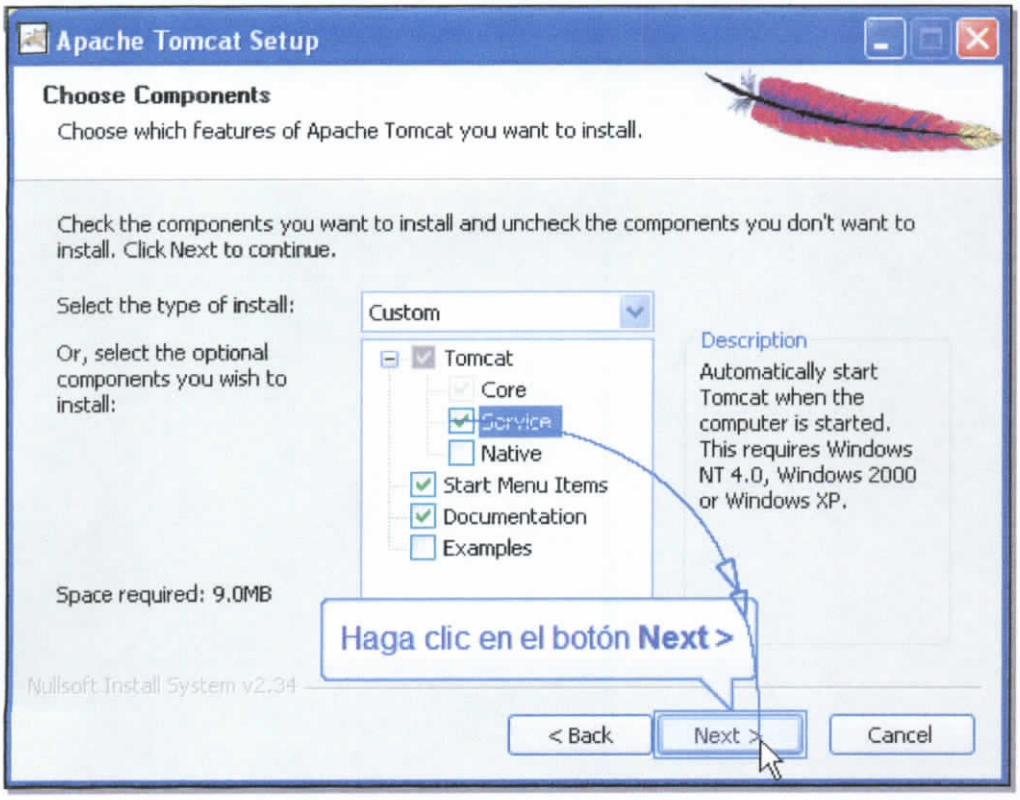
Se ejecuta el instalador contenido en el CD (apache-tomcat-6.0.16).



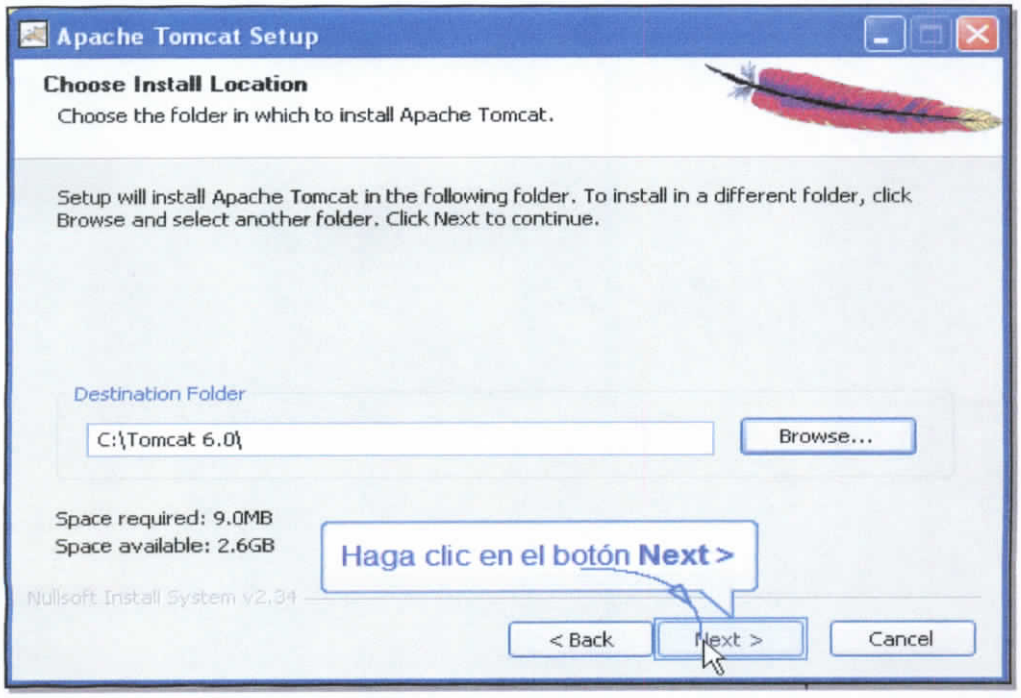
Se acepta los términos de instalación.



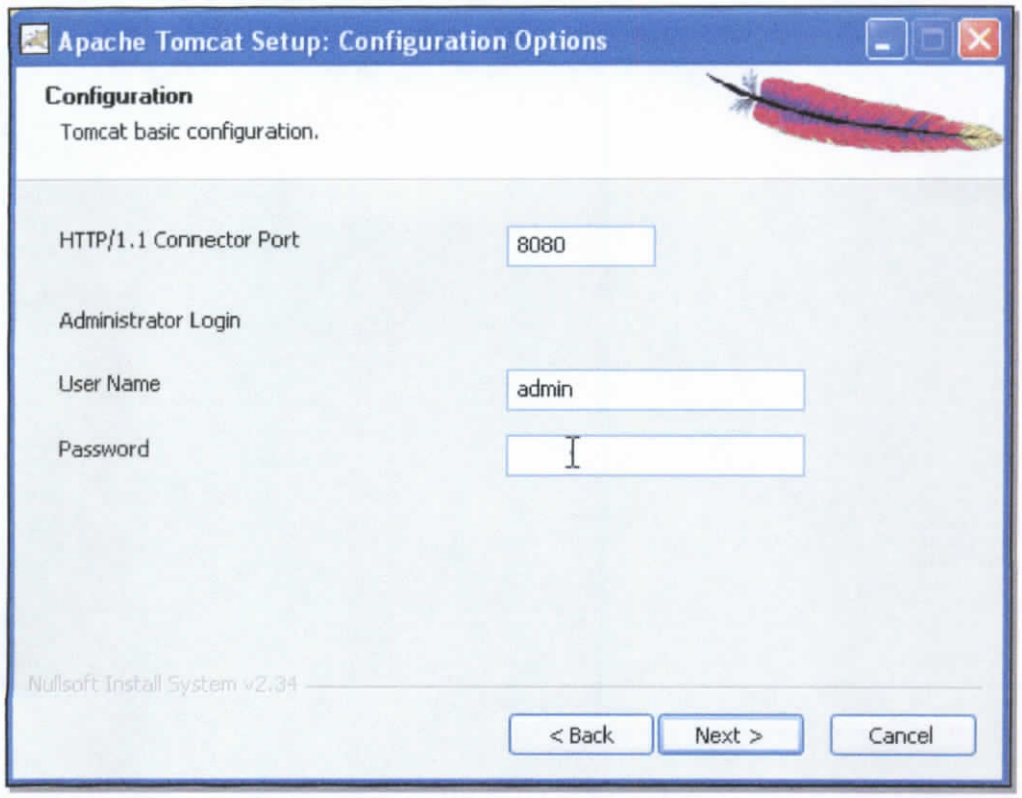
De los componentes por defecto de instalación es necesario añadir la opción de servicio para que el Tomcat se pueda levantar externamente.



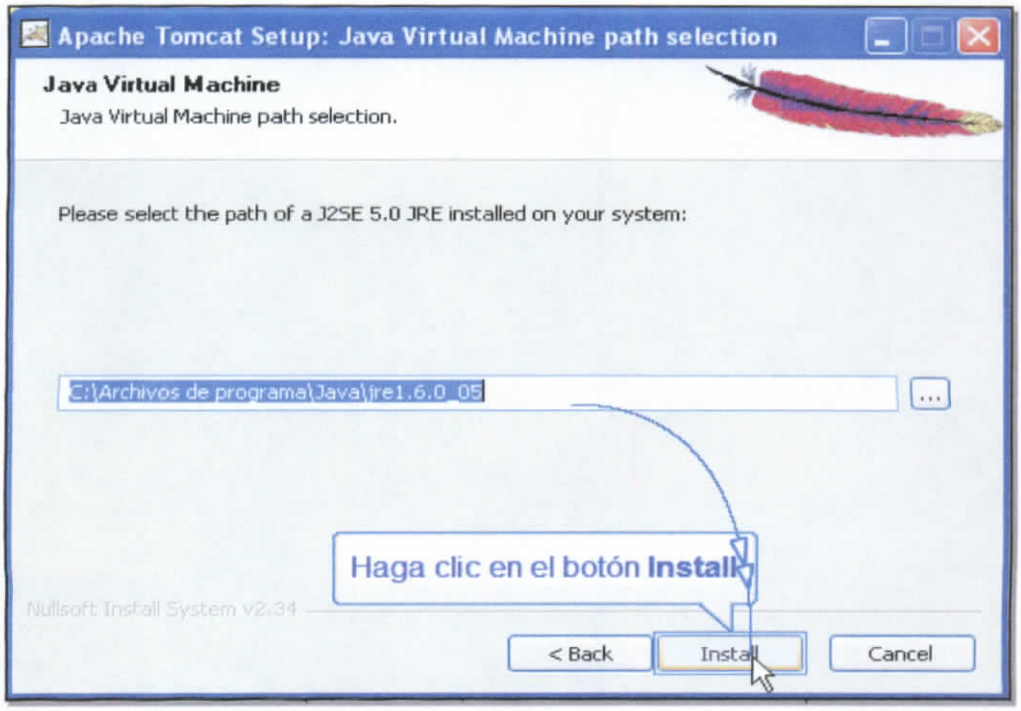
Se selecciona la ruta física de instalación, que para fines de este sistema se encuentra en la raíz del disco (D:\Tomcat 6.0).



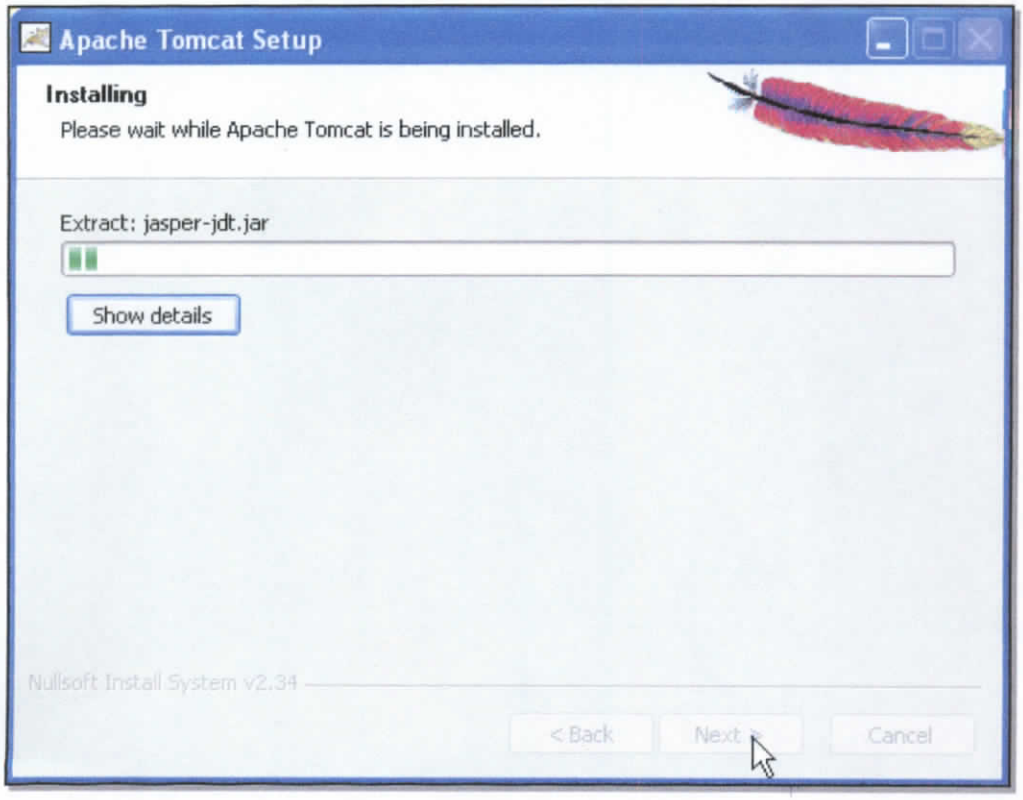
Se emplea el puerto 8080 para la instalación y se emplea una contraseña para la cuenta de administración del Tomcat.



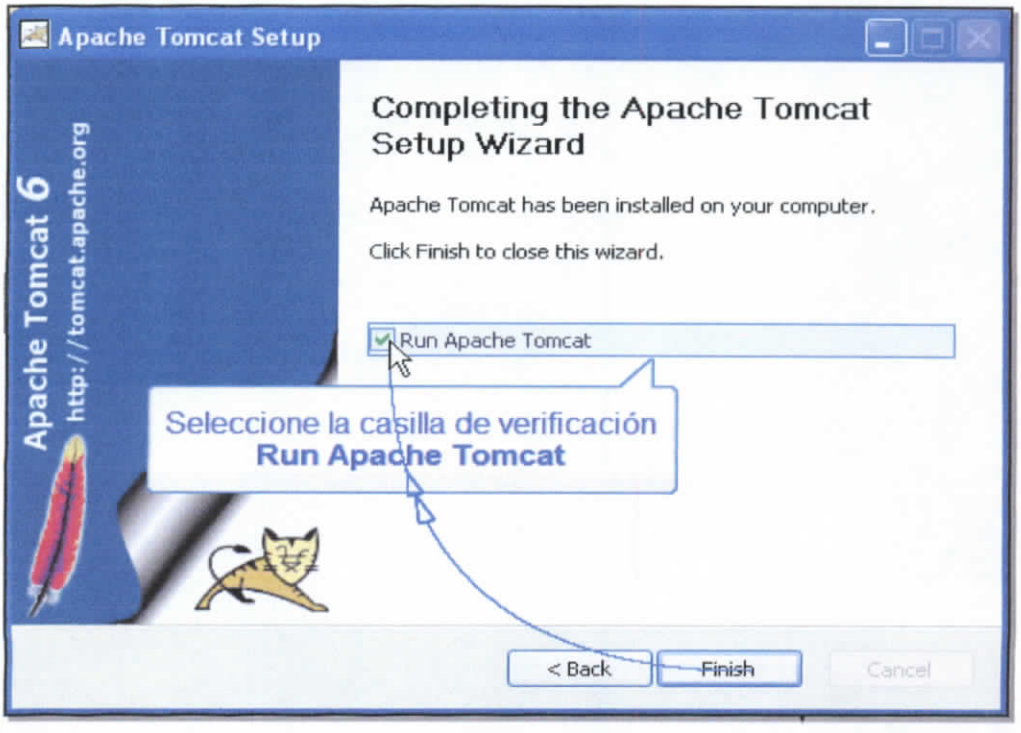
Posteriormente se selecciona la máquina virtual de Java que se instalo previamente.



El proceso de instalación se inicia.



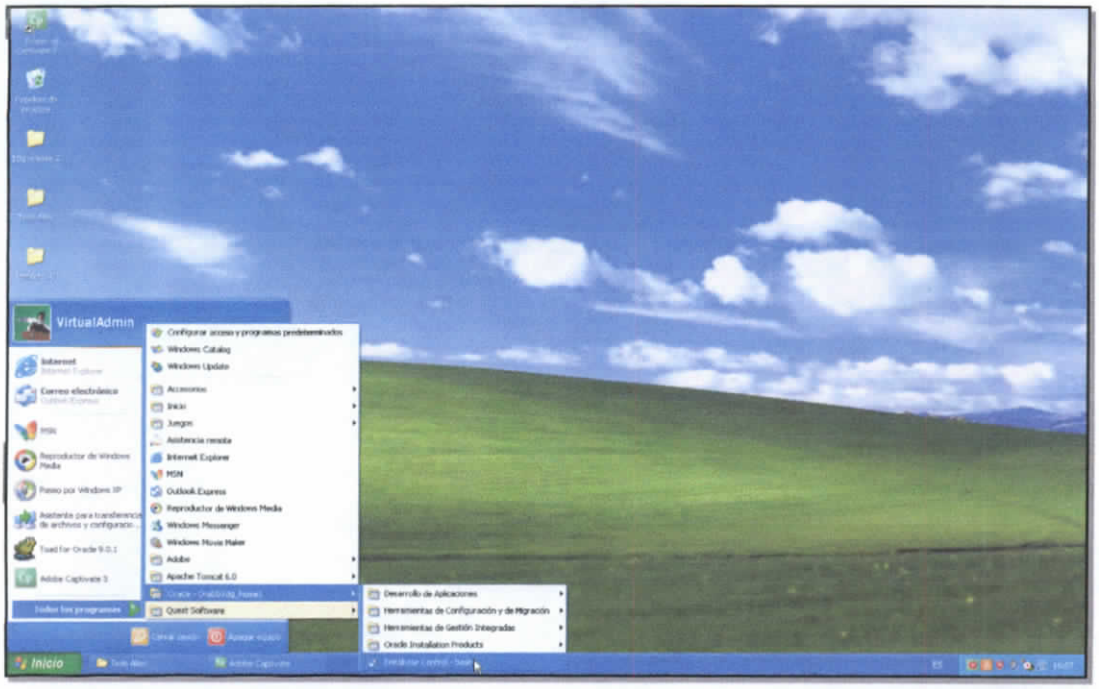
Posteriormente, el proceso de instalación es finalizado exitosamente, el servicio del Tomcat es levantado externamente.



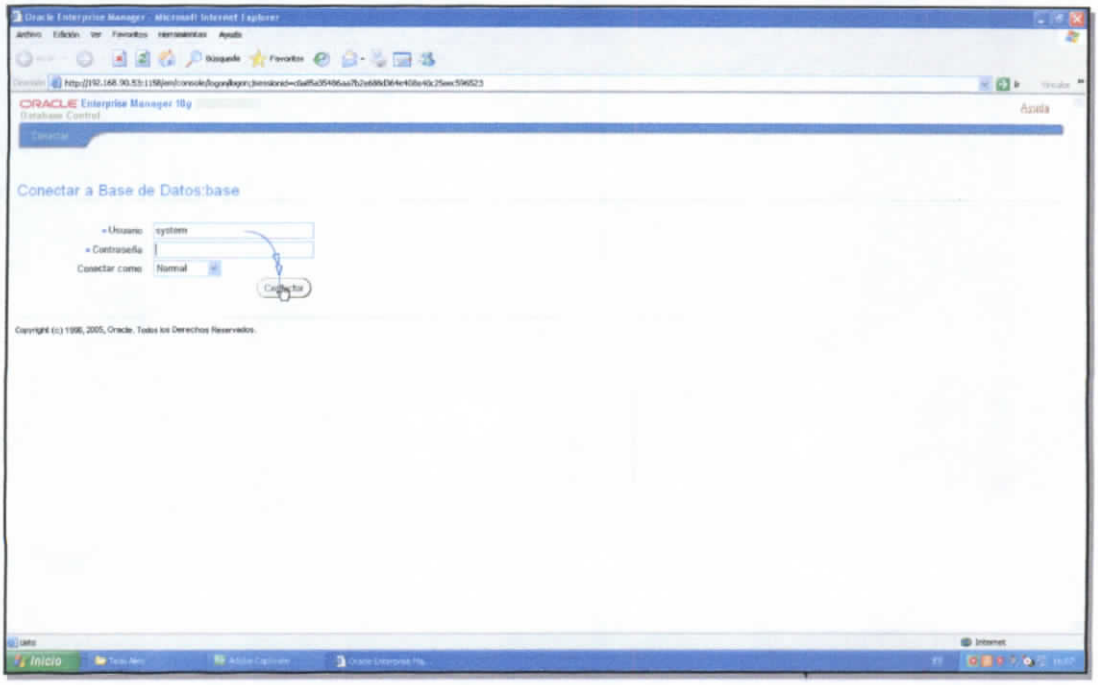
4. Creación de esquema de base de datos

Es necesario crear el esquema de base de datos que contendrá las tablas y estructura de datos del sistema de evaluación del personal administrativo de la PUCESA. A continuación se detallan los pasos de configuración:

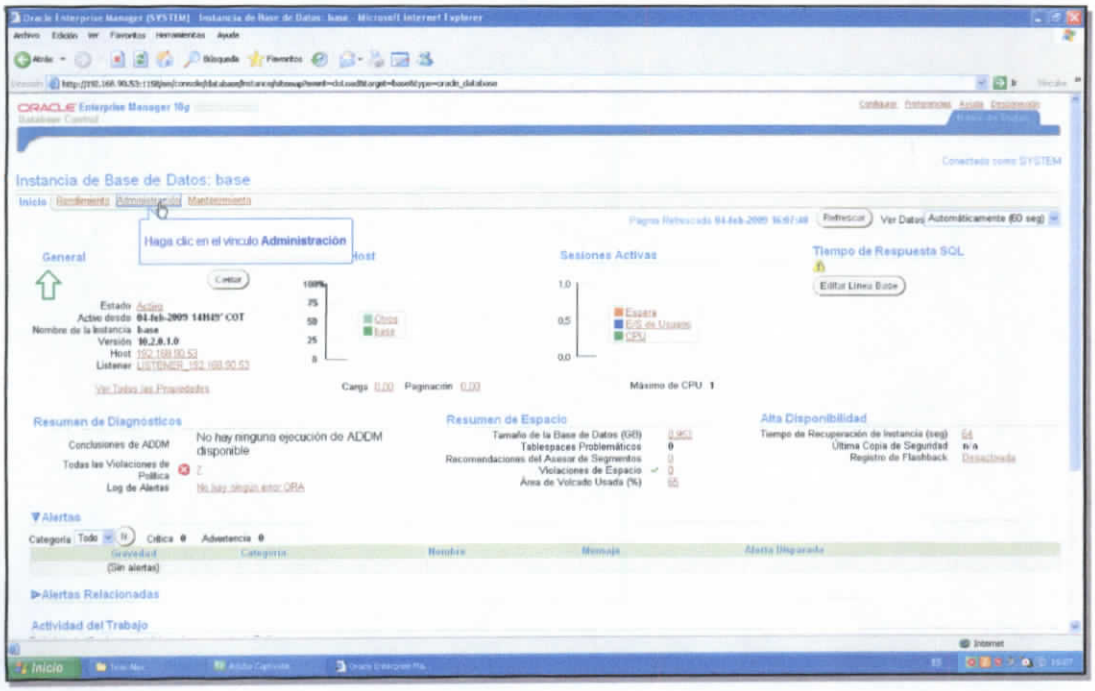
Se procede abrir la consola de administración de Oracle.



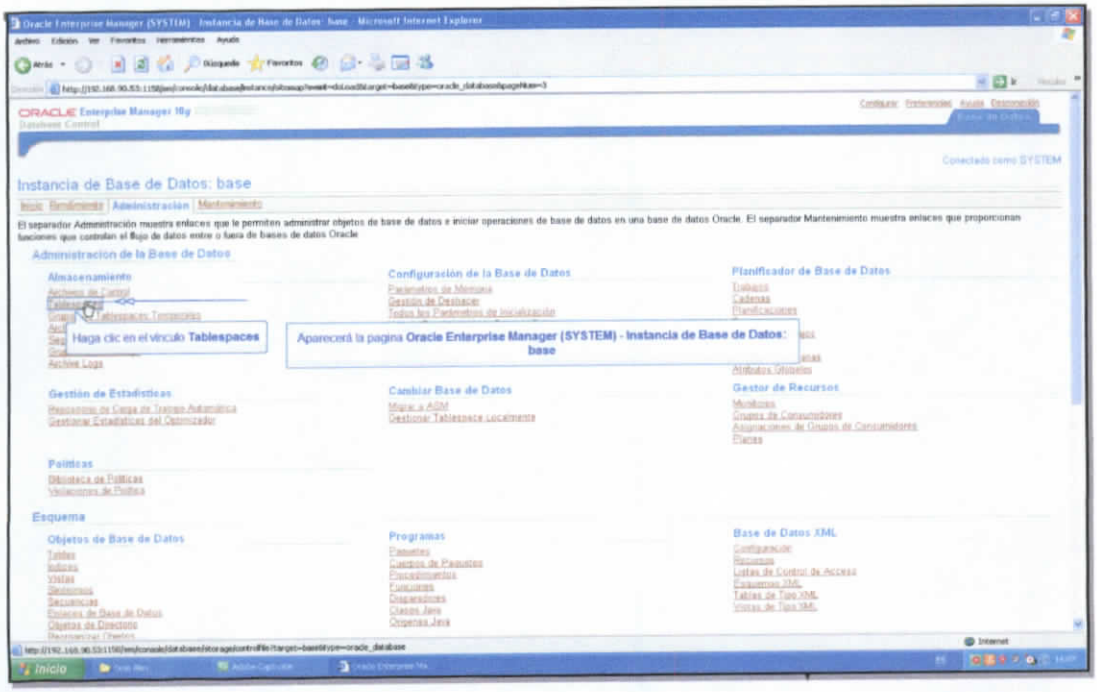
Se ingresa con la cuenta de system.



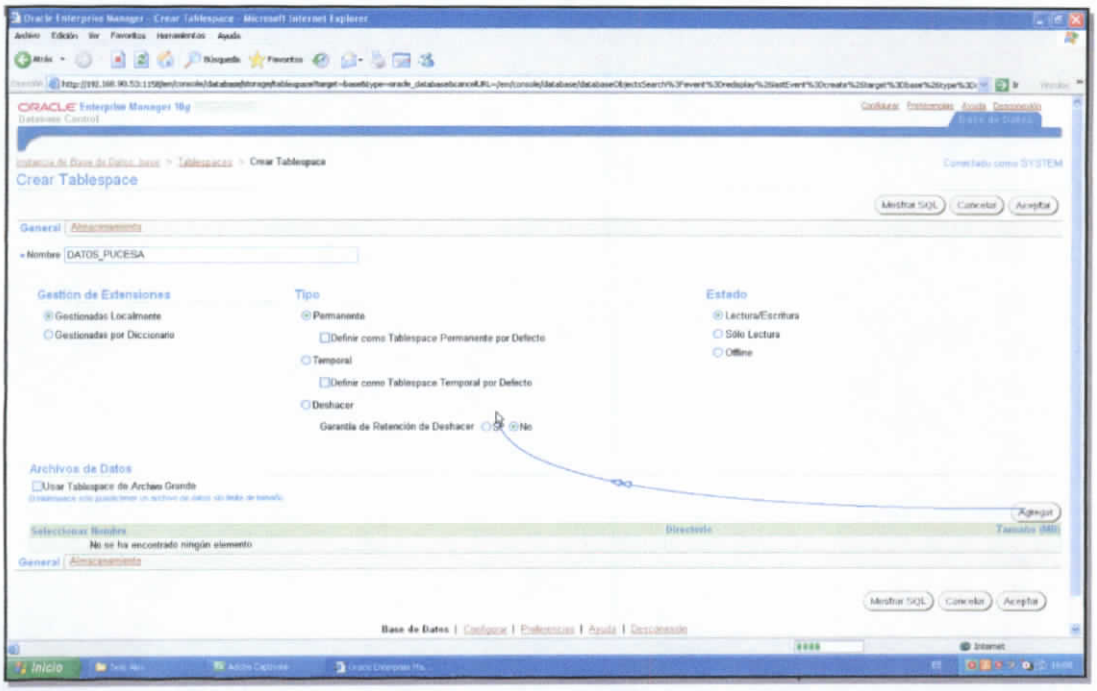
Es necesario crear los espacios de tabla, para ese efecto se debe dirigir al módulo de administración.



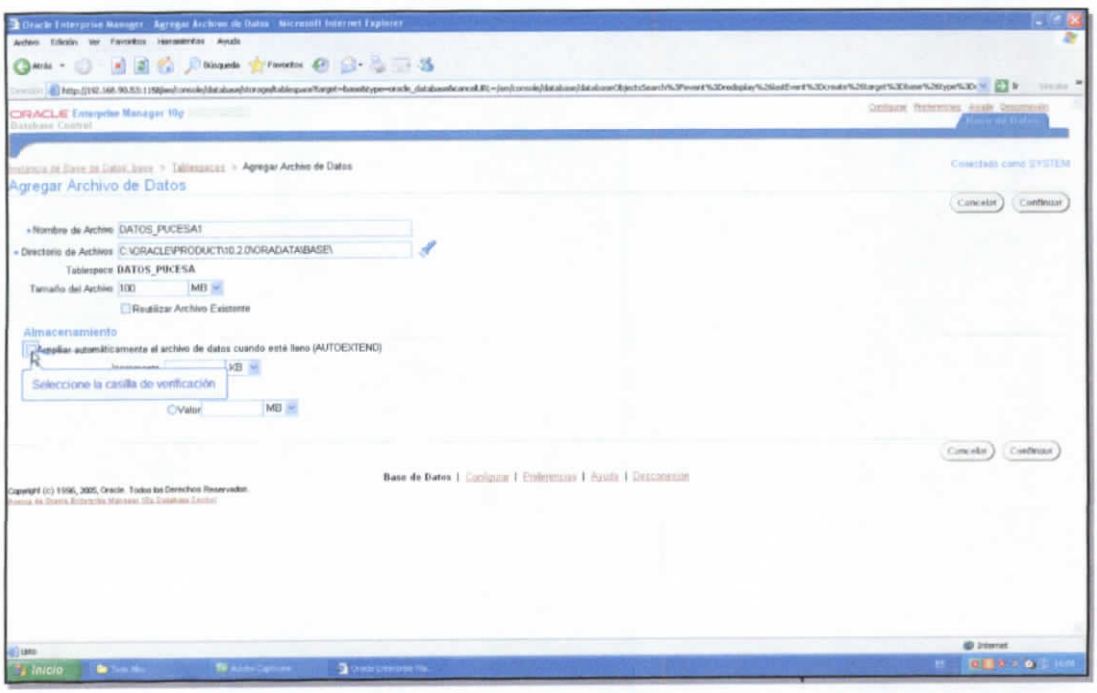
Se selecciona el vínculo de espacios de tabla.



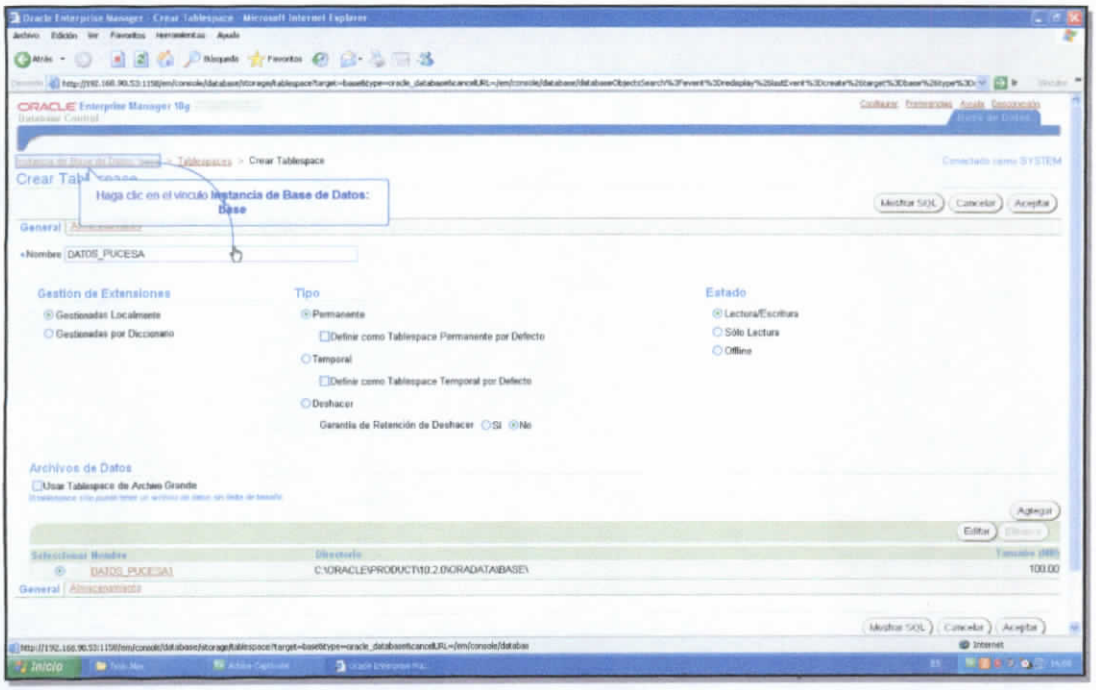
Se procede con la creación del espacio de tabla denominado (DATOS_PUCESA).



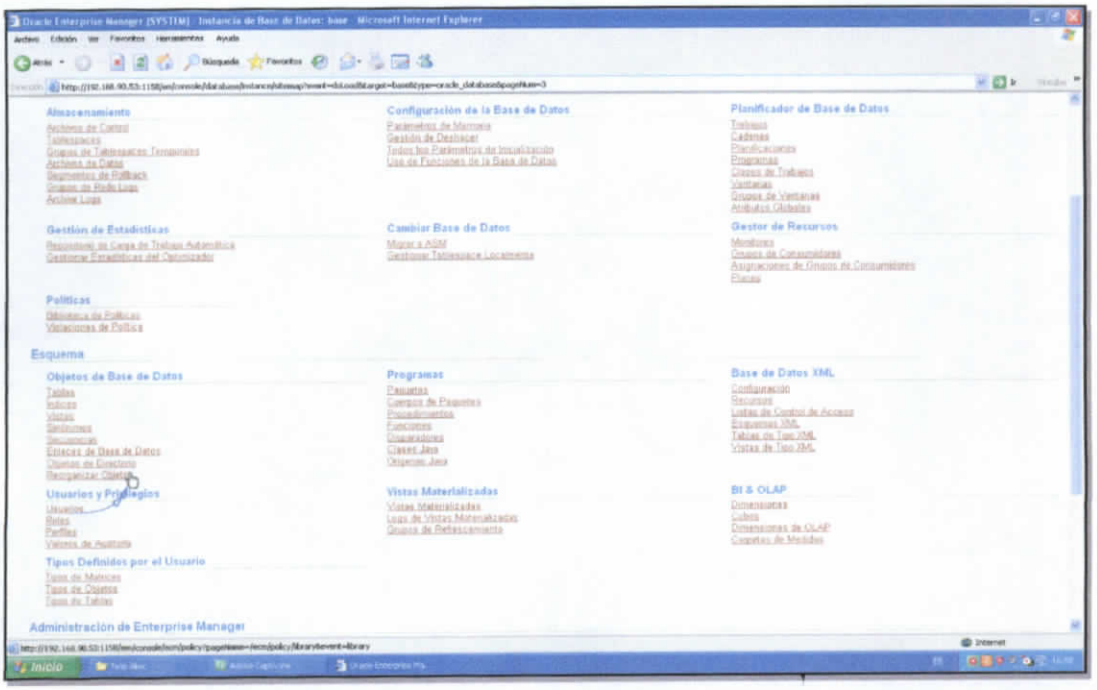
Se configura para que el espacio de tablas tenga un archivo de almacenamiento que se incremente automáticamente.



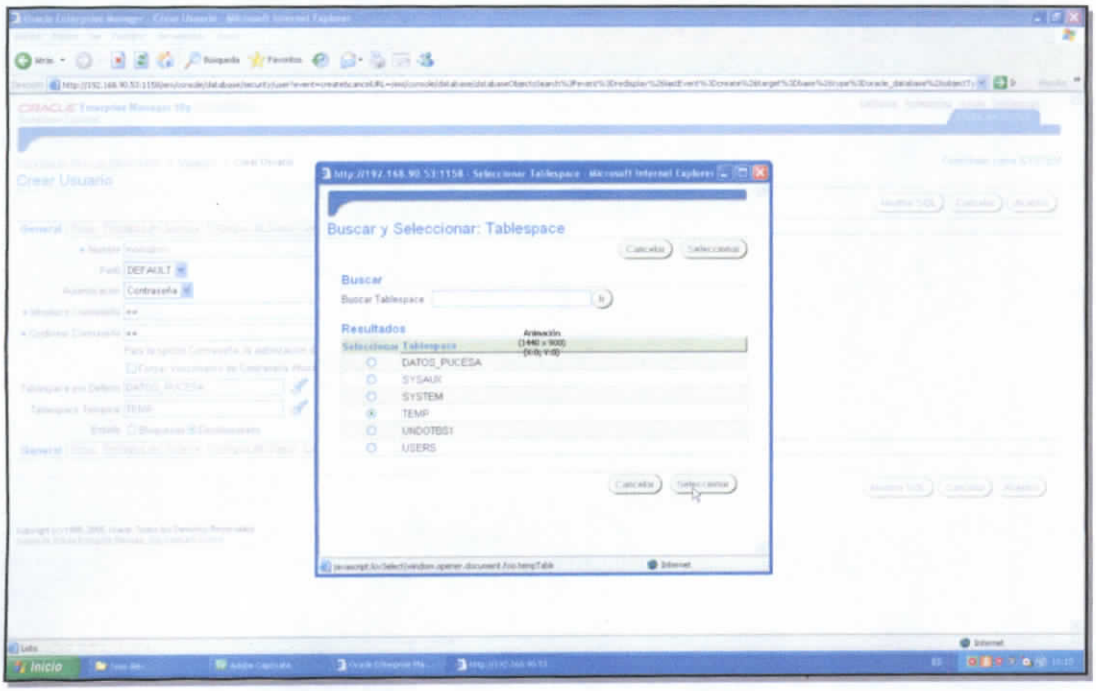
Se regresa a la página principal de la consola de administración.



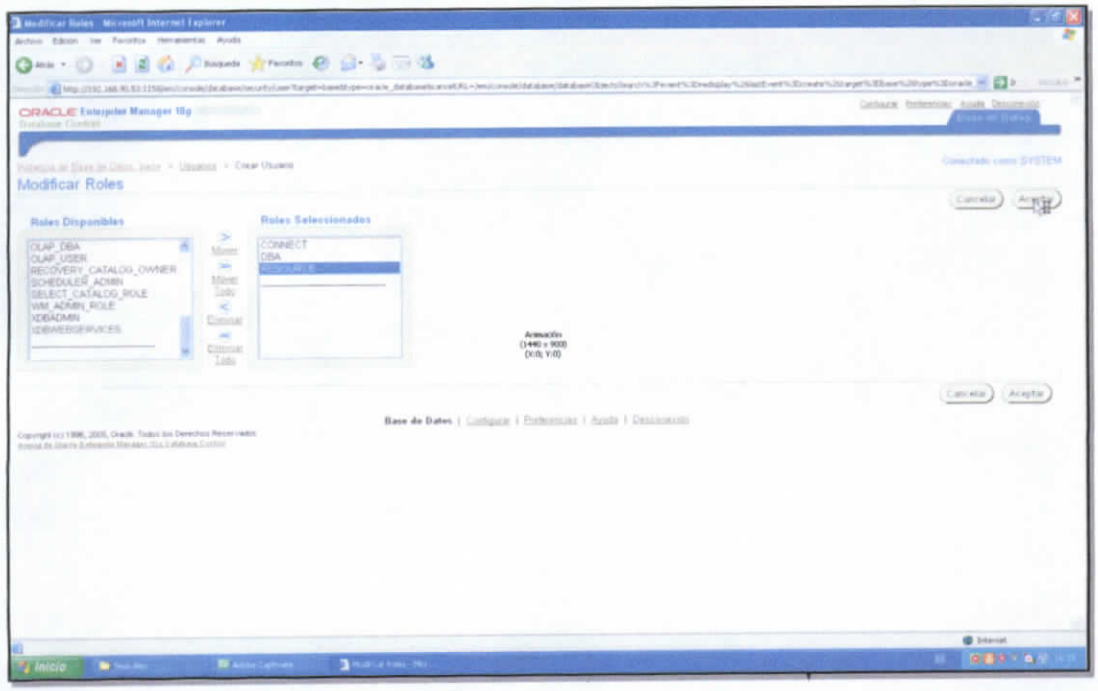
En la opción de administración se selecciona usuarios para la creación del usuario denominado evaluation que contendrá toda la estructura de datos.



Se procede a la creación de un nuevo usuario, es necesario seleccionar el espacio de tabla creado anteriormente (DATOS_PUCESA) como espacio de tabla por defecto. Se selecciona al espacio de tabla temporal (TEMP).

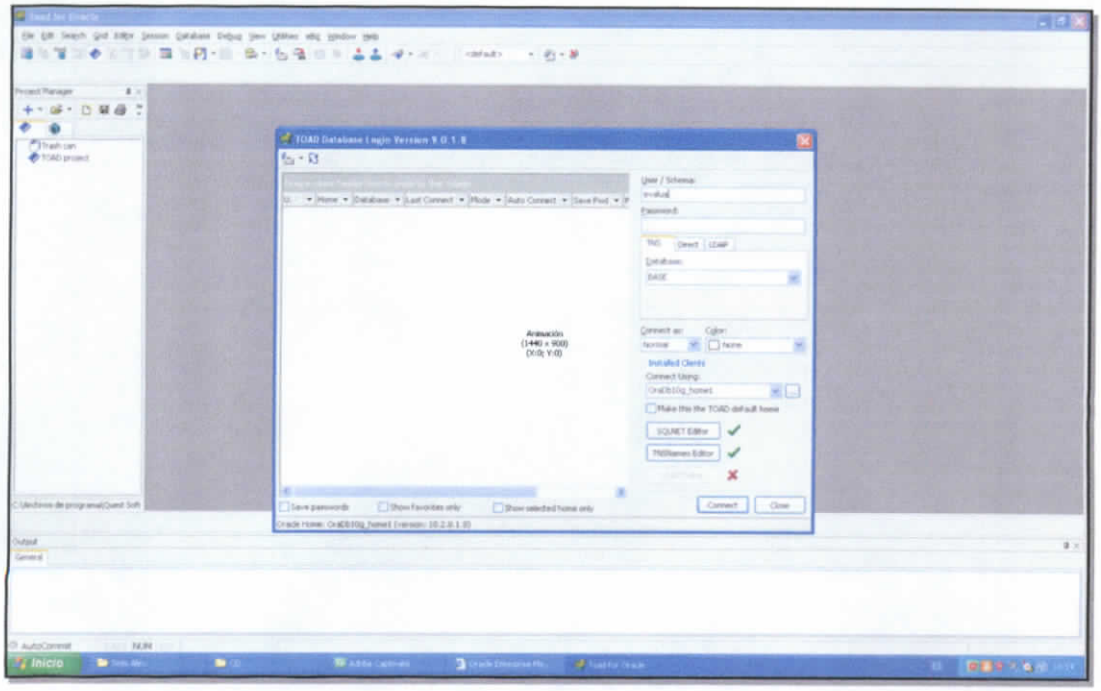


Es necesario editar los roles de servicio para añadir los permisos necesarios.

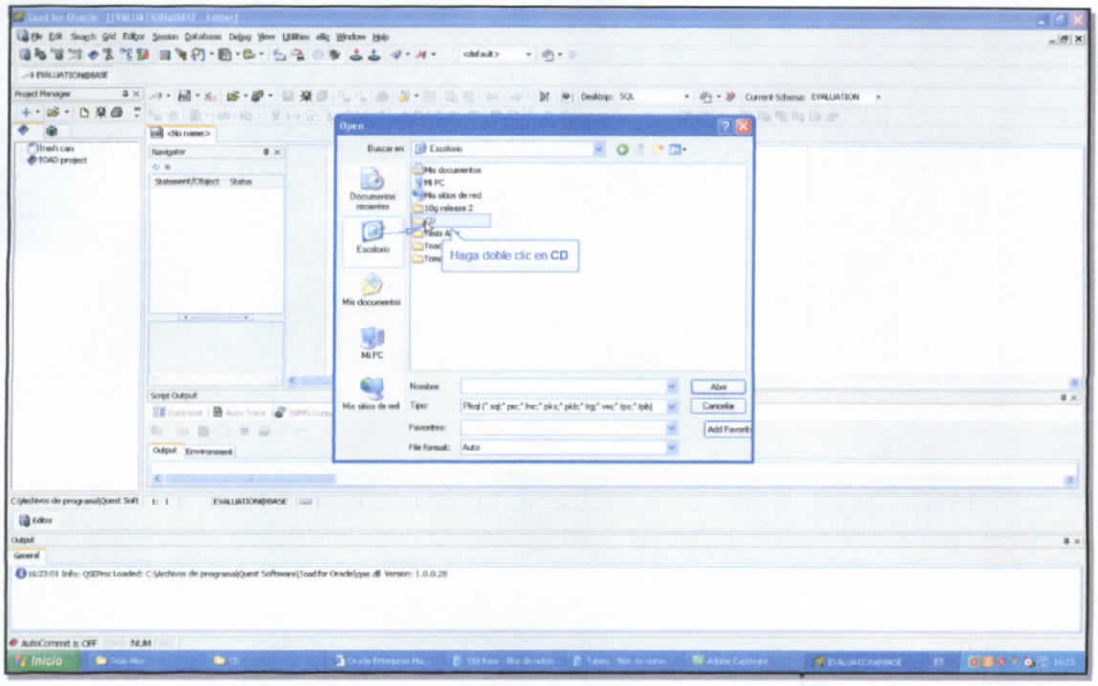


4.1 Ejecución del script de instalación

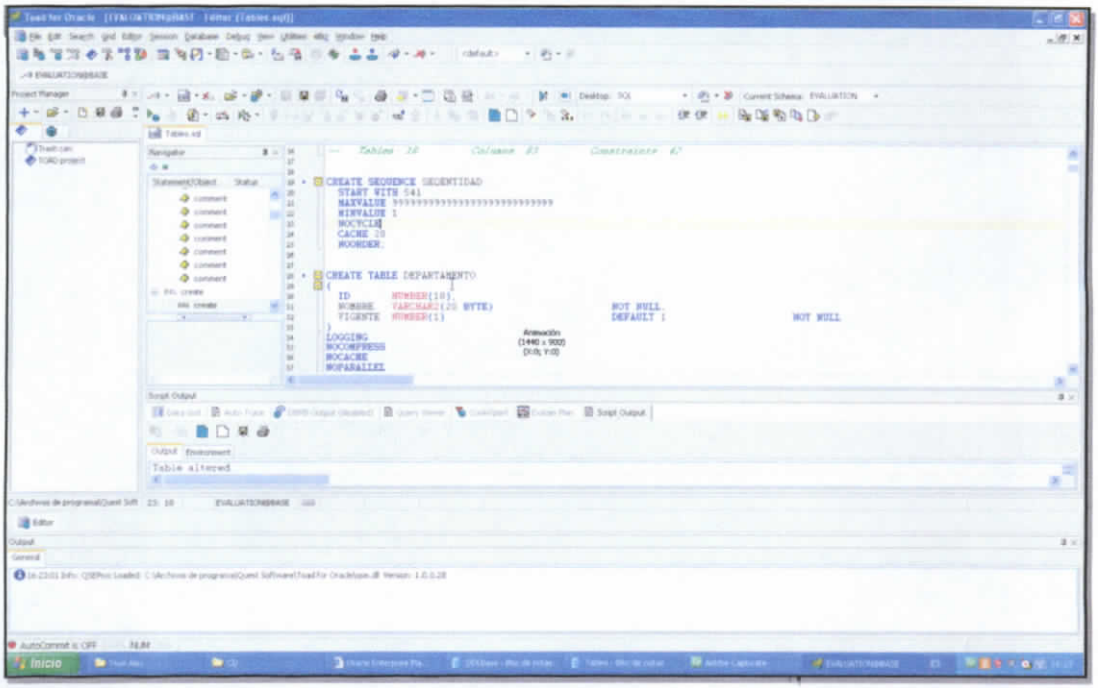
Se debe emplear una herramienta de ejecución de scripts, como por ejemplo: ISQL Plus, PL/SQL Developer para ejecutar el script contenido en el CD de instalación. Para este efecto se empleó la herramienta Toad.



Se realiza la apertura del script de instalación de la estructura (Tables.sql).



Se procede a la ejecución del script.



5. Publicación de la aplicación

Como último paso es necesario publicar la aplicación en el servidor de aplicaciones. En el CD de instalación se encuentra un archivo WAR que requiere ser deployado en el Tomcat. A continuación se detallan los pasos:

Se copia el archivo evaluation.war del CD de instalación.

