PROPUESTA DE UN PROCESO DE DESARROLLO DE COMPONENTES SOFTWARE REUTILIZABLES

ING. FREDY HUMBERTO VERA RIVERA

MAESTRÍA EN INGENIERÍA ÁREA EN INFORMÁTICA Y CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
UNIVERSIDAD INDUSTRIAL DE SANTANDER
BUCARAMANGA, 2009

TITULO
PROPUESTA DE UN PROCESO DE DESARROLLO DE COMPONENTES SOFTWARE REUTILIZABLES

PROYECTO DE INVESTIGACIÓN PARA OPTAR AL TÍTULO DE: MAGISTER EN INGENIERÍA ÁREA INFORMÁTICA Y CIENCIAS DE LA COMPUTACIÓN.

AUTOR
ING. FREDY HUMBERTO VERA RIVERA

DIRECTOR
Mcc. FERNANDO ROJAS MORALES

CODIRECTOR
Esp. ENRIQUE TORRES LOPEZ

MAESTRÍA EN INGENIERÍA ÁREA EN INFORMÁTICA Y CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
UNIVERSIDAD INDUSTRIAL DE SANTANDER
BUCARAMANGA, 2008

A Dios mi guía, por darme la oportunidad de cumplir este sueño,
a quien ofrezco este trabajo, todas las dificultades y logros
obtenidos a lo largo de mi formación profesional.

A mi Esposa Maribel y mi hijo Diego Alejandro,
fuentes de mi inspiración, por su gran apoyo,
sacrificio y esfuerzos realizados para
ayudarme a cumplir esta meta.

A mis Padres Humberto y Gladys, gracias a su sacrificio,
a su trabajo, a su esfuerzo, a sus enseñanzas y
apoyo incondicional.

A mi hermano José Luis, mis hermanas Diana Andrea y Luz Adriana,
por estar a mi lado y de alguna manera siempre
aportaron algo a lo largo de este proceso.

Fredy Humberto Vera Rivera

TABLA DE CONTENIDO

LISTADO DE TABLAS

LISTADO DE FIGURAS

RESUMEN

TITULO: PROPUESTA DE UN PROCESO DE DESARROLLO DE COMPONENTES SOFTWARE REUTILIZABLES[*].

AUTORES:
Fredy Humberto Vera Rivera[**].

PALABRAS CLAVES:
Ingeniería del Software Basada en Componentes, Componentes Software Reutilizables, Modelo de Componentes, Enterprise Java Bean, Servicios Web.

DESCRIPCIÓN:

El presente trabajo corresponde a la investigación para obtener el título de magister titulada: "Propuesta de un proceso de desarrollo de componentes software reutilizables", mediante la cual se busca establecer los pasos necesarios para crear componentes software reutilizables en Java Edición Empresarial 5 (Java EE 5). En primer lugar se hace el marco teórico y la descripción de la investigación planteando la problemática que se evidencia en el desarrollo de software empresarial y cómo la Ingeniería del Software Basada en Componentes (ISBC) puede ayudar a resolverla; se aclara la definición de componente y se plantean las preguntas de investigación. Después se plantea la estructura de un componente software reutilizable siguiendo el modelo de componentes de Java, el cual consta principalmente de Entidades (pojos, antiguos EJB de entidad), EJBs (de sesión o manejador de mensajes), componentes o controles personalizados para la interfaz de usuario y servicios web que exponen las funcionalidades encapsuladas en los EJBs como servicios web. Después se propone el proceso de desarrollo de componentes reutilizables el cual consta principalmente de las siguientes fases:

1. Análisis de requisitos del dominio de aplicación.
2. Modelo de selección de componentes software reutilizables.
3. Definición de la arquitectura.
4. Procedimiento de desarrollo, pruebas y especificación de componentes.
5. Esquemas de distribución del componente.

Por último se detalla la elaboración de un componente software reutilizable para la División de Servicios de información siguiendo el proceso de desarrollo planteado.

---

[*] Proyecto de Investigación.

** Facultad de Ingenierías Físico-Mecánicas, Maestría en Ingeniería área Informática y Ciencias de la Computación.
Director: Mcc.Fernando Rojas Morales. Profesor Escuela de ingeniería de Sistemas - UIS.
Codirector: Esp. Enrique Torres Lopez, División de Servicios de Información – UIS.

SUMMARY

TITLE: ***PROPOSAL OF A PROCESS OF THE DEVELOPMENT OR REUSE SOFTWARE COMPONENTS****.

AUTHORS:
Fredy Humberto Vera Rivera**.

KEY WORDS:
Component – Based Software Engineering, Reuse software component, Component Model, Enterprise Java Bean, Web Services.

DESCRIPTION:

This work corresponds to the research to obtain the master degree: "Proposal of a process of the development or reuse software components" by which it is wanted to set the necessary steps to create reuse software components in Java Enterprise Edition 5 (Java EE 5). First of all, an introduction is made to set out the problem that is evident in the development of enterprise software and how the Component Based Software Engineering (CBSE) can help to solve it; the definition of component is clarified and the research questions are set. Later it is explained the methodology used in the research that comprises the descriptive research and applied technologic research. After, it is set up the structure of a reuse software component following the component model of Java, which consist mainly of Entities (pojos, old EJB entities), EJB (session beans, message driver bean), components or personalized controllers for the user interface and web services that present the covered operations in the EJBs like web services. Later, it is established a process of development or reuse software components, the process cover mainly the next phases:

1. Analysis of domain application requirements.
2. Model of selection of reuse software components.
3. Definition of the architecture.
4. Procedure of development, test and specification of the component.
5. Deploy the reuse software component.

Finally, the book detail the elaboration of reuse software component for the Division of Information Services follow the process described previously.

---

* Term paper: Mode of management practice

** Faculty of Physical-Mechanical Engineerings, Master in Engineering of System and Computer Science. Headmistresss: Mcc. Fernando Rojas Morales, Codirector: Esp. Enrique Torres Lopez, Division of Information Services.

0. INTRODUCCIÓN

Los principios de reutilización de software han estado presentes a lo largo de muchos años y es sueño de todo ingeniero de software, poder tener un conjunto de unidades software o aplicaciones modulares que se puedan ensamblar para formar un sistema nuevo más grande y complejo, en vez de tener que desarrollarlo desde cero. A estas unidades software se les conoce como componentes. Al poder utilizar estos componentes software, que ya han sido probados y verificados se puede disminuir el tiempo de desarrollo y hacer sistemas informáticos más confiables y seguros. Cuando se requiera construir un nuevo sistema, se realizaría buscando los componentes en el repositorio, se combinarían y adaptarían estos componentes, en vez de estar codificando y construyendo las mismas aplicaciones cada vez.

La ingeniería del software basada en componentes (ISBC) es un proceso que se centra en el diseño y construcción de sistemas basados en computadora que utilizan componentes de software reutilizables. La ISBC lucha por conseguir un conjunto de componentes de software preconstruidos y estandarizados que estén disponibles para encajar en un estilo arquitectónico específico para algún dominio de aplicación. La aplicación se ensambla entonces utilizando estos componentes y no las piezas por separado de un lenguaje de programación convencional[2].

Actualmente existen portales Web donde venden diferentes clases de componentes para diferentes plataformas, algunos de estos portales son: Componentsource (www.componentsource.com), Flashline (www.flashline.com) y WrldComp (www.wrldcomp.com), también encontramos marcos de trabajo (frameworks) que permiten utilizar componentes ya implementados para desarrollar nuevas aplicaciones. En cuanto a estándares para el desarrollo y comunicación de componentes existen varios modelos de componentes como son: .NET, COM (Component Object Model), DCOM (Distributed Component Object Model) de Microsoft, JavaBeans y EJB (Enterprise Java

[2] Roger S. Pressman. Ingeniería del Software un enfoque práctico. Quinta Edición. Mc. Graw Hill.. 2002. Página 473.

Beans) de Sun Microsystems y CORBA (Common Object Request Broker Architecture) del Object Management Group.

Las empresas desarrolladoras de software y la industria en general, deben ir aplicando la ISBC para poder crear su propia biblioteca de componentes para hacer más fácil, seguro y rápido el desarrollo de nuevos sistemas informáticos. El ensamblaje de componentes lleva a una reducción en el tiempo de desarrollo, una reducción en los costos del proyecto y un aumento en el índice de productividad. Aunque estos resultados están en función de la robustez de la biblioteca de componentes, no hay duda que el ensamblaje de componentes proporciona ventajas significativas para los ingenieros de software y las empresas en general.

1. DESCRIPCIÓN DE LA INVESTIGACIÓN

1.1. MARCO DE REFERENCIA

1.1.1. MARCO CONCEPTUAL

El marco conceptual tiene como finalidad unificar los significados de conceptos y siglas empleados dentro del plan de investigación para efectos de claridad en el lenguaje utilizado. De esta manera, se evitan interpretaciones diferentes de las palabras usadas en este documento.

1.1.2. TÉRMINOS UTILIZADOS

Commercial Of-The-Shelf (COTS): Componente software diseñado, elaborado y comercializado por terceras partes, generalmente de bajo precio o de libre distribución, utilizado por los desarrolladores de software como partes preexistentes que puede incorporar en su arquitectura sin crearlos.

Common Object Request Broker Architecture (CORBA): Es un modelo de componentes estandar diseñado para permitir la interoperabilidad entre componentes de diferentes plataformas, lenguajes, autores o vendedores. En el mercado existe una gran variedad de implementaciones, por ejemplo Orbix y Orbacus de IONA, ObjectBroker de Bea Systems, o VisiBroker de Visigenic/Borland.

Component Object Model (COM): Modelo de componentes de Microsoft. Define un estilo arquitectónico para componentes software de Microsoft.

Componente: Un componente es una unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio. (Szyperski, 1998)

Desarrollo de Software Basado en Componentes (DSBD): Una perspectiva del desarrollo de software en donde todos los “artefactos" se pueden construir mediante ensamblaje, adaptación e integración de componentes existentes y en una variedad de configuraciones.

Enterprise Java Beans (EJB): Es el modelo de componentes del lado servidor de Sun Microsystems.

Especificación: Un documento que prescribe de forma completa, precisa y fiable los requisitos, diseño, comportamiento o características de un sistema o de un componente.

eXtensible Markup Language (XML): Lenguaje de marcas extensible. Ampliamente usado para modelización de datos, intercambio de información, formato para el almacenamiento de información en repositorios, entre otras aplicaciones.

Modelo de componentes: Define la forma como se especifica un componente y la forma como se ensambla el componente, también incluye una serie de tipos de componente, sus interfaces y una especificación de los patrones aceptables de interacción entre tipos de componentes. Ejemplos de modelo de componentes: COM/DCOM, .Net, CORBA CCM, EJB, OSGi y Web Services[3].

Servicio web: Un servicio web en un sistema software identificado por un URI (identificador de recursos uniforme), cuyas interfaces públicas y enlaces están definidos y descritos en XML, y su definición puede ser localizada por otros sistemas de software que pueden luego interactuar con el servicio web en la manera preestablecida por su definición, utilizando mensajes basados en XML transmitidos por protocolos de Internet.

Unified Modeling Language (UML): Lenguaje de modelado de sistemas software desarrollado por OMG.

[3] Hyung Cho y John D. McGregor. IEEE – 2005 Artículo: Component Specification for Enterprise software Development on Web Services Environment

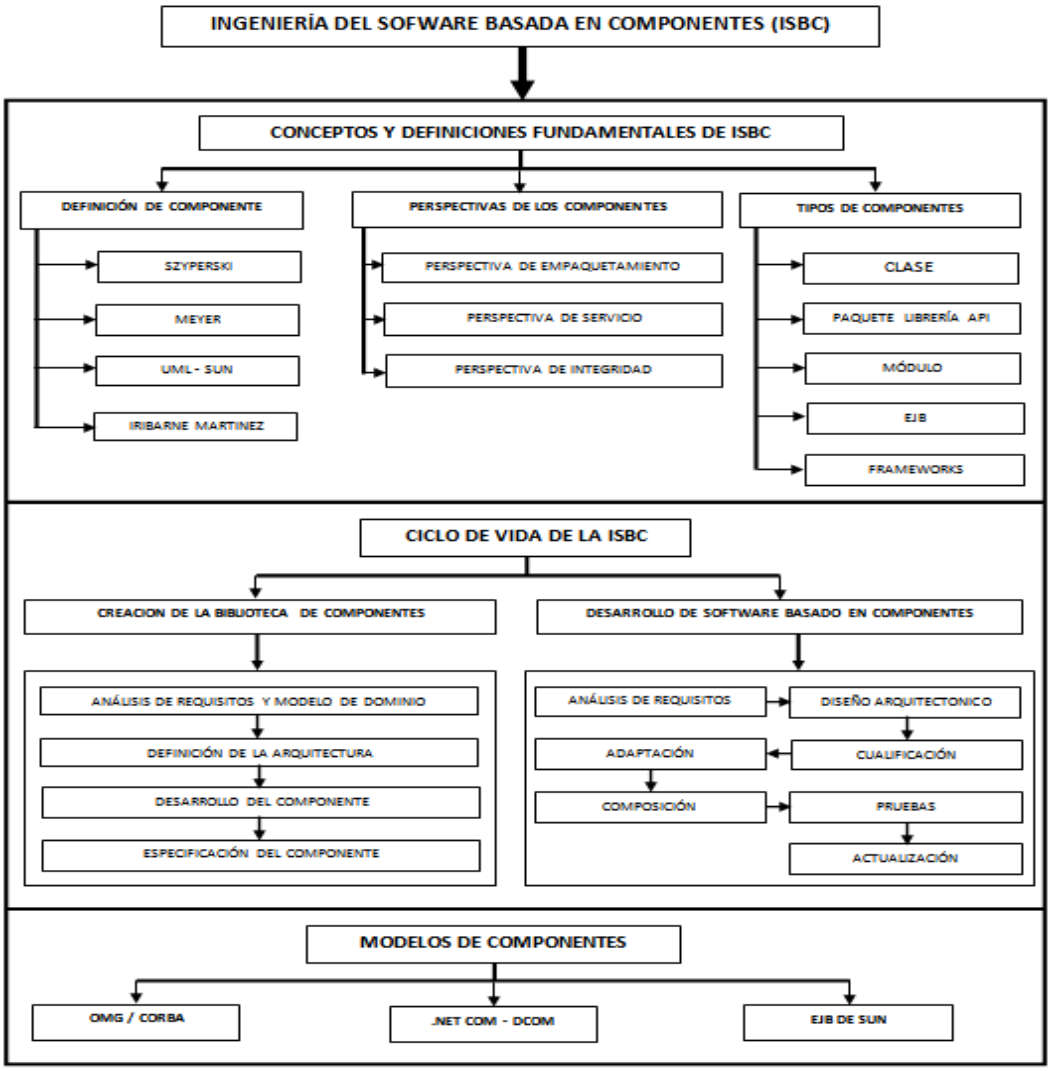
1.1.3. SIGLAS

API	Application Programming Interface
ASP	Active Server Page
CBD	Component-Based Development
CBSE	Component-Based Software Engineering
CCM	CORBA Component Model
CGI	Common Gateway Interface
COM	Common Object Model
CORBA	Common Object Request Broker Architecture
COTS	Commercial Of-The-Shelf
DSBC	Desarrollo de Software basado en Componentes
DCBSE	Distributed Component-Based Software Engineering
DCOM	Distributed Common Object Model
DCE	Distributed Computing Environment
DOM	Document Object Model
DTD	Document Type Definition
EJB	Enterprise JavaBeans
ISBC	Ingeniería del Software Basada en Componentes
JSP	Java Active Page
JSF	Java Server Faces
JVM	Java Virtual Machine
OMG	Object Management Group
RMI	Remote Method Invocation
RUP	Rational Unified Model
UDDI	Universal Description, Discovery, and Integration
UML	Unified Modeling Language
WSDL	Web Services Description Language
XML	eXtensible Markup Language

1.2. MARCO TEÓRICO

En esta sección se presentan las bases teóricas de la presente investigación, centrada principalmente en el área de la Ingeniería del Software Basada en Componentes, el contenido del marco teórico se presenta en la siguiente figura.

Figura 1. Marco Teórico de la Investigación.



1.2.1. CONCEPTOS Y DEFINICIONES FUNDAMENTALES DE LA ISBC

En esta sección se darán los conceptos y definiciones más importantes que se tratan en la ISBC para poder abarcar y entender su ciclo de vida y las actividades que este comprende. En primer lugar, el concepto de componente muchos autores lo definen de forma diferente:

Según Szyperski[4], "Un *componente* es una unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio" también dice que "Un componente es un módulo y una serie de recursos. Un módulo es una serie de clases, procedimientos y funciones. Un recurso es una colección congelada de elementos tipados".

Meyer[5] define siete criterios que debe cumplir un elemento software para poder definirse como componente:

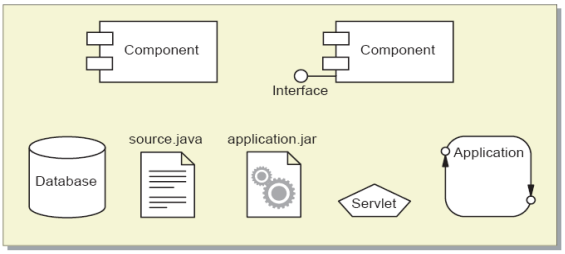
1. Debe ser usado por otros elementos software.
2. Debe ser usado por clientes sin la intervención del desarrollo de componentes.
3. Incluir una especificación de todas las dependencias.
4. Incluir una especificación de las funcionalidades que ofrece.
5. Es usado en base solo a su especificación.
6. Se puede componer con otros componentes para formar un software.
7. Puede ser integrado rápidamente y fácilmente.

En el Lenguaje Unificado de Modelamiento (UML), mas específicamente en la metodología de desarrollo utilizada por Sun Microsystems[6], definen a un componente como una unidad de software que debe tener una interfaz que lo exporte como un servicio a otros componentes, puede ser algo grande y abstracto. En esta metodología consideran

[4] Szyperski, C. (1998). Component Software. Beyond Object-Oriented Programming. Addison-Wesley.
[5] Meyer, B. (1999). The Significance of Components. Beyond Objects column, Software Development.
[6] Object-Oriented Analysis and Design Using UML. Copyright 2003 Sun Microsystems,

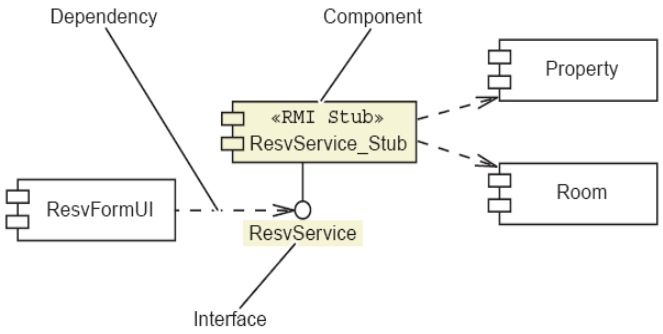
como componente software a los elementos de la base de datos, a una clase java distribuida en código fuente, a una librería de clases .jar, a un servlet, a una aplicación completa.

Figura 2. Ejemplos de Componentes.



En UML se utiliza el diagrama de componentes para mostrar las dependencias, relaciones e interfaces entre los componentes software y para representar las dependencias entre las estructuras de datos implementadas. También se utilizan para detallar el modelo de arquitectura y el modelo de distribución de un sistema informático. En la figura Nº 3 se pueden apreciar un ejemplo de diagrama de componentes.

Figura 3. Ejemplo de Diagrama de Componentes Software



Hyung Cho y John D. McGregor[7], definen el término componente como piezas software que pueden ser combinadas para construir algo más grande y completo (otro componente, subsistema, sistema).

[7] Hyung Cho y John D. McGregor. IEEE – 2005 Artículo: Component Specification for Enterprise software Development on Web Services Environment

Iribarne Martínez Luis F dice que un componente software puede ser desde una subrutina de una librería matemática, hasta una clase en Java, un paquete en Ada, un objeto COM, un JavaBeans, o incluso una aplicación que pueda ser usada por otra aplicación por medio de una interfaz especificada[8].

Después de ver las definiciones que diferentes autores dan al concepto de componente, se puede definir a un componente como una unidad software utilizada para ensamblar o componer un sistema más grande y complejo, un componente debe contener una especificación que permita identificarlo y reutilizarlo. Un elemento software para poder clasificarse como componente debe cumplir con los siguientes criterios:

- Debe ser una unidad software sin dependencia estructural de otras unidades.
- Debe tener un alto grado de cohesión.
- Debe tener una identificación, una descripción y una especificación de las funcionalidades que realiza.
- Debe seguir un modelo de componentes.
- Debe poder ensamblarse de forma rápida y fácil con otros componentes para formar un sistema más grande.
- Debe tener una interfaz que permita utilizar, modificar y adaptar las funcionalidades que maneja.
- Debe estar certificado y probado para asegurar que cumple con las funcionalidades para lo que fue implementado.
- Debe permitir el mantenimiento y la actualización de forma individual, sin afectar estructuralmente al sistema que compone.

1.2.2. PERSPECTIVAS DE LOS COMPONENTES

IRIBARNE MARTÍNEZ identifica tres perspectivas de un componente:

[8] IRIBARNE MARTÍNEZ, Luis F. Un Modelo de Mediación para el Desarrollo de Software basado en Componentes COTS. Tesis Doctoral. Universidad de Málaga. España. 2003.

La perspectiva de empaquetamiento (packaging perspective): que considera un componente como una unidad de empaquetamiento, distribución o de entrega. Algunos ejemplos de componente de esta perspectiva son los archivos, documentos, directorios, librerías de clases, ejecutables, o archivos DLL, entre otros.

La perspectiva de servicio (service perspective): que considera un componente como un proveedor de servicios. Ejemplos son los servicios de bases de datos, las librerías de funciones, o clases COM, entre otros.

La perspectiva de integridad (integrity perspective): que considera un componente como un elemento encapsulado, como por ejemplo una base de datos, un sistema operativo, un control ActiveX, una applet de Java, o cualquier aplicación en general.

1.2.3. TIPOS DE COMPONENTES REUTILIZABLES

A continuación se detallan la forma como se utilizan ciertos elementos software como componentes reutilizables, este tipo de componentes se encuentran en páginas Web que venden componentes, en artículos y libros que hablan de la ingeniería del software basada en componentes y en el desarrollo de software en general.

Clase

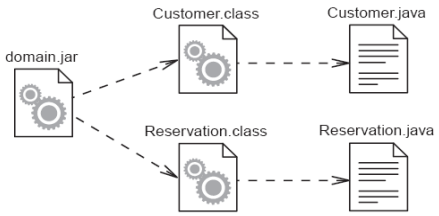
Una clase es una descripción generalizada que describe una colección de objetos similares. Las clases encapsulan datos (atributos) y los métodos que manipulan esos datos. Entre las clases existe una jerarquía de clases, en la cual los atributos y operaciones de la superclase son heredados por subclases que pueden añadir, cada una de ellas sus propios atributos y métodos. Al tener herencia las clases no se pueden instalar como una unidad software independiente, ya que algunas clases tienen relaciones con otras clases padres y/o clases hijas. También las clases no hacen referencia explicita de sus dependencias y requisitos.

Paquete / Librería / API

Un paquete es un conjunto de clases, usualmente agrupadas conceptualmente. Un paquete no es un elemento ejecutable, debe incluirse una referencia en la aplicación que quiere utilizar las clases que posee. Un paquete permite un alto grado de reutilización y es orientado a objetos.

Por ejemplo, el API que se utiliza para generar los archivos pdf en aplicaciones java, y en aplicaciones Web, se distribuye en un archivo .jar o en el paquete com.logawie.* Estos componentes son de la forma mostrada en la siguiente figura.

Figura 4. Ejemplo de Librería



Otro ejemplo de este tipo de componentes, son los componentes java que se encuentran en la página de Component Source. Esta empresa distribuye estos componentes, en un archivo *.jar con las clases que van a manejar ciertas funcionalidades para un sistema informático. Se tomo como ejemplo un componente generador de informes, que consta de javaBeans para ser utilizados en las aplicaciones que se requiera.

Módulo

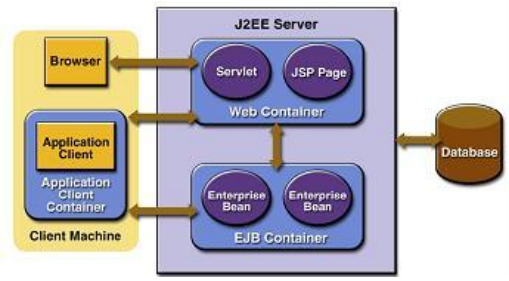
Un módulo es un conjunto de clases y otros elementos no orientados a objetos, como pueden ser procedimientos y funciones. Un módulo se utiliza para implementar las clases, procedimientos y funciones generales que se van a utilizar en toda la aplicación. Es posible que un mismo módulo se utilice en varias aplicaciones, pero al igual que las

clases no tiene una especificación que permita identificarlos claramente para hacer más fácil su reutilización, también al actualizar un módulo afecta notablemente el funcionamiento de la aplicación que lo utiliza. Ejemplo de este tipo de módulos son las dll desarrolladas en Visual Basic, que pueden ser utilizadas en aplicaciones web y aplicaciones de Windows.

Enterprise Java Beans

La especificación de JavaBeans Enterprise define una arquitectura para un sistema transacional de objetos distribuidos basado en componentes. La especificación manda un modelo de programación (API EJB). Este modelo de programación proporciona a los desarrolladores de Beans y a los vendedores de servidores EJB un conjunto de contratos que definen una plataforma de desarrollo común. El objetivo de estos contratos es asegurar la portabilidad a través de los vendedores y el soporte de un rico conjunto de funcionalidades. Se puede apreciar en la figura que los enterprise java bean se encuentran alojados en el servidor de aplicaciones. Las paginas jsp, servlets y aplicaciones cliente, los utilizan para manejar ciertas funcionalidades encapsuladas que los EJB ofrecen. Los EJB se encargan de manejar el acceso a datos, las transacciones y sesiones de la aplicación.

Figura 5. Componentes Enterprise JavaBean

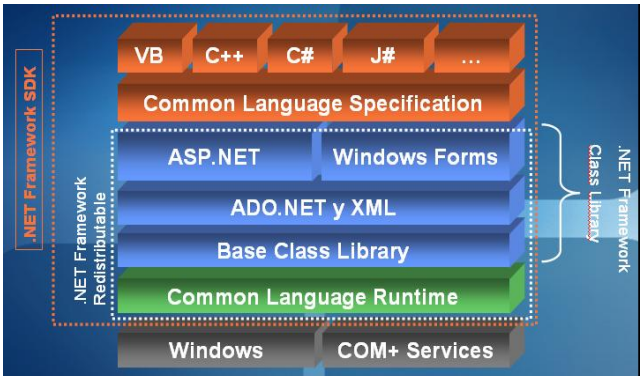


Frameworks – Marcos de Trabajo.

Un marco de trabajo puede definirse como una implementación concreta de uno o más patrones de diseño mediante componentes reutilizables, realizado a medida para un

dominio de uso específico[9]. Un marco de trabajo es un conjunto de componentes, de los cuales unos están fijos por el propio marco y otros son implementados por el usuario para especializar el marco de trabajo. Un ejemplo de marco de trabajo lo encontramos en el .NET de Microsoft que contiene un conjunto de componentes utilizados para desarrollar cualquier tipo de aplicación.

Figura 6. Framework de .NET de Microsoft



1.2.4. CICLO DE VIDA DE LA ISBC

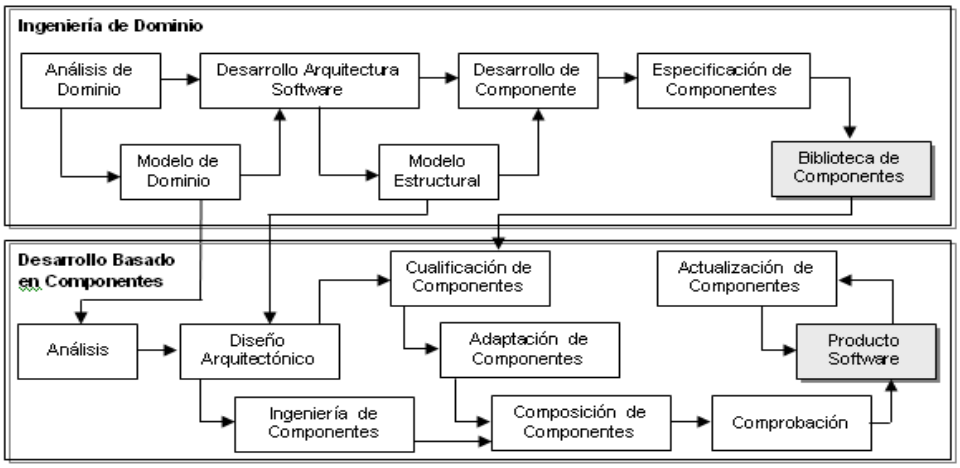
Un ciclo de vida define el conjunto de actividades que se llevan a cabo para desarrollar un producto software desde su concepción hasta su instalación y mantenimiento. En la ISBC se desarrollan dos actividades fundamentales: La ingeniería de dominio y el desarrollo basado en componentes[10]. En la figura 6 se puede apreciar el ciclo de vida propuesto por Pressman para la ingeniería del software basada en componentes.

La ingeniería del dominio explora un dominio de aplicaciones con la intención de encontrar específicamente los componentes de datos funcionales y de comportamiento candidatos para la reutilización. Estos componentes se encuentran en bibliotecas de reutilización.

[9] Johnson, R. (1997). Frameworks = (Components + Patterns). Communications of the ACM, pags. 39–42.
[10] Roger S. Pressman. Ingeniería del Software un enfoque práctico. Quinta Edición. Mc. Graw Hill.. 2002. pagina 473.

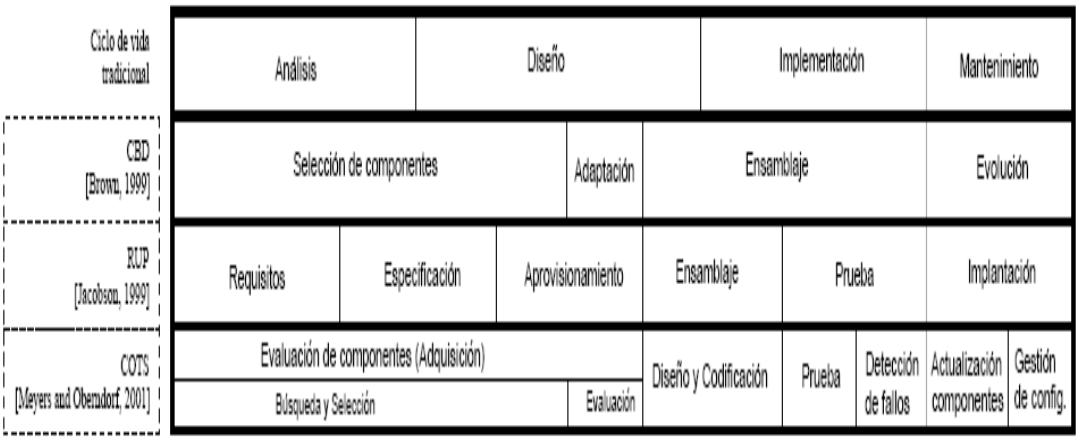
El desarrollo basado en componentes obtiene los requisitos del cliente y selecciona el estilo arquitectónico adecuado para cumplir los objetivos del sistema que se va a construir, y a continuación: (1) selecciona posibles componentes para la reutilización; (2) cualifica los componentes para asegurarse de que encajan adecuadamente en la arquitectura del sistema; (3) adapta los componentes si se deben hacer modificaciones para poderlos integrar adecuadamente; (4) integra los componentes para formar subsistemas y la aplicación completa.

Figura 7. Actividades fundamentales de la ISBC.



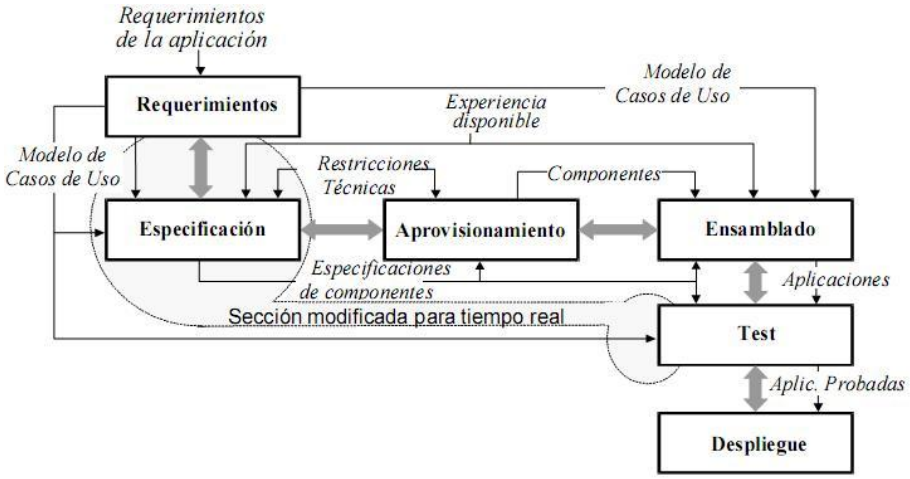
IRIBARNE MARTÍNEZ hace una comparación entre los diferentes procesos de desarrollo basados en componentes, identificando las fases más importantes que se deben cumplir para desarrollar un producto software con base en componentes, en la figura 8 se pueden apreciar estas fases y la relación con las fases de los otros procesos de desarrollo.

Figura 8. Comparación entre los diferentes procesos de desarrollo basados en componentes.



DRAKE, José M; MEDINA, Julio Luís y GONZALEZ HARBOUR, Michael[11], hacen una adaptación del proceso unificado de Rational (RUP) para el desarrollo de sistemas basado en componentes con modificaciones para tiempo real, en la figura 8, se puede apreciar este proceso, que contiene como actividades principales: requerimientos, especificación, aprovisamiento, ensamblado, prueba y despliege.

Figura 9. Adaptación del RUP para el desarrollo de componentes en tiempo real



[11] DRAKE, José M; MEDINA, Julio Luís y GONZALEZ HARBOUR, Michael. Artículo: Entorno para el Diseño de Sistemas Basados en Componentes de Tiempo Real. Grupo de Computadores y Tiempo Real. Universidad de Cantabria. Los Castros Santander – España.

Las actividades fundamentales de la ingeniería del software basada en componentes se pueden dividir en dos grades grupos, en primer lugar, las actividades necesarias para la creación de biblioteca de componentes, y el segundo, las actividades necesarias para el desarrollo de software basado en componentes.

1.2.4.1. ACTIVIDADES NECESARIAS PARA LA CREACIÓN DE LA BIBLIOTECA DE COMPONENTES

Análisis de Requisitos y Modelo de Dominio: El análisis de dominio se utiliza para identificar los componentes reutilizables que son relevantes para el dominio de aplicación de una situación determinada, intenta definir un conjunto de características que sean compartidas por todos los componentes que van a ser parte de la biblioteca de componentes que se quiere formar. También se obtienen los requisitos funcionales y no funcionales que van a manejar cada uno de los componentes. El producto final del análisis de dominio es un modelo de dominio que contiene los componentes identificados para la reutilización.

Definición de la Arquitectura Software: En esta fase se pretende caracterizar el modelo estructural del dominio de aplicación de la biblioteca de componentes que se esta formando. El modelo estructural es un estilo arquitectónico que puede y debe reutilizarse en aplicaciones pertenecientes al dominio, representa la estructura, las propiedades de los componentes reutilizables y las interrelaciones que tienen lugar entre todos los elementos arquitectónicos del dominio de aplicación.

Desarrollo de Componentes: Esta fase de Desarrollo comprende las actividades necesarias para implementar los componentes reutilizables que van a ser parte de a biblioteca de componentes. Para su implementación se puede utilizar cualquier lenguaje de programación, pero teniendo en cuenta que el desarrollo es para la reutilización.

Especificación de Componentes: Uno de los aspectos más importantes en el proceso de creación de la biblioteca de componentes reutilizables es la especificación del componente, que permite identificar y describir el componente, detallando las funcionalidades que presta, para hacer más fácil su búsqueda dentro de la biblioteca de

componentes. Un componente tiene que mostrar información suficiente para poder ser implementado y usado.

Otro de los aspectos importantes dentro del desarrollo y especificación del componente es la forma como el ingeniero software pueda encontrar el componente que realmente necesita, el ingeniero recibe el componente de la biblioteca ejecutando consultas, si hubo muchos componentes candidatos para ser utilizados, el usuario debe seleccionar el que mejor se adapte a sus necesidades. También es importante que una vez implementado el componente sea clasificado de una manera acertada dentro de la biblioteca de componentes, existen esquemas de clasificación para componentes de software reutilizables algunos de ellos son: métodos de las ciencias de la documentación y de biblioteconomía, métodos de inteligencia artificial y sistemas de hipertexto.

1.2.4.2. ACTIVIDADES NECESARIAS PARA EL DESARROLLO DE SOFTWARE BASADO EN COMPONENTES

El desarrollo basado en componentes abarca las actividades necesarias para desarrollar un producto software ensamblando componentes reutilizables disponibles en una biblioteca. El desarrollo basado en componentes se adapta a cualquier metodología de desarrollo software, después de establecer requisitos se establece un diseño arquitectónico pero en lugar de entrar inmediatamente en tareas de diseño detalladas, el equipo examina los requisitos para determinar cuáles de ellos están dispuestos para la composición, y no para la construcción.

Las actividades más importantes dentro del desarrollo basado en componentes se describen a continuación:

Análisis de Requisitos: En esta fase se realiza la selección de los componentes candidatos a reutilizar, que cumplen con los requisitos establecidos del sistema informático a crear y se adaptan al diseño arquitectónico del sistema. También se especifican los nuevos componentes que son necesarios de implementar para cumplir a satisfacción con los requerimientos del cliente.

Diseño Arquitectónico: Una vez seleccionados los componentes necesarios para ensamblar el sistema informático, se debe diseñar la arquitectura del sistema, la arquitectura define la estructura del sistema, la cual está formada por componentes software y sus relaciones. El diseño arquitectónico debe satisfacer los requisitos funcionales y no funcionales del sistema.

Selección y cualificación de Componentes: En esta actividad se asegura que el componente candidato cumple con los requisitos funcionales y con el diseño arquitectónico del sistema a crear, es decir que encaje adecuadamente en la arquitectura del sistema, también se debe verificar que cumpla con las características de calidad (por ejemplo, rendimiento, fiabilidad, usabilidad) necesarias para la aplicación.

Adaptación de Componentes: La adaptación de componentes tiene que ver con la modificación de los componentes cualificados para eliminar los conflictos que se puedan presentar al integrarlos al sistema informático a crear. Para tratar estos conflictos se suele utilizar una técnica de adaptación llamada encubrimiento de componentes, existen varios tipos de adaptación:

- El encubrimiento de caja blanca examina los detalles del procesamiento interno del componente y realiza las modificaciones a nivel de código para eliminar los conflictos.
- El encubrimiento de caja gris se aplica cuando la biblioteca de componentes proporciona un lenguaje de extensión de componentes, o API, que hace posible eliminar o enmascarar los conflictos.
- El encubrimiento de caja negra requiere la introducción de un pre-procesamiento o post-procesamiento en la interfaz de componentes para eliminar o enmascarar conflictos.

El equipo de software debe determinar si se justifica el esfuerzo requerido para envolver adecuadamente un componente o si por el contrario se debería diseñar un componente personalizado.

Composición de Componentes: La tarea fundamental de la composición de componentes es el ensamblaje de todos los componentes cualificados, adaptados y diseñados para armar el sistema informático. Este ensamble se hace teniendo como base el diseño arquitectónico del sistema.

La especificación de la interfaz lista las signaturas de los métodos suministrados por el componente, los roles jugados por el componente y las precondiciones y poscondiciones para cada método. El modelo de componentes asegura que los componentes desarrollados en distintos lenguajes de programación que residan en distintas plataformas pueden ser interoperables.

Pruebas: Una vez ensamblado el sistema se deben realizar las pruebas funcionales del sistema completo, como también las pruebas de integridad para verificar que el sistema cumple con los requisitos establecidos.

Actualización: La actualización del sistema se realiza a nivel de componente y no de todo el sistema, se realiza cambiando al componentes de forma que no afecte el funcionamiento de todo el sistema, al ensamblar una actualización de un componente se debe asegurar que no afecta al sistema que compone, debe seguir cumpliendo con los requisitos para los que fue hecho.

1.2.5. MODELOS DE COMPONENTES

Actualmente existen varios modelos de componentes, entre los que se destacan los siguientes[12]:

OMG/CORBA COMPONENT MODEL. El Grupo de gestión de objetos (Object Management Group) ha publicado una arquitectura común de distribución de objetos (OMG/CORBA). Los distribuidores de objetos proporcionan toda una gama de servicios que hacen posible que los componentes reutilizables se comuniquen con otros componentes, independientemente de su ubicación dentro del sistema.

[12] Roger S. Pressman. Ingeniería del Software un enfoque práctico. Quinta Edición. Mc. Graw Hill.. 2002. pagina 480.

.NET, COM, DCOM de Microsoft: Microsoft ha desarrollado un modelo de objetos para componentes que proporciona una especificación para utilizar los componentes elaborados por diferentes fabricantes dentro de una aplicación única bajo el sistema operativo Windows o en ambiente web.

Componentes Enterprise JavaBean de SUN. El sistema de componentes JavaBean es una infraestructura ISBC portátil e independiente de la plataforma que utiliza el lenguaje de programación Java. El sistema de componentes JavaBean acompaña un conjunto de herramientas llamadas Kit de Desarrollo Bean (BDK), que permite a los desarrolladores:

1. Analizar el funcionamiento de los Beans (componentes).
2. Personalizar su comportamiento y aspecto.
3. Establecer mecanismos de coordinación y comunicación.
4. Desarrollar Beans personalizados para su utilización en una aplicación específica.
5. Probar y evaluar el comportamiento de un Bean.

1.3. PLANTEAMIENTO DEL PROBLEMA

A lo largo de los últimos años el desarrollo de los sistemas informáticos ha venido evolucionando de tal forma que la complejidad en los lenguajes, en las herramientas utilizadas para el desarrollo y en el mismo proceso de desarrollo se ha evidenciado. Esta complejidad trae problemas enmarcados dentro de lo que se denominó "crisis del software"[13]. La crisis del software enmarca una variedad de sucesos que se pueden observar en los proyectos de desarrollo de software:

- Los proyectos no terminan en el plazo planeado.
- Los proyectos no se ajustan al presupuesto inicial.
- Baja calidad del software generado

[13] La crisis del software es un término informático acuñado en 1968, en la primera conferencia organizada por la OTAN sobre desarrollo de software, de la cual nació formalmente la rama de la Ingeniería de Software. El término se adjudica a F. L. Bauer, aunque previamente había sido utilizado por Edsger Dijkstra en su obra *The Humble Programmer*.

- Software que no cumple las especificaciones.
- Código difícil de mantener que obstaculiza la gestión y evolución del proyecto.

Estas situaciones se siguen presentando actualmente en las empresas de desarrollo software. El desarrollo de software a nivel empresarial ha evolucionado mucho los últimos años, empresas de gran prestigio como Sun Microsystem y Microsoft, se han encargado de estudiar esta problemática, y proponen soluciones creando frameworks especializados y bien formados que permiten desarrollar cualquier tipo de sistema informático. Aunque estos frameworks reducen la dificultad del desarrollo de software, se sigue presentando cierta complejidad en la tarea de programar, se tienen que conocer y aprender muchas cosas (lenguajes, patrones, procesos, herramientas, etc) y los cambios a los que se tiene que ver sometido un sistema para ser continuamente adaptado a los nuevos requisitos de los usuarios hacen sin duda del desarrollo de software una tarea de alta complejidad.

Esta problemática se puede evidenciar en la División de Servicios de Información (DSI), unidad encargada de desarrollar los productos software de la Universidad Industrial de Santander (UIS) y en las empresas de desarrollo software del medio. El desarrollo de los productos informáticos en la División de Servicios de información, se centra en varios sistemas que contienen cada uno un número considerable de aplicaciones que interactúan entre sí, manejando procesos de gran complejidad. Hay aplicaciones de gestión con varios años de antigüedad, con muchas modificaciones acumuladas, muy difíciles de mantener: la más pequeña modificación puede hacer que falle todo un sistema. También existen aplicaciones de ingeniería antiguas, pero vigentes, cuya estructura interna nadie conoce muy bien con detalle, haciendo difícil el mantenimiento y la actualización.

El desarrollo de los sistemas de información en la División de Servicios de Información se realiza utilizando varios paradigmas de la programación, existen sistemas realizados en un lenguaje procedimental y no orientado a objetos como lo es Informix4gl, en el cual se realizan los sistemas utilizando programas, funciones y una base de datos relacional. Los sistemas desarrollados en este lenguaje funcionan muy bien y prestan los servicios más importantes a la universidad, pero se presentan problemas en la falta de reutilización de los programas y rutinas desarrolladas, muchas veces se repite el desarrollo de la misma

rutina o programa varias veces, dependiendo del sistema donde se utilice. Por ser un lenguaje procedimental hace que la reutilización sea más difícil, el mantenimiento de muchos programas y miles de líneas de código desarrolladas por varias personas se hace tedioso y demorado. También al cambiar un subprograma o procedimiento se afecta el funcionamiento de los programas que lo utilizan, teniendo que hacer muchos cambios y actualizaciones en dichos programas. Al fallar un programa muchas veces no se sabe exactamente en qué procedimiento o subprograma es que ocurre realmente el error, teniendo que hacer una búsqueda exhaustiva de la falla para poder corregirla.

EL otro paradigma de programación es el utilizado para desarrollar las aplicaciones web de la universidad, para estos desarrollos se utiliza la especificación de J2EE, la cual utiliza servlets, paginas jsp y javabeans. Se realizan conexiones a una base de datos relacional. Estas aplicaciones también funcionan muy bien, aunque java es un lenguaje orientado a objetos y permite reutilizar código, pero muchas veces esta reutilización se realiza es a nivel de aplicación, no existe una librería común para todos los desarrolladores y para todas las aplicaciones, o si existe, no está bien estandarizada y especificada; lo cual conlleva a la repetición de código y de funcionalidades en cada sistema desarrollado, haciendo más difícil el mantenimiento, ya que el cambio en un proceso implica cambiar muchas cosas en las diferentes aplicaciones que utilizan o soportan este proceso. También existen aplicaciones desarrolladas en ASP (Active Server Pages) y en .NET de Microsoft, teniendo que repetir el desarrollo de muchas funciones o métodos en este lenguaje.

En la actualidad, no existen herramientas que permitan estimar, antes de comenzar el proyecto, cuál es el esfuerzo que se necesitará para desarrollar un sistema. Este hecho provoca que la mayoría de las veces no sea posible estimar cuánto tiempo llevará un proyecto, ni cuánto personal será necesario. Cuando se fijan plazos normalmente no se cumplen por este hecho.

Por último, las aplicaciones de hoy en día son programas muy complejos, inabordables por una sola persona.

Una vez analizada la situación que se presenta actualmente en el desarrollo de software se puede formular el problema de investigación:

Problema de Investigación:

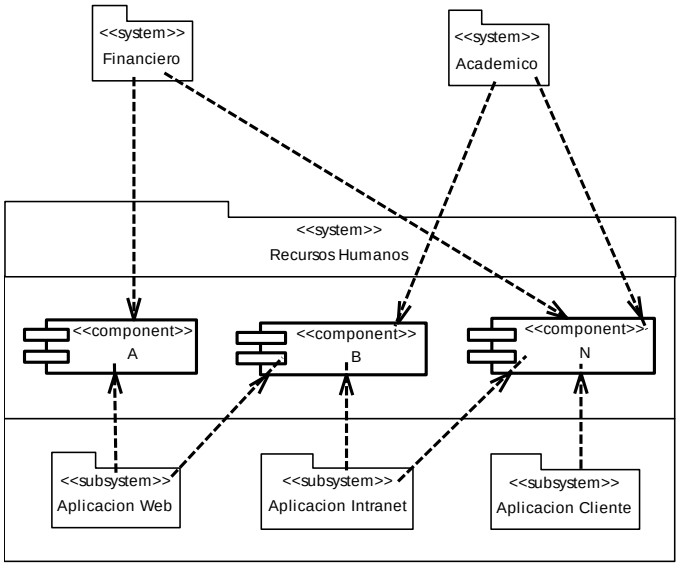
¿Cómo elaborar componentes reutilizables que sirvan de apoyo al proceso de desarrollado software?

Con la ISBC se busca desarrollar productos software a partir del ensamblaje de módulos independientes llamados componentes. Estos componentes están disponibles en una biblioteca o repositorio, donde el ingeniero desarrollador los puede obtener para utilizarlos en el nuevo sistema. Al basarse en componentes ya existentes se tiene menos software por desarrollar y por tanto los nuevos sistemas se pueden realizar con más rapidez.

En la propuesta de elaboración de un proceso de desarrollo de componentes reutilizables, se pretende hacer el estudio de los pasos que se deben llevar a cabo para poder implementar de forma adecuada los componentes reutilizables, permitiendo así que los nuevos sistemas se realicen con base en esos componentes ya creados, probados y aceptados, pudiendo así mejorar la calidad y hacer más rápido el desarrollo.

En la figura No. 10 se puede apreciar una propuesta de diagrama de componentes del sistema de información de Recursos humanos de la Universidad, en la cual las aplicaciones informáticas del sistema de Recursos Humanos se basan en componentes ya creados, utilizando sus funciones y facilidades que cada componente les ofrece. También las aplicaciones de los otros sistemas de información pueden utilizar estos componentes.

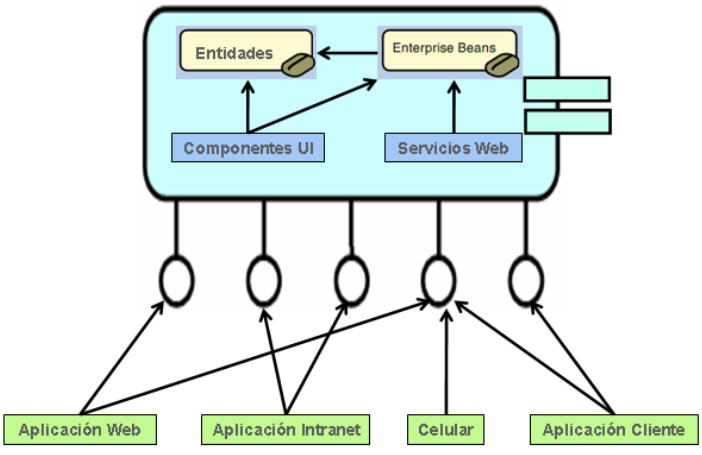
Figura 10. Propuesta de diagrama de Componentes Sistema de Recursos Humanos.



Es importante aclarar para ésta investigación cómo son los componentes software reutilizables que se pretenden implementar, de tal forma que se pueda entender el alcance y complejidad de la investigación. En la figura que aparece a continuación se puede apreciar el contenido y las características fundamentales de un componente.

Como se puede observar en la figura, el componente de estudio de la presente investigación es un conjunto de subcomponentes, los cuales se detallan a continuación:

Figura 11. Componente Software Reutilizable



- **Entidades**[14] o antiguos EJB de Entidad, son las que contienen el modelo del dominio de aplicación del componente, manejan la persistencia y almacenamiento en la base de datos.
- **Enterprise Java Beans (EJB)**, son los encargados de manejar o controlar la lógica del negocio del dominio de aplicación para el cual será implementado el componente, modelan los servicios que va a prestar el componente.
- **Componentes UI:** Son componentes personalizados para la interfaz de usuario. Utilizados para implementar los servicios que ofrece el componente ya sea en un ambiente Web o por cualquier aplicación.
- **Servicios Web,** son la implementación de la lógica de negocio del componente para ser utilizados remotamente ya sea por otros sistemas o aplicaciones que requieran esos servicios.

Con la implementación de este tipo de componentes, al ingeniero desarrollador se le presentan un conjunto de subcomponentes encapsulados en el propio componente, los cuales pueden ser utilizados de la forma como el desarrollador crea conveniente, ya sea utilizando los EJB, los controles personalizados o invocando los servicios Web que le ofrece el componente; así el componente se podría utilizar en el desarrollo de aplicaciones Web, aplicaciones cliente e inclusive en aplicaciones móviles.

Después de entender lo que pretende la ISBC y ver la forma como se podría utilizar el enfoque de desarrollo basado en componentes, se pueden plantear las preguntas que se pretenden resolver con la presente investigación.

Pregunta de Investigación 1:

Es necesario definir los criterios de selección de los componentes, para saber identificar que puede ser o no ser un componente, poder identificar que funcionalidades se pueden

[14] Los EJB de entidad, en la versión EJB 3.0 se llaman solo entidades o POJOS y fueron reemplazado por el API de persistencia de JAVA.

reutilizar en otros desarrollos para ser encapsuladas dentro de un componente, debido a lo enunciado nace el primer interrogante a resolver:

¿Cómo identificar los componentes candidatos a implementar y cuáles son los criterios de selección de estos componentes?

Pregunta de Investigación 2:

Para el desarrollo de los componentes reutilizables se requiere definir su modelo estructural, es decir, se requiere definir la forma como se pueden interrelacionar los elementos que forman el componente, por lo tanto surge la siguiente pregunta de investigación:

¿Qué alternativas de arquitectura se pueden utilizar para el desarrollo de los componentes reutilizables?

Pregunta de Investigación 3:

Otros puntos importantes para el desarrollo de componentes son la forma como se implementan de modo que se puedan reutilizar, y también la manera de realizar su especificación. Las técnicas que existen actualmente para la especificación de los componentes y de las funcionalidades que ofrece no están muy claras, como lo dicen Jean-Guy Schneider y Jun Han[15] en su artículo, por lo tanto se puede formular otro interrogante para el estudio a realizar.

¿Cómo se implementarían y especificarían los componentes software, de tal forma que se pueda hacer fácil su reutilización?

Pregunta de Investigación 4:

[15] SCHNEIDER, Jean Guy y HAN, Jun. Artículo: Components – the Past, the Present ,and the Future. School of Information Technology, Swinburne University of Technology. Hawthorn, Victoria, Australia.

Una vez implementado y especificado el componente es necesario definir el modo como los desarrolladores lo pueden utilizar, es importante saber la forma de distribuir el componente, por estos motivos se puede plantear el último interrogante de la presente investigación:

¿Cuál es el esquema de distribución y de utilización más adecuado de los componentes, de tal forma que se pueda reutilizar en varios proyectos de desarrollo?

1.4. OBJETIVOS

1.4.1. OBJETIVO GENERAL

Elaborar un proceso de desarrollo de componentes software reutilizables teniendo como base técnicas de la Ingeniería del software basada en componentes.

1.4.2. OBJETIVOS ESPECÍFICOS

- Elaborar un documento donde se recopile el análisis del modelo de componentes Enterprise Java Beans versión 3.0 de Sun, para identificar sus fortalezas y debilidades.

- Elaborar un modelo para la selección de componentes a implementar, estableciendo los criterios de selección.

- Realizar una tabla comparativa de las alternativas de arquitectura software de los componentes a desarrollar, identificando cuando utilizar cada alternativa.

- Determinar un procedimiento para el desarrollo y especificación de componentes software reutilizables.

- Diseñar e implementar un componente software reutilizable para la División de Servicios de Información.

- Elaborar el esquema de distribución y de utilización del componente, identificando los ambientes donde se puede utilizar y la forma de utilizarlo.

1.5. MARCO METODOLÓGICO

1.5.1. TIPO DE TRABAJO

El presente trabajo de investigación se basa principalmente en los conceptos de la Investigación descriptiva y la investigación tecnológica aplicada.

La investigación descriptiva tiene como objetivo describir y analizar lo que existe en la realidad con respecto a las variaciones o a las condiciones de una situación. Con ellos se obtiene información acerca de las características y comportamiento actual de fenómenos, hechos, conjunto de sujetos o áreas de interés.[16]. Para el caso de la presente investigación se pretende describir y analizar el proceso de desarrollo de componentes software reutilizables, estudiando los métodos que existen en la actualidad para el desarrollo de los componentes, analizando y definiendo los pasos que se deben llevar a cabo para crear los componentes, además se estudiará los beneficios y desventajas que tiene este nuevo paradigma de desarrollo.

En cuanto a la investigación tecnológica aplicada, la cual, aplica los conocimientos a la solución de un problema práctico inmediato generalmente particular. Se va a diseñar e implementar un componente software reutilizable para la División de Servicios de información, teniendo como base los fundamentos descritos en la parte de la investigación descriptiva.

[16] TOLEDO DÍAZ, Édison Yamir. Elementos de Metodología de la Investigación. La Habana – Cuba. Marzo 2002.

1.5.2. ESTRATEGIAS DE RECOLECCIÓN DE INFORMACIÓN

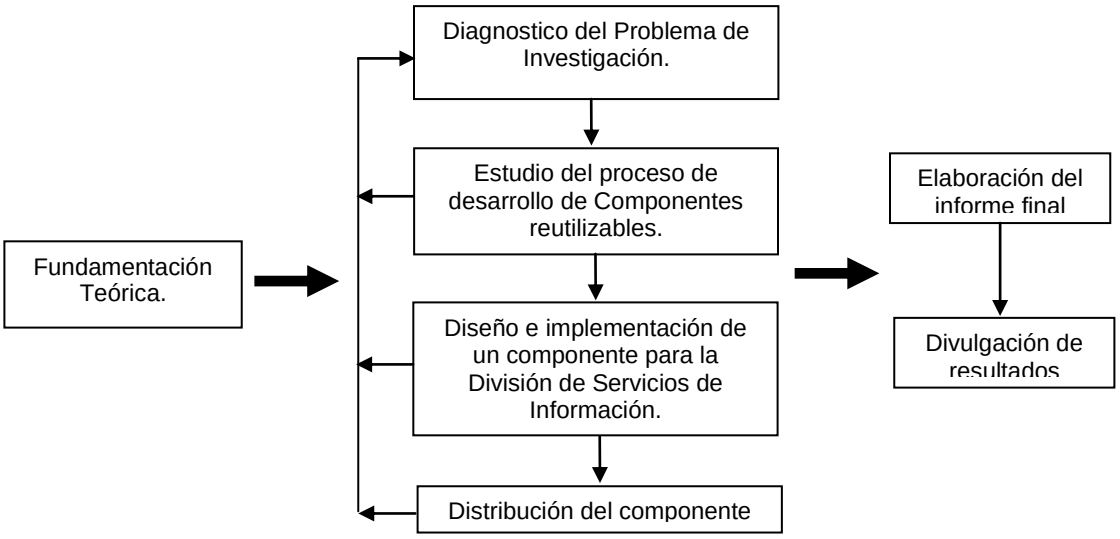
Para recopilar la información de la presente investigación se tendrá en cuenta las siguientes actividades:

- Consulta de bases de datos especializadas
- Consultas a profesores relacionados con el área de investigación, como también a desarrolladores de software expertos y a gerentes de empresas de desarrollo software.
- Consultas en Internet y en bibliotecas nacionales e internacionales que tengan libros y revistas asociados con el tema de investigación.
- Manuales y documentación en general de las empresas más importantes que proporcionan herramientas, frameworks y componentes para el desarrollo de software de Sun.

1.5.3. PROCESO DE INVESTIGACIÓN

El proceso de investigación define las actividades a realizar en el desarrollo de la investigación. Las fases fundamentales que se van a llevar a cabo se detallan en la figura 12. Se inicia la investigación con recopilación de información necesaria para tener los fundamentos teóricos de la investigación, después se realiza un diagnostico del problema que se pretende resolver, fase que se realiza iterativa e incrementalmente junto con las siguientes fases: Estudio de alternativas para el desarrollo de componentes, diseño e implementación del componente y la distribución del componente; hasta resolver las preguntas de investigación planteadas y cumplir con los objetivos. Por último se realiza la elaboración del informe final y la divulgación de los resultados.

Figura 12. Proceso de Investigación



A continuación se detallan las actividades principales que se deben desarrollar en cada una de las fases de la presente investigación.

Fase 1: Fundamentación Teórica.

En esta fase se pretende formar el marco teórico de referencia que va ser la guía para la investigación.

Actividades:

- Revisión y análisis de la literatura correspondiente al desarrollo de software basado en componentes.
- Revisión y análisis de artículos en bases de datos especializadas y en artículos de revistas a nivel nacional e internacional de investigaciones relacionadas con la ingeniería del software basada en componentes, para poder determinar las preguntas por resolver. (IEEE, ACM, InfoSCI)
- Revisión y comparación de los modelos de componentes que existen actualmente para el desarrollo de software basado en componentes. (EJB).

- Elaboración del estado del arte del desarrollo de software basado en componentes.

Resultados Esperados:

- Marco Teórico correcto y completo para la investigación.
- Viabilidad e Impacto de la Investigación.

Fase 2: Diagnóstico del Problema de Investigación.

En esta fase se pretende formular el problema de investigación, estableciendo el alcance y los objetivos que se van a llevar a cabo.

Actividades:

- Identificación de los problemas del desarrollo de Software para los cuales la ISBC es una alternativa de solución.
- Entrevista con desarrolladores de software para hacer un análisis de los problemas del desarrollo de software y de la aplicabilidad de los componentes.
- Formulación del problema de Investigación y de las preguntas a resolver.

Resultados Esperados:

- Documento con el problema de investigación claro y especifico.
- Documento que detalla los beneficios y desventajas de la utilización de componentes reutilizables en el desarrollo de software.

Fase 3: Estudio del Proceso de Desarrollo de Componentes.

En esta fase se va a analizar el modelo de componentes software de SUN y el estándar Java EE 5, para así entender y poder especificar detalladamente el proceso de desarrollo de componentes software reutilizables.

Actividades:

- Estudio del modelo de componentes Enterprise Java Beans (Versión 3.0) de Sun Microsystems.
- Estudio del API de persistencia de las entidades de JAVA (Versión 1.0)
- Estudio de la creación de Componentes UI personalizados en JAVA.
- Análisis y diseño de un modelo para la selección de componentes candidatos a implementar.
- Estudio de las alternativas de arquitectura que se pueden utilizar para el desarrollo de componentes reutilizables.
- Estudiar y adaptar un procedimiento para el desarrollo y especificación de componentes.
- Determinar el procedimiento de pruebas de los componentes reutilizables.
- Determinar las alternativas de distribución de los componentes reutilizables.

Resultados Esperados:

- Elaboración clara y completa de un proceso de desarrollo de componentes software reutilizables.
- Artículo para publicar el proceso de desarrollo de componentes.

Fase 4: Diseño e Implementación de un Componente para la División de Servicios de Información.

Actividades:

- Determinar el componente a desarrollar.
- Análisis y elección de las herramientas a utilizar para implementar el componente. La herramientas que se van a tener en cuenta son las que se utilizan para el desarrollo, prueba y distribución del componente.
- Definición de la arquitectura del componente.
- Implementación del componente siguiendo el proceso de desarrollo elaborado en la fase anterior.

Resultados Esperados:

Un componente software reutilizable implementado y probado listo para ser utilizado en el desarrollo de otros sistemas informáticos.

Fase 5: Distribución del Componente.

En esta fase se va a detallar la forma como se puede utilizar el componente desarrollado, para crear otro sistema informático más complejo.

Actividades:

- Definición del esquema de distribución del componente.
- Determinar los posibles ambientes en los que se puede utilizar el componente.
- Pruebas de integración del componente con otros sistemas.

Resultado Esperado:

- Documento donde se especifica la forma de utilizar el componente reutilizable.

Fase 6: Elaboración del Informe Final.

Actividades:

- Recopilación de los informes preliminares de las etapas anteriores.
- Elaboración de la propuesta final

Resultado Esperado:

Propuesta informática de un proceso de desarrollo de componentes reutilizables.

Fase 7: Divulgación de Resultados.

Actividades:

- Creación y publicación de un artículo para ser divulgado en una revista especializada.

Resultado Esperado:

- Artículo que informa a la comunidad académica sobre los resultados alcanzados con el desarrollo de la investigación.

2. DESCRIPCIÓN DEL MODELO DE COMPONENTES JAVA EE 5

2.1. ENTERPRISE JAVA BEAN 3.0

Los EJB se ejecutan en el contenedor EJB del servidor de aplicaciones, el cual provee servicios a nivel del sistema tales como transacciones y seguridad. Estos servicios permiten construir rápidamente y distribuir los EJB, los cuales son el corazón del las aplicaciones Java EE. Un EJB es un componente del lado del servidor que encapsula la lógica del negocio de una aplicación.

Beneficios de los EJB:

Los EJB simplifican el desarrollo de una aplicación grande y distribuida porque:

- El contenedor EJB proporciona servicios de transacciones, administración y seguridad de autorización, permitiendo al desarrollador concentrarse en los problemas de la lógica del negocio de su aplicación.
- En lugar que la aplicación cliente contenga la lógica de negocio de la aplicación, el desarrollador de la aplicación cliente no debe codificar las rutinas que implementan las reglas del negocio o acceso a bases de datos, se debe enfocar en su presentación, como resultado de esto el cliente no es pesado.
- Los EJB son componentes portables, el ensamblador de la aplicación puede construir nuevas aplicaciones utilizando componentes existentes.

Cuando usar EJBs:

- La aplicación debe ser escalable. Acomodarse a un gran número de usuarios, puede necesitar distribuir los componentes de la aplicación en múltiples máquinas.
- Las transacciones deben asegurar integridad de datos. Los EJB soportan transacciones, el mecanismo que administra la concurrencia de objetos compartidos.

- La aplicación tendrá una variedad de clientes. Con solo unas pocas líneas de código, los clientes remotos pueden fácilmente localizar al EJB. Estos clientes pueden ser finos, variados y numerosos.

2.1.1. TIPOS DE EJB.

Nota: Los Entity beans fueron reemplazados por las entidades del Java Persistence API (JPA).

Session Bean

Lleva a cabo una tarea para un cliente, opcionalmente puede implementar un Servicio Web, representa un cliente simple dentro del servidor de aplicaciones. Para acceder una aplicación que es distribuida en el servidor de aplicaciones, el cliente invoca métodos del session bean. El session bean lleva a cabo trabajos para su cliente, protegiendo al cliente de la complejidad de ejecutar tareas de la lógica del negocio dentro del servidor. No es compartido, puede tener solo un cliente, no es persistente.

Hay dos tipos de session beans:

Session Bean con estado: El estado de un objeto se compone del valor de sus variables de instancia. En un session bean con estado las variables de instancia representan el estado de una única sesión del cliente – bean. Debido a que el cliente interactúa con su bean, este estado es frecuentemente llamado estado conversacional. El estado es conservado por la duración de la sesión del cliente. Si el cliente remueve el bean o termina, la sesión finaliza y el estado desaparece.

Session Bean sin estado: Un sesión bean sin estado no mantiene el estado conversacional con el cliente. Cuando un cliente invoca los métodos del sesión bean, las variables de instancia del bean pueden contener un estado específico a este cliente, pero solo para la duración de la invocación. Cuando el método es finalizado, el estado del cliente específico no debe ser guardado. Los clientes pueden, sin embargo, cambiar el

estado de las variables de instancia en el pool de beans sin estado, y su estado es sostenido en la próxima invocación del pool, excepto durante la invocación del método, donde todas las instancias de los beans sin estado son equivalentes, permitiendo al contenedor EJB asignar una instancia a cualquier cliente. Esto es, el estado del bean sin estado debe aplicarse sobre todos los clientes. Debido a que los session bean sin estado pueden soportar múltiples clientes, pueden ofrecer mejor escalabilidad para las aplicaciones que requieran un número grande de clientes. Un session bean sin estado puede implementar un servicio Web, pero otro tipo de EJB no puede.

En general se debe usar un session bean si se posee algunas de las siguientes circunstancias:

- En algún momento dado, solo un cliente tiene acceso a la instancia del bean.
- El estado del bean es no persistente, existe solo por un periodo corto.
- El bean implementa un servicio Web.

Los session bean con estado son apropiados si algunas de las siguientes condiciones son verdaderas:

- El estado del bean representa la interacción entre el bean y un cliente específico.
- El bean necesita mantener la información acerca del cliente a través de la invocación de métodos.
- El bean intermedia entre el cliente y los otros componentes de la aplicación, presentando una vista simplificada al cliente.
- El bean maneja el flujo del trabajo de varios EJB.

Para mejorar el rendimiento, puede escoger un session bean sin estado si tiene alguna de estas características:

- EL estado del bean no tiene datos para un cliente específico.

- En una invocación simple de un método, el bean lleva a cabo una tarea específica para todos los clientes. Por ejemplo se puede usar un session bean sin estado para enviar un email que confirma una orden en línea.

Message Driven Bean

Actúa como un controlador de un tipo particular de mensajes, tales como los JMS API. Es un EJB que le permite a las aplicaciones JEE procesar mensajes asincrónicamente. Normalmente actúa como un oyente de mensajes JMS. El mensaje puede ser enviado por cualquier componente EJB o por una aplicación JMS o cualquier sistema que no use la tecnología JEE, pueden procesar mensajes JMS o cualquier tipo de mensaje.

La principal diferencia con los EJB de sesión es que el cliente del message-driven bean no lo accede a través de interfaces. A diferencia de un sesión bean este solo tiene una clase.

Todas las instancias de un message-driven bean, permiten al contenedor EJB asignar un mensaje a cualquier instancia de este tipo de EJB. El contenedor puede agrupar esas instancias para permitir flujos de mensajes para ser procesados concurrentemente.

Tienen las siguientes características:

- Ejecutan en recepción de un mensaje simple del cliente.
- Son invocados asíncronamente.
- Son relativamente de vida corta.
- Pueden ser consciente de transacciones.
- Son sin estado.

Cuando llega un mensaje, el contenedor llama el método onMessage del bean para procesar el mensaje. El método onMesage normalmente lanza el mensaje a uno de los 5 tipos de JMS mensajes y lo maneja en concordancia con la lógica del negocio de la aplicación. El método onMessage puede llamar métodos auxiliares o puede invocar un

session bean para procesar la información del mensaje o almacenarlo en la base de datos.

¿Cuándo usar un Message Driven Bean?

Los session bean permiten enviar mensajes JMS y recibirlos sincrónicamente, pero no asincrónicamente. Para evitar sobrecarga de los recursos del servidor, no se debe usar bloques sincrónicos recibidos en un componente del lado del servidor, y en general los mensajes JMS no deben ser enviados o recibidos sincrónicamente. Para recibir mensajes asincrónicamente, se debe usar un Message-Driven bean.

2.1.2. ACCESO DEL CLIENTE CON INTERFACES

Un cliente puede acceder un session bean solo a través de los métodos definidos en las interfaces del bean de negocio. Las interfaces definen las vistas del cliente de un bean. Todos los otros aspectos del bean (implementación de los métodos y configuración de distribución) están ocultas para el cliente.

Las interfaces bien diseñadas simplifican el desarrollo y mantenimiento de las aplicaciones JEE. Hacer interfaces no solo protegen al cliente de la complejidad de la capa de EJB, también permiten al bean cambiar internamente sin afectar los clientes. Es importante diseñar cuidadosamente las interfaces para separar sus clientes de los posibles cambios en el bean. Cuando se diseñan aplicaciones JEE, una de las primeras decisiones es el tipo de clientes permitidos por los EJBs: remotos, locales o servicios Web.

Clientes Remotos

Un cliente remoto tiene las siguientes características:

- Puede ejecutarse en un maquina diferente y una JVM diferente que el EJB que lo accede.

- Puede ser un componente Web, una aplicación cliente u otro EJB.
- Para un cliente remoto, la localización del EJB es transparente.

Para crear un EJB que permita acceso remoto, debe hacer:

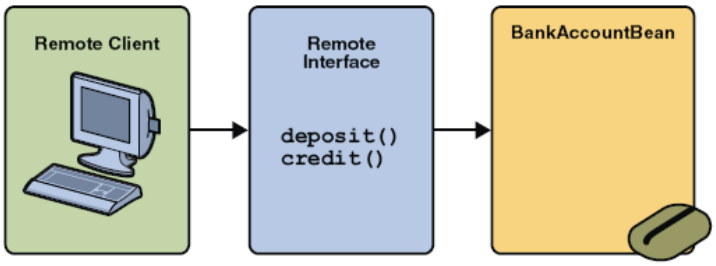
Definir la interface del EJB con la anotación @Remote:

```
@Remote(InterfaceName.class)
public interface InterfaceName { ... }
```

Definir la clase del bean con la anotación @Remote, especificando la o las interfaces:

```
@Remote(InterfaceName.class)
public class BeanName implements InterfaceName { ... }
```

Figura 13. Cliente remoto de un EJB



Clientes Locales

Tienen estas características:

- Deben ejecutarse en la misma JVM que el EJB accedido.
- Puede ser un componente Web u otro EJB.
- Para los clientes locales la localización del EJB accesado no es transparente.

Las interfaces son por defecto locales, para construir interfaces que solo permitan acceso local, debe hacer, pero no es requerido:

- Establecer la interfaz del EJB con la anotación @Local:

  ```
  @Local
  public interface InterfaceName { ... }
  ```

- Especificar la interfaz definiendo la clase EJB con la anotación @Local y especificar el nombre de la interfaz:

  ```
  @Local(InterfaceName.class)
  public class BeanName implements InterfaceName { ... }
  ```

Decidir Sobre Acceso Remoto o Local

Acoplamiento fuerte o débil entre los beans relacionados: Para incrementar el rendimiento se recomienda acceso local para EJB altamente acoplados.

Según el tipo de Cliente: Cuando los clientes se ejecutan en maquinas diferentes que el servidor de aplicaciones, se recomienda utilizar acceso remoto.

Distribución del componente: las aplicaciones JEE son escalables porque sus componentes del lado del servidor pueden ser distribuidos a través de múltiples maquinas. Para este caso se recomienda acceso remoto.

Rendimiento: Debido a diferentes factores, invocaciones remotas pueden ser más lentas que invocaciones locales. Por otro lado, si se distribuyen los componentes a través de diferentes servidores, se puede mejorar el rendimiento global de la aplicación. El rendimiento puede variar en diferentes medios operacionales, pero es un factor importante a tener en cuenta en el diseño de la aplicación, ya que el modo como se diseñe el acceso remoto o local puede afectar el rendimiento de la aplicación.

Si no se está seguro del tipo de acceso del EJB, se debe selleccionar acceso remoto, ya que da mayor flexibilidad. En el futuro se pueden distribuir sus componentes para acomodar la gran demanda de su aplicación. La misma interfaz no pude ser local y remota al tiempo.

Cliente Servicio Web

El servicio Web puede acceder una aplicación JEE en dos modos: primero, el cliente puede acceder un servicio Web creado con JAX-WS, segundo, un servicio Web puede invocar métodos de negocio de un sesión bean sin estado. Message Driven beans no pueden ser accedidos por el cliente servicio Web.

Suministrando los protocolos correctos cualquier servicio Web puede acceder un session bean sin estado, estén o no escritos en java. Además, los EJB y los componentes Web pueden ser clientes de los servicios Web. Esta flexibilidad permite integrar las aplicaciones JEE con servicios Web.

Un servicio Web cliente accesa un session bean sin estado a través de la clase Web service end point del bean. Por defecto, todos los métodos públicos de la clase son accesibles por el servicio Web cliente. La anotación @WebMethod puede ser usada para personalizar el comportamiento de los métodos del servicio Web. Si la anotación @WebMethod es usada solo los métodos definidos con esta anotación son expuestos al servicio Web cliente.

2.1.3. CICLO DE VIDA DE UN EJB

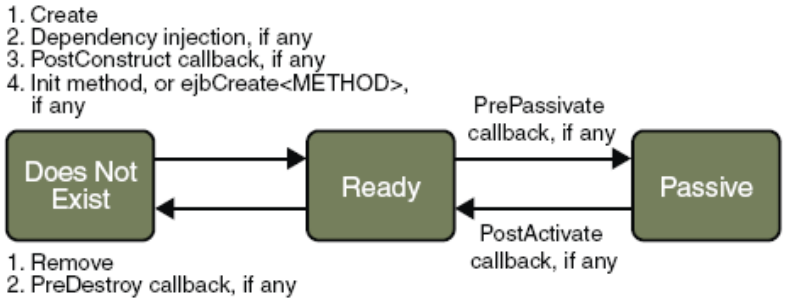
Cada tipo de EJB tiene un ciclo de vida diferente.

Ciclo de Vida de un session Bean con estado:

En la figura se pueden apreciar los estados por los cuales pasa un session bean durante su tiempo de vida. El cliente inicia el ciclo de vida obteniendo una referencia a un session bean sin estado. El contenedor lleva a cabo alguna inyección de dependencia e invoca el

método anotado con @PostConstruct. El bean ahora esta listo para tener sus invocaciones de métodos de negocio invocados por el cliente.

Figura 14. Ciclo de vida de un Session Bean con estado.



Mientras se encuentre en el estado READY, el contenedor puede decidir desactivar (Passive) el bean moviéndolo desde la memoria a almacenamiento secundario. El contenedor ejecuta el método anotado con @PrePassivate, inmediatamente después de desactivarlo. Si el cliente invoca un método de negocio sobre el bean mientras se encuentra en estado desactivado (Passive), el contenedor EJB activa al bean, llama al método anotado con @PostActivate y lo mueve al estado READY.

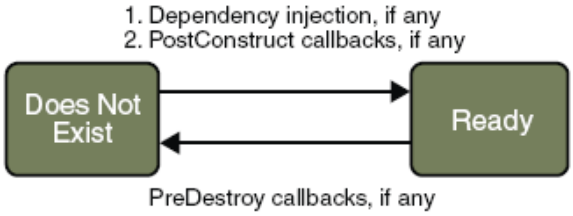
Al final del ciclo de vida, el cliente invoca un método anotado con @Remove, y el contenedor EJB llama al método anotado @PreDestroy. La instancia del bean queda listo para el recolector de basura (garbage collection).

Su código controla la invocación de solo un método del ciclo de vida: el método anotado con @Remove. Todos los otros métodos son invocados por el contenedor EJB.

Ciclo de Vida de un Session Bean sin Estado:

Debido a que el session bean sin estado nunca es desactivado (passive), su ciclo de vida tiene solo dos estados: no Existente y listo (READY) para la invocación de métodos. En la figura se muestra estos estados.

Figura 15. Ciclo de vida de un Session Bean sin estado.

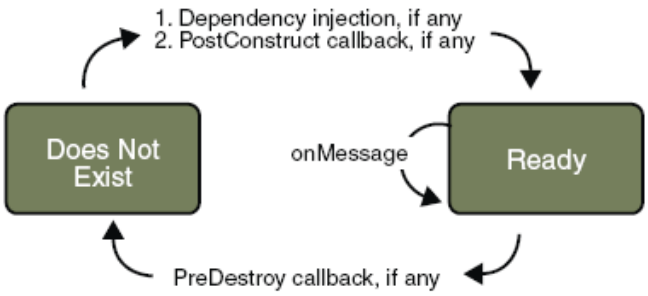


El cliente inicia el ciclo de vida obteniendo una referencia a un session bean sin estado. El contenedor lleva a cabo cualquier dependencia de inyección y entonces invoca el método anotado con @PostConstruc. El bean está ahora listo para la invocación de los métodos de negocio por parte del cliente. Al final del ciclo de vida el contenedor EJB llama al método anotado con @PreDestroy, la instancia del bean queda lista para el recolector de basura (garbage collection).

Ciclo de Vida de un Message-Driven Bean

En la figura se muestra el ciclo de vida de un message-driven bean.

Figura 16. Ciclo de vida de un Message-Driven Bean.



El contenedor EJB usualmente crea un pool de instancias de message driven bean. Para cada instancia el contenedor EJB realiza estas tareas:

1. Si el bean usa un dependencia de inyección, el contenedor inyecta esta referencia antes de instanciar el objeto.

2. El contenedo llama al método anotado con @PostConstruct.

Como los session bean sin estado, un message-driven bean nunca es desactivado y solo tiene dos estados: No existe y Listo (Ready). Al finalizar el ciclo de vida el contenedor ejecuta el método anotado con @PreDestroy y el bean queda listo para el recolector de basura. (garbage collection).

Para crear un EJB se debe realizar la interfaz de negocio que especifica los métodos del EJB y la clase que implementa estos métodos. Estas clases se compilan y se empaquetan en un archivo .jar. Una aplicación cliente de este EJB también se empaqueta en un archivo .jar. Un cliente Web se empaquete en un archivo .war.

2.2. API DE PERSISTENCIA DE JAVA

El Api de persistencia de Java provee el mapeo entre el mundo de objetos / relacional para los desarrolladores Java, sirve para manejar datos relacionales en aplicaciones Java. Se divide en tres areas:

- El Api de persistencia de Java
- El JPQL: lenguaje de consulta
- Objeto / Relacional metadata de mapeo.

2.2.1. ENTIDADES

Una entidad es un objeto de dominio que tiene persistencia en la base de datos. Típicamente una Entidad representa una tabla en una base de datos relacional, cada instancia de la entidad corresponde a un registro en la tabla.

El estado de persistencia de una entidad es representado por medio de campos de persistencia o por propiedades de persistencia. Estos campos o propiedades usan

anotaciones de mapeo entre Objeto / Relacional para mapear las entidades o relaciones entre entidades, con los datos relacionales en la base de datos.

Requerimientos para las Clases de Entidad

- Debe tener la anotación javax.persistence.Entity.
- La clase debe tener obligatoriamete un constructor sin argumentos public o protected, aunque la clase puede contener otros costructores.
- La clase no debe ser declarada como final. Los métodos o instancias de persistecia no deben ser declarados como final.
- Si la clase va a ser accedida remotamente debe implementar la interfaz Serialezable.
- Las entidades pueden heredar de otras entidades o de clases que no sean de entidad, y también las clases no entidad pueden extender entidades.
- Las variables de instancia de persistencia deben ser declaradas privadas, protegidas o package – private, y pueden solo ser accedidas directamente por los métodos de la clase de entidad. Los clientes deben acceder al estado de la entidad a través de métodos de la lógica del negocio o métodos get.

Campos de Persistencia y Propiedades en una Clase de Entidad

Los campos o propiedades deben ser de los siguientes tipos de datos del lenguaje Java:

- Tipos primitivos de Java
- Java.lang.String
- Wrappers(Envolventes) de los tipos primitivos de java.
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time

- java.sql.TimeStamp
- Tipos de datos serializables definidos por el usuario.
- Tipos Enumerated
- Otras entidades o lista de entidades.
- Clases embebidas.

Las entidades pueden usar cualquiera de las dos, campos persistentes(anotaciones en los atributos de la clase) o propiedades persistentes (anotaciones en los métodos get). No se pueden utilizar ambos tipos de persistencia se debe escoger uno solo.

Si la entidad usa campos persistentes, la runtime de Persistencia accede directamente a los atributos de la clase. Todos los campos no anotados con javax.persistence.Transient o no marcados como Java transient serán persistentes en la base de datos. Las anotaciones de mapeo entre Objeto / Relacional pueden ser aplicadas a los atributos de la entidad.

Si la entidad usa propiedades persistentes, las anotaciones del mapeo Objeto / Relacional deben ser aplicadas en los métodos get. Las anotaciones de mapeo no pueden ser aplicadas a los campos o propiedades anotadas como @Transient o marcadas transient.

Llaves Primarias en las Entidades

Cada entidad tiene un unico objeto identificador, que permite localizar una instancia en particular rápidamente. Toda entidad debe tener una llave principal. Una entidad puede tener una llave primaria simple o una llave primaria compuesta.

Una llave primaria simple debe tener la anotacion javax.persistence.Id en una propiedad o campo.

Las llaves primarias compuestas deben ser definidas en una clase auxiliar de llave primaria. Debe ser anotada usando las anotaciones javax.persistence.EmbeddedId y

javax.persistence.IdClass. Los atributos o propiedades de una clase de llave compuesta deben ser alguno de los siguientes tipos de datos java:

- Tipos primitivos de Java.
- Wrappers de los tipos primitivos Java
- Java.lang.String
- Java.util.Date
- Java.sql.Date

Tipos de punto flotante nunca deben ser usados en llaves primarias. Si usa una llave primaria generada, solo tipos integrados serán portables.

Una Clase de llave primaria debe reunir los siguientes requerimientos:

- El control de la clase o modificador de la clase debe ser público.
- Las propiedades de la Clase de llave primaria deben ser públicas o protegidas.
- La clase debe tener un constructor público por defecto.
- La clase debe implementar los métodos hashCode(), equals(Object o) y toString() de Object obligatoriamente.
- La clase debe ser serializable
- Una llave primaria compuesta debe ser representada y mapeada a múltiples campos o propiedades de la clase de entidad, o debe ser representada y mapeada como una clase embebida.
- Si la clase es mapeada a múltiples campos o propiedades de la clase de entidad, los nombre y tipos de los campos o propiedades de la llave primaria en la clase de llave primaria deben coincidir con los de la clase de entidad.

2.2.2. RELACIONES ENTRE LAS ENTIDADES

Multiplicidad en las Relaciones entre Entidades

Hay cuatro tipos de multiplicidad:

Uno a Uno (One-to-one): Cada entidad se relaciona con una sola instancia de la otra entidad. Usan la anotación javax.persistence.OneToOne en el correspondiente campo o propiedad.

Uno a Muchos (One-to-many): Una entidad está relacionada con múltiples instancias de otra entidad. Se utiliza la anotación javax.persistence.OneToMany.

Muchos a Uno (Many-to-one): Múltiples instancias de una entidad se relacionan con una instancia simple de otra entidad. Utiliza la anotación javax.persistence.ManyToOne.

Muchos a Muchos (Many-to-many): Las instancias de la entidad pueden estar relacionadas con varias instancias de otra entidad. Usan la anotación javax.persistence.ManyToMany.

Dirección en las Relaciones entre Entidades

- **Relaciones Bidireccionales**

Posee una parte dueña de la relación y otra parte inversa. La parte dueña determina cómo la rutina de persistencia hace las actualizaciones en la base de datos.

Cada entidad tiene un campo o propiedad que hace referencia a otra entidad. A través de la relación, el código de una clase entidad puede acceder a su objeto relacionado y viceversa.

Debe seguir las siguientes reglas:

  - El lado inverso de la relación bidireccional debe referirse al lado poseedor usando el elemento mappedBy de las anotaciones @OneToOne, @OneToMany, @ManyToMany. El mappedBy designa la propiedad o el campo en la entidad que es la dueña de la relación.

- El lado Muchos en la relación Muchos – a – uno (Many-to-One) no debe definir el elemento mappedBy. El lado Muchos siempre es el dueño de la relación.
- Para la relación bidireccional uno – a – uno (One-to-One), el dueño de la relación corresponde al lado que contiene la llave foránea correspondiente.
- Para la relación Muchos – a – Muchos (Many-to-Many) cualquier lado puede ser el dueño de la relación.

- **Relaciones Unidireccionales**

Posee solo la parte dueña de la relación. Solo una entidad tiene el campo o propiedad que se refiere a la otra entidad. Solo una entidad tiene la referencia a la otra entidad, mientras la otra entidad no conoce a la entidad que la refiere.

Las consultas del JPQL ofrecen navegación a través de las relaciones. La dirección de una relación determina cuando una consulta puede navegar de una entidad a otra.
La eliminación en cascada se especifica usando el elemento cascade=REMOVE especificado para las relaciones @OneToOne y @OneToMany.

2.2.3. HERENCIA DE ENTIDADES

Las entidades soportan herencia entre clases, asociaciones polimorfitas y consultas polimorficas. Pueden heredad de clases que no son de entidad, y clases no entidad pueden heredar de clases de entidad. Las clases de entidad pueden ser abstractas y concretas.

- **Entidades Abstractas**

Para definir una entidad como clase abstracta se utiliza la anotación @Entity.Abstract. Las entidades abstractas pueden ser consultadas solo como consultas concretas. Si una entidad abstracta es el objetivo de una consulta, la consulta opera sobre todas las subclases concretas de la entidad abstracta.

- **Superclases Mapeadas**

Las entidades pueden heredar de superclases que contienen estados persisitentes e información de mapeo, pero no son entidades. Es decir, la superclase no es anotada con @Entity, y no es mapeada como una entidad por el proveedor de Persistencia de Java. Las superclases son comúnmente utilizadas cuando se tienen estados e información de mapeo común a múltiples clases de entidad.

Las superclases mapeadas no se pueden consultar, no pueden ser usadas en operaciones del EntityManager o consultas. Se debe usar las subclases de entidad de la superclase mapeada en operaciones del EntityManager o consultas. Las superclases mapeadas no pueden ser el blanco de relaciones de entidades y pueden ser abstractas o concretas.

Las superclases mapeadas no tienen tablas correspondientes en la base de datos. Las entidades que heredan de ella definen el mapeo a las tablas.

- **Superclases No Entidad.**

Las entidades pueden tener superclases no entidad, El estado e estas clases no es persistente, el estado heredado de ella es no persistente, no pueden ser usadas por operaciones del EntityManager o consultas. Cualquier mapeo o anotaciones de relación en estas clases son ignorados.

Estrategias de Mapeo en la herencia de entidades

Se puede configurar como el proveedor de Persistencia de Java mapea la entidades heredadas a la base de datos, decorando la clase raiz (root class) con la anotación javax.persistence.Inheritance. Existen tres tipos de mapeo:

- Una tabla simple por jerarquía de clase
- Una tabla por clase concreta.

- Una estrategia “join”, cuando los campos o propiedades que son especificas a la subclase son mapeados a una tabla diferente que los campos o propiedades que son comunes a la clase padre.

Las estrategia de mapeo se definen en enumerador javax.persistence.InheritanceType:

```
Public enum InheritanceType {
        SINGLE_TABLE,
        JOINED,
        TABLE_PER_CLASS
}
```

La estrategia por defecto es SINGLE_TABLE. En esta estrategia todas las clases en la jerarquia son mapeadas a una tabla simple en la base de datos. Esta tabla tiene una columna discriminadora que contiene un valor que identifica la subclase a la cual la instancia es representada por la fila correspondiente.

En la estrategia de una tabla por clase concreta (TABLE_PER_CLASS), cada clase concreta es mapeada a una tabla separada en la base de datos. Todos los campos o propiedades en la clase, incluyendo campos o propiedades heredadas, son mapeados a columnas en la tabla de la clase en la base de datos. Esta estrategia no es soportada por todos los proveedores de Persistencia de Java.

En la estrategia “JOINED” subclases, la clase raíz de la jerarquía de clases es representada por una simple tabla, y cada subclase tiene una tabla separada que solo contiene aquellos campos específicos a la subclase. La tabla de la subclase además tiene una columna o columnas que representa su llave primaria, la cual es una llave foránea a la llave principal de la tabla de la superclase.

2.2.4. ADMINISTRANDO CLASES DE ENTIDAD

Las entidades son manejadas por el entity manager. El Entity Manager es representado por las instancias de javax.persistence.EntityManager, cada instancia es asociada con un contexto de persistencia. Un contexto de persistencia define el alcance bajo el cual instancias de entidades particulares son creadas, persistidas y removidas.

El contexto de Persistencia

Un contexto de persistencia es una seria de instancias de entidades manejadas que existen en una base de datos particular. La interfaz EntityManager define los métodos que son usados para interactuar con el contexto de persistencia.

La interfaz EntityManager

El API EntityManager crea y remueve instancias de entidades persistentes, encuentra entidades por su llave primaria y permite realizar consultas sobre las entidades.

- **Cotainer-Managed Entity Manager**

Con un container – manager entity manager, una instancia EntityManager del contexto de persistencia es automanticamente propagada por el contenedor a todos los componentes de la aplicación que usan la instancia del EntityManager dentro de una simple transacción del Java Transaction Architecture (JTA).

Una transacción JTA usualmente involucra llamadas a través de componentes de la aplicación. Al completar una transacción JTA. Estos componentes usualmente necesitan acceder a un contexto de persistencia simple. Esto ocurre cuando un EntityManager es inyectado dentro de los componentes de la aplicación por medio de la anotación javax.persistence.PersistenceContext. El contexto de persistencia es automáticamente propagado con la actual transacción JTA, y las referencia del EntityManager que son mapeadas a las mismas unidades de persistencia suministrando acceso al contexto de persistencia dentro de aquella transacción. Por propagarse automáticamente el contexto de persistencia, los componentes de la aplicación no necesitan pasar referencias de las

instancias del EntityManager para ejecutar los cambios en una transacción simple. El contenedor Java EE maneja el ciclo de vida del container-managed entity managers.

Para obtener una instancia del EntityManager, inyecta el entity manager dentro de los componentes de la aplicación:

```
@PersistenceContext
EntityManager em;
```

- **Aplication – Manager Entity Managers**

Con application-mannaged entity managers, el contexto de persistencia no es propagado a los componentes de la aplicación, y el ciclo de vida de la instancia del EntityManager es manejada por la aplicación. Son usados cuando la aplicación necesita acceder al contexto de persistencia que no es propagado con la transacción JTA a través de las instancias del EntityManager en una unidad de persistencia particular. En este caso, cada EntityManager crea un nuevo contexto de persistencia aislado. El EntityManager, y su contexto de persistencia asociado, es creado y destruido explícitamente por la aplicación.

La aplicación crea una instancia del EntityManager usando el método createEntityManager de la anotación javax.persistence.EntityManagerFactory. Para obtener una instancia del EntityManager, primero se debe obtener una instancia del EntityManagerFactory por inyección dentro de los componentes de la aplicación por medio de la anotación javax.persistence.PersistenceUnit:

```
@PersistenceUnit
EntityManagerFactory emf;
```

Entonces, para obtener un EntityManager desde la instancia del EntityManagerFactory se utiliza

```
EntityManager em = emf.createEntityManager();
```

- **Buscando entidades con el Entity Manager**

Se utiliza el método EntityManager.find, busca la entidad por medio de su llave primaria.

- **Manejando el ciclo de vida de una instancia de una Entidad**

Las entidades se manejan por medio de la invocación de operaciones de la entidad utilizando una instancia del EntityManager. La instancia de la Entidad puede estar en cualquiera de los siguientes estados (New, managed, datached or removed).

Una instancia de entidad nueva (New) no tiene identidad de persistencia y no está asociada aun con un contexto de persistencia.

Una instancia de entidad manejada tiene una identidad de persistencia y esta asociada con un contexto de persistencia.

Las instancias de entidad sueltas (datached) tienen una identidad de persistencia y no están actualmente asociadas con un contexto de persistencia.

Las instancias de entidad removidas tienen identidad de persistencia, están asociadas con un contexto de persistencia, y están programadas para removerse de la base de datos.

- **Persistiendo Instancia de Entidades**

Nuevas instancias de entidades llegan a ser manejadas y persistentes con la invocación del método persist o por operaciones persistentes en cascada invocadas por entidades relacionadas que tienen los elementos cascade=PERSIST o cascade=ALL asigandos en las anotaciones de relación. Esto significa que los datos de las entidades son almacenados en la base de datos cuando la transacción asociada con las operaciones de persistencia es completada. Si la entida y es manejada, la operación de persistencia es

ignorada, aunque la operación persist serán manejadas en cascada si las entidades relacionadas tienen el elemento cascade asignado a PERSIST o ALL en las anotaciones de relación. Si persist es llamado sobre una instancia de entidad Removida, pasa a ser manejada. Si la instancia de la entidad está en el estado Datached el método persist lanza una IllegalArgumentException, o el commit de la transacción falla.

- **Removiendo Instancias de Entidades**

Instancias de entidad manejadas son removidas por la invocación del método remove, o por operaciones de eliminación en cascada invocadas desde entidades relacionadas que tienen los elementos cascade=REMOVE o cascade=ALL asignados en las anotaciones de relación. Si el método remove es invocado sobre una entidad en estado NEW, la operación es ignorada, aunque se propaga a las entidades relacionadas que tengan el elemento cascade en ALL o REMOVE. Si remove es invocado sobre una entidad en estado Datached se lanza la excepción IllegalArgumentException, o el commit de la transacción falla. Si el método remove es invocado sobre una entidad ya removida, es ignorado. Los datos de la entidad son removidos de la base de datos cuando la transacción es completada, o como un resultado de la operación flush.

- **Sincronizando los Datos de la Entidad con la base de datos**

El estado de persistencia de las entidades es sincronizado a la base de datos cuando la transacción con la cual la entidad es asociada es terminada por un Commit. Si una entidad manejada está en una relación bidireccional con otra entidad manejada, los datos serán persistidos basados en el lado dueño de la relación. Para forzar la sincronización de una entidad con los datos almacenados, se invoca el método flush de la entidad. Si se tienen entidades relacionadas con el elemento de la anotación de relación cascade en PERSIST o ALL, los datos de las entidades relacionadas serán sincronizados con la base de datos. Si la entidad es removida, el llamado al método flush elimina la entidad de la base de datos.

- **Creando Consultas.**

Los métodos para crear consultas a los datos almacenado usando JPQL son: EntityManager.createQuery y EntityManager.createNamedQuery. El método createQuery es usado para crear consultas dinámicas, que están definidas directamente dentro de la lógica del negocio de la aplicación.

El método createNameQuery es usado para crear consultas estaticas, que son definidas usando la anotación javax.persistence.NamedQuery. El elemento name de la anotación @NamedQuery especifica el nombre de la consulta que será usado en el método createNamedQuery, el elemento query de la anotación es la consulta en JPQL.

- **Parámetros Nombrados en Queries**

Los parámetros nombrados son parámetros en una consulta que están prefijados con (:). Los parámetros son asignados por el método javax.persistence.Query.setParameter(String name, Objetc value).

- **Parámetros posicionales en Queries**

Se pueden utilizar parámetros posicionales utilizando (?) seguido por el número de la posición del parámetro en la consulta. El método Query.setParameter(integer posición, Object value) es usado para asignar el valor de los parámetros.

2.2.5. UNIDADES DE PERSISTENCIA

Una unidad de persistencia define el conjunto de todas las clases de entidad que son manejadas por las instancias del EntityManager en una aplicación. Este conjunto de clases de entidad representan los datos contenidos en la base de datos.

Las unidades de persistencia están definidas por el archivo de configuración persistence.xml. El archivo JAR o directorio META – INF que contiene el persistence.xml

es llamada la raíz de las unidades de persistencia. El alcance de las unidades de persistencia es determinado por el root de las unidades de persistencia.

Cada unidad de persistencia debe ser identificada con un nombre que es único al alcance de las unidades de persistencia.

Las unidades de persistencia pueden ser empaquetadas como parte de un archivo WAR o EJB JAR, o pueden ser empaquetadas como una archivo JAR que puede entonces ser incluido en un archivo WAR o EAR.

Si se empaquetan como un conjunto de clases en un archivo EJB JAR, el persistence.xml debe ser puesto en el directorio META-INF del EJB JAR.

Si se empaquetan como un conjunto de clases en un archivo WAR, persistence.XML debe ser localizado dentro del directorio WEB-INF/clases/META-INF del archivo WAR.

Si se empaquetan dentro de un archivo JAR para ser incluido dentro de un WAR o EAR, el JAR debe ser localizado en:

- En el directorio WEB-INF/lib del WAR
- En el primer nivel de un EAR.
- En el directorio de librerías del EAR.

2.3. COMPONENTES PERSONALIZADOS PARA LA INTERFAZ DE USUARIO

Los componentes de la interfaz de usuario (Componentes UI) son los elementos de un software que establecen la comunicación directa entre el cliente y el sistema. Se encargan principalmente del manejo de eventos producidos por el cliente, validación de datos y definen la navegación del cliente sobre las pantallas o paginas.

En la actualidad para el desarrollo de aplicaciones web en Java existen varios frameworks que contienen componentes reutilizables para la interfaz de usuario, entre estas librerías

de componente encuentran Java Server Faces (JSF) que es la más utilizada y Richfaces que es una implementación de componentes JSF utilizando AJAX para su funcionamiento.

Las características principales del framework JSF definidas en la guía del kit de desarrollo de componentes richfaces[17] son:

- El framework es una arquitectura basada en componentes.
- El componente JSF no es solo una serie de código HTML enviado e interpretado por un explorador web, el componente JSF es una combinación de elementos del lado del cliente con elementos del lado del servidor que representan al componente, incluye validación de datos, manipulación de eventos, conexión con la lógica del negocio de la aplicación, etc.
- JSF permite un paradigma orientado a componentes para construir interfaces de usuario bien diseñadas, personalizables basadas en componentes reutilizables.

Uno de los principales problemas para desarrollar un componente JSF es que el tiempo de implementación es muy largo y el desarrollo es complejo. Para la creación de un componente JSF se requiere implementar:

- La clase que hereda de UIComponent, que es la clase base para todos los componentes.
- La clase Tag, que define la etiqueta requerida para utilizar el componente.
- La clase Converter si es necesaria. Que se utiliza para convertir los valores introducidos al componente en un objeto java determinado.
- La clase Renderer en caso que sea necesaria, la cual es la encargada de traducir la etiqueta JSF que representa el componente a código HTML.
- El archivo de configuración facesconfig.xml
- En el momento de utilizarlo en un ambiente facelet se debe suministrar el *.tablib.xml.

[17] SERGEY, Smirnov. CDK Developer Guide of RichFaces. Jboss Comunitiy. 2008. Capitulo 1 página 1.

Para la creación de un componente de richfaces se requiere inclusive más tiempo. Adicionalemente se debe proveer:

- ListenerTagHandler
- Una clase para crear una interface *listener*.
- Un método de procesamiento de evento en la interfaz *listener.*
- Una clase Evento.
- Una clase Renderer específica para cada posible render kit utilizado con el componente.

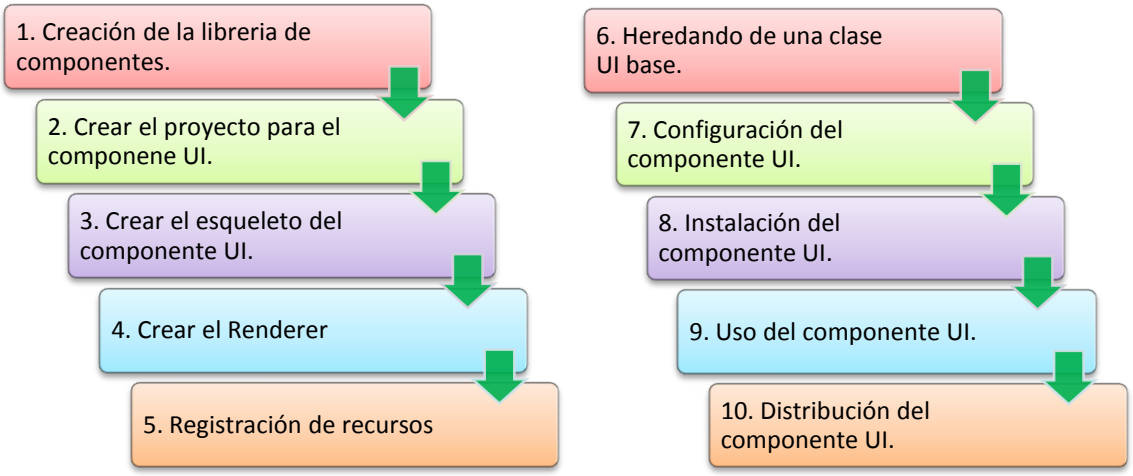
A pesar de la complejidad, el proceso de creación de componentes JSF es repetible según dicen Jonas Jacobi y John R. Fallows en el libro "Pro JSF and Ajax Bulling Rich Internet Components", en el cual definen el proceso en detalle. Con base en esta descripción la gente de Jboss realizaron el Kit de desarrollo de componentes de richfaces, el cual permite una creación más rápida y automatizada de componentes richfaces con soporte Ajax.

Las características principales del CDK de richfaces son:

- Inicio rápido del desarrollo. El desarrollo de un nuevo componente inicia con una estructura pre generada, la cual contiene todos los archivos necesarios para el desarrollo.
- Solo se necesita especificar las propiedades del componente en un meta-data y el código específico del componente. Todos los otros artefactos descritos atrás son generados automáticamente.
- Ciclo de vida del desarrollo independiente, cada componente se desarrolla separadamente de los otros.
- Facilidad de automatizar las pruebas.
- Soporte para Ajax y para adicionar Ajax a componentes existentes.
- Optimización para diferentes implementaciones de JSF.

Los pasos necesarios para la creación un componente reutilizable se puede apreciar en la siguiente figura.

Figura 17. Fases para la creación de componentes UI.



A continuación se detallan las actividades principales que se tienen que hacer en cada fase hasta tener implementado el componente UI. Para una mayor referencia se puede estudiar el documento CDK developer guide.

2.3.1. CREACIÓN Y CONFIGURACIÓN DE LA LIBRERÍA DE COMPONENTES UI.

Antes de crear la librería de componentes es necesario tener instaladas las siguientes herramientas, las cuales permiten crear la librería de componentes y los propios componentes. Las herramientas son:

- Java SE 5 Development Kit (JDK)
- Apache Maven 2.0.9
- Apache Tomcat 6.0
- Browser (Internet explorer 7 o mozilla firefox)

En primer lugar se debe tener configurado el repositorio de Maven, luego se debe crear un directorio donde quedaran almacenados los componentes, dentro del directorio se

debe crear el archivo de configuración pom.xml. El contenido de los archivos de configuración y del pom.xml se puede consultar en el documento CDK developer guide a partir de la página 6.

Los elementos más importantes del archivo de configuración pom.xml de la librería de componente creada se pueden apreciar en la siguiente tabla.

Tabla 1. Elementos del pom.xml de la Librería de componentes UI.

ELEMENTOS DEL pom.xml DE LA LIBRERÍA DE COMPONENTES UI.	
Elemento	**Descripción**
groupId	Prefijo para el paquete Java de la librería de componentes.
url	Namespace para la librería de componentes. Es el nombre o alias que se utiliza en las cabecera de las páginas para utilizar los componentes. Se usa en el TDL (Tag Definition Languaje).
version	Versión de la librería de componentes.
scope	Es usado para limitar la transitividad de una dependencia, afecta al classpath usado para varias tareas de construcción. El scope "Provided" indica que se espera que el JDK o el contenedor provean las dependencias en tiempo de ejecución.

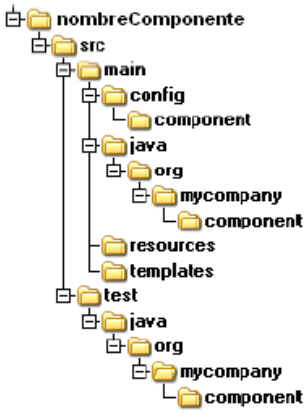
2.3.2. CREACIÓN DEL PROYECTO PARA EL DESARROLLO DEL COMPONENTE UI.

Para crear el proyecto para el componente UI se debe ingresar al directorio donde se encuentra su librería y ejecutar el siguiente comando.

mvn archetype:create –DarchetypeGroupId = org.richfaces.cdk –darchetypeArtifactId = mavenarchetype-jsf-component -DarchetypeVersion=3.3.0.GA -DartifactId=nombreComponente

El commando anterior crea una estructura de directorios como se muestra en la siguiente figura.

Figura 18. Estructura del proyecto para la creación de componentes UI.



Los directorios generados más importantes son:

src/main/config: Contiene los archivos meta-data, que son archivos xml necesarios para la configuración del componente.

src/main/java: Contiene el código java el pregenerado y el escrito por el desarrollador.

src/main/resources: Se usa para almacenar los recursos del componente, tales como, imágenes, código javascript, archivos css, etc. También contiene el archivo resource-config.xml para el registro de recursos.

src/main/templates: Contiene las plantillas jsp utilizadas para la generación del componente.

Se debe adicionar el plugin maven-compiler-plugin a las sección plugins en el archivo pom.xml generado en el directiorio del componente UI. Se debe adicionar el siguiente código:

```
<plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <inherited>true</inherited>
        <configuration>
```

```
            <source>1.5</source>
            <target>1.5</target>
        </configuration>
</plugin>
```

2.3.3. CREAR EL ESQUELETO DEL COMPONENTE UI.

Para crear el esqueleto del componente se debe ejecutar el siguiente comando en el directorio generado para el componente a desarrollar.

```
mvn cdk:create -Dname=nombreComponente
```

El resultado de este comando es la generación de tres archivos principales:

- Un archivo de configuración XML (nombreComponente.xml), para los metadatos del componente. Se genera en el directorio src/main/config**.**
- Una clase UI Generada en el directorio src/main/java/org/mycompany/component. Su nombre es UINombreComponente.java.
- Una plantilla jsp, generada en el directorio src/main/templates.

2.3.4. CREAR EL RENDERER.

El funcionamiento del componente se centra principalmente en dos acciones: codificar y decodificar datos. La decodificación (decode) es el proceso de conversión de los parámetros request a valores del componente; la codificación (encode) es el proceso de convertir los valores actuales del componente en las correspondientes etiquetas para ser mostradas en la página. En la siguiente figura se puede apreciar en el ciclo de vida de los componentes JSF y el momento cuando ocurren los procesos de codificación y decodificación.

Figura 19. Fases del ciclo de vida de un componente JSF[18].

Un componente JSF consta principalmente de dos partes: las clase UIClass del componente y el renderer. La clase UIClass es responsable del estado y ambiente del componente UI. La clase renderer se encarga de la representación del componente, genera las etiquetas apropiadas en el cliente y convierte la información suministrada por el cliente en valores del componente. Para un componente UI es necesario crear las siguientes clases:

- **NombreComponenteRenderer** en la cual se sobreescribe el método encode() para codificar etiquetas y recursos. Esta clase se genera al instalar el componente, se crea a partir del mecanismo de plantilas jspx creadas al generar el componente.
- **NombreComponenteRendererBase** se sobreescribe el método decode() y se define el convertidor para la clase. En esta clase se definen los método utilizados en la pagina jsp para la generación del componente.

Para la creación de la plantilla que genera la clase renderer se procede a modificar la página /src/main/templates/org/mycompany/ htmlNombreComponente.jspx. Se crea con etiquetas de HTML 4.0 y a partir de tags predefinidos, la referencia para los tag que se puedan utilizar en la plantilla se puede apreciar en el capítulo 11 del CDK developer guide. En la plantilla también se puede utilizar etiquetas html normales y código jsp tradicional. En el capítulo de desarrollo del componente reutilizable para la División de Servicios de Información se entrará más en detalle en la creación de esta plantilla, de la clase RendererBase y de los convertidores.

[18] Tomada de: CDK Developer Guide. Capitulo 4 página 16.

2.3.5. REGISTRAR RECURSOS.

Los recursos deben ser registrados en el archivo resource-config.xml. Se puede registrar imgenes, JavaScript , CSS, XCSS, SWF, (X)HTML, XML, Logs, JAR, etc), a continuación se presenta un ejemplo del registro de una imagen.

```
<resource>
        <name>org/mycompany/renderkit/html/images/icono.png</name>
        <path>org/mycompany/renderkit/html/images/icono.png</path>
</resource>
```

2.3.6. HEREDANDO DE UNA CLASE UI BASE.

La clase base para todos los componentes JSF es UIComponent, Al abrir el archivo generado en la creación del esqueleto del componente se puede apreciar que la clase UINombreComponente.java hereda de UIComponentBase, la cual es una sublcase de UIComponent y provee una implementación por defecto de todos los métodos abstractos de UIComponent.

2.3.7. CONFIGURACIÓN DEL COMPONENTE UI.

El archivo de configuración fue generado en el momento de crear el esqueleto del component UI, su nombre es nombreComponente.xml, se utiliza para la generación de la clase UINombreComponente completa, el Tag Handler para JSP, el archivo faces-config.xml y los descriptores para JSP y Facelets.

Consta principalmente de las siguientes partes:

- Mapeo a la clase UINombreComponente.

```
<name>org.mycompany.NombreComponente</name>
<family>org.mycompany.NombreComponente</family>
<classname>org.mycompany.component.html.HtmlNombreCompon</classname>
<superclass>org.mycompany.component.UINombreComponente</superclass>
```

- Mapeo de la clase Renderer y de la plantilla jspx.

```
<renderer generate="true" override="true">
<name>org.mycompany.NombreComponenteRenderer</name>
<template>org/mycompany/htmlNombreComponente.jspx</template>
</renderer>
```

- El tag handler para Jsp

```
<tag>
<name>NombreComponente</name>
<classname>org.mycompany.taglib.NombreComponenteTag</classname>
<superclass>
        org.ajax4jsf.webapp.taglib.HtmlComponentTagBase
</superclass>
</tag>
```

- Propiedades del componente.

```
<property>
        <name>value</name>
        <classname>java.lang.Object</classname>
        <description>The value of the component</description>
</property>
<property>
        <name>title</name>
        <classname>java.lang.String</classname>
```

```
        <description>Defines a title of the component</description>
        <defaultvalue>&amp;quot;NombreComponente&amp;quot;</defaultvalue>
    </property>
```

Si quiere incluir en el componente definición para eventos html, se debe adicionar las siguientes entidades en el archivo de configuración.

```
    &ui_component_attributes;
    &html_events;
    &ui_input_attributes;
```

2.3.8. INSTALACIÓN DEL COMPONENTE UI.

Para realizar la instalación del componente se procede a ejecutar el siguiente comando:

```
mvn clean install
```

El cual genera en el directorio /target/classes/META-INF los archivos *.tdl, *.taglib.xml, resoruces-config.xml y faces-config.xml.

La clase NombreComponenteTag se genera en directorio /target /clases /org /mycompany / taglib.

Con la generación de estos archivos y del jar nombreComponente-1.0-SNAPSHOT.jar se culmina la creación del componente, el archivo jar se puede encontrar en el directorio target.

2.3.9. USO DEL COMPONENTE UI

Para utilizar el componente en una página jsp se tiene que poner la siguiente cabecera:

```
    <%@ taglib uri="http://mycompany.org/NombreComponente" prefix="my"%>
```

Luego se utiliza la etiqueta del componente:

<my:inputDate **value**="#{bean.text}">

Se puede resaltar que la dirección http://mycompany.org es la url suministrada en el archivo de configuración pom.xml de la biblioteca de componentes creada al principio de este capítulo.

2.3.10. DISTRIBUCIÓN DEL COMPONENTE UI.

Para la distribución del componente se debe suministrar a los desarrolladores el archivo nombreComponente-1.0-SNAPSHOT.jar para que lo copien en la librería del proyecto donde se piensa utilizar y también se debe copiar en la librería del servidor de aplicaciones para ser reconocido en tiempo de ejecución.

3. COMPONENTE SOFTWARE REUTILIZABLE PROPUESTO

Como se mencionó en el planteamiento del problema en la sección 1.3. el componente software reutilizable que se desarrolla en la presente investigación consta de los siguientes elementos (ver figura 11):

- Entidades.
- Enterprise Java Beans (Controladores)
- Controles personalizados para la interfaz de usuario.
- Servicios web.

En primer lugar es importante analizar si el componente propuestos se puede considerar como un componente software reutilizable teniendo en cuenta las definiciones propuestas por Szyperski[19], Meyer[20], Sun Microsystem[21] e Iribarne Martínez[22], planteadas en el marco teórico de la investigación. En la tabla siguiente se puede apreciar la verificación y análisis del componente propuesto contra los requisitos que debe cumplir un elemento software para que se pueda clasificar como componente software reutilizable.

Tabla 2. Evaluación de requisitos para clasificar como componente.

EVALUACIÓN DE LOS REQUISITOS DE COMPONENTE		
REQUISITO	**EVALUACIÓN**	**ANÁLISIS**
1. Dependencia estructural: Alta cohesión y bajo acoplamiento.	OK.	Con la utilización del modelo de selección de componentes se logra que los componentes cumplan con este requisito, este modelo busca que los elementos tengan alta cohesión, es decir, que manejen solo la información que les corresponde y un bajo acoplamiento, lo cual determina que el componente no debe depender de otros componentes para que pueda funcionar.
2. Especificación de las funcionalidades.	OK.	El procedimiento de especificación del componente, planteado en una sección posterior, permite identificar y describir el componente, detallando las funcionalidades que presta, para poder realizar su

[19] SZYPERSKY, C. Component Software. Beyond Object-Oriented Programming. Addison-Wesley. 1998.
[20] MEYER, Bertrand. The Significance of Components. Beyond Objects column, Software Development. 1999.
[21] SUN MICROSYSTEM. Tutorial: Object-Oriented Analysis and Design Using UML. Copyright 2003.
[22] IRIBARNE MARTÍNEZ, Luis F. Un Modelo de Mediación para el Desarrollo de Software basado en Componentes COTS. Tesis Doctoral. Universidad de Málaga. España. 2003.

		búsqueda dentro de la biblioteca de componentes.
3. Seguir modelo de componentes.	OK.	Para el desarrollo del componente se sigue el modelo de componentes propuesto en Java EE 5 (EJB 3.0), el cual plantea las reglas para hacer las entidades y los Ejb, que son los subcomponentes claves para el desarrollo del componente propuesto.
4. Ensamblaje con otros componentes.	OK.	El componente propuesto se puede ensamblar de diferentes maneras, en varios tipos de aplicaciones informáticas, ya sea utilizando las entidades o los EJBs, en otros EJBs, como también se puede ensamblar los controles personalizados de la interfaz de usuario en cualquier aplicación web para interactuar con otros componentes de jsf, richfaces o html, y por último los servicios web que tiene el componente se pueden utilizar en otros lenguajes de forma remota.
5. Tener una interfaz.	OK.	Cada EJB tiene definida su propia interfaz que especifica las funcionalidades que maneja, con el conjunto de interfaces de los EJBs que forma el componente se tiene la especificación de sus servicios.
6. Estar certificado y probado.	OK.	Con el proceso de desarrollo de componentes propuesto se desarrollan componentes que cumplan con etapas específicas de prueba y certificación asegurando que el componente cumpla con las funcionalidades para lo que fue hecho.

Con el análisis de la tabla anterior se puede concluir que el componente planteado, desarrollado bajo el proceso de desarrollo de componentes propuesto se puede considerar como un componente software reutilizable porque cumple con todos los requisitos teóricos planteados.

La edición empresarial de Java versión 5 proporciona un conjunto de frameworks con los cuales es posible implementar el componente propuesto. Los subcomponentes son desarrollados siguiendo el modelo de componentes Java EE 5 [7]. En la siguiente tabla se muestran las principales librerías utilizadas para desarrollar el componente software reutilizable.

Tabla 3. Librerias Java EE 5 para implementar el componente.

LIBRERIAS JAVA EE 5 PARA IMPLEMENTAR EL COMPONENTE		
SUBCOMPONENTES	**LENGUAJES O LIBRERIAS**	**DESCRIPCIÓN**
Entidades	Api de persistencia de Java -	Contiene clases y anotaciones especiales para

	JPA 1.0	realizar el mapeo entre objetos y una base de datos relacional. Permite realizar la persistencia automática de las entidades.
	JPQL (*Java Persistence Query Language*, Lenguaje de consulta para persistencia en java.	Es un lenguaje de consulta parecido al SQL, que permite realizar consultas sobre las entidades.
	Jboss SEAM	Contiene clases y anotaciones para permitir el acople entre los diferentes niveles de una aplicación Java empresarial, es decir, permite la comunicación directa entre las entidades, EJBs y capa de presentación.
EJB	Api EJB 3.0	Clases que permiten la creación de la lógica del negocio de una aplicación, contiene servicios de seguridad, administración, transacción y configuración de los componentes en el contenedor.
	Jboss SEAM	Contiene clases y anotaciones para permitir el acople entre los diferentes niveles de una aplicación Java empresarial, es decir, permite la comunicación directa entre las entidades, EJBs y capa de presentación.
Controles UI	Api java server faces (JSF)	Contiene controles para ser utilizados en la interfaz de usuario, como: combos, menús, cuadros de textos, tablas, etc. Permite la validación de los datos introducidos por el usuario, manejo de eventos y conexión con la capa de presentación y la lógica de negocio.
	Api richfaces CDK – Kit de desarrollo de componentes.	Richfaces es una librería de componentes para la interfaz de usuario, parecido a los JSF, pero incorporan la tecnología AJAX en los componentes. También contiene un kit de desarrollo de componentes, el cual facilita la implementación de componentes richfaces.
	Apache Maven	Framework que permite automatizar las tareas relacionadas con el desarrollo de componentes o de cualquier aplicación. Las tareas que permite automatizar son: compilar, copiar, distribuir, limpiar, etc.
Servicios Web	Api JAX-WS	Conjunto de clases y herramientas necesarias para desarrollar, probar y distribuir servicios web y sus clientes, los cuales pueden ser basados o no en la plataforma java.

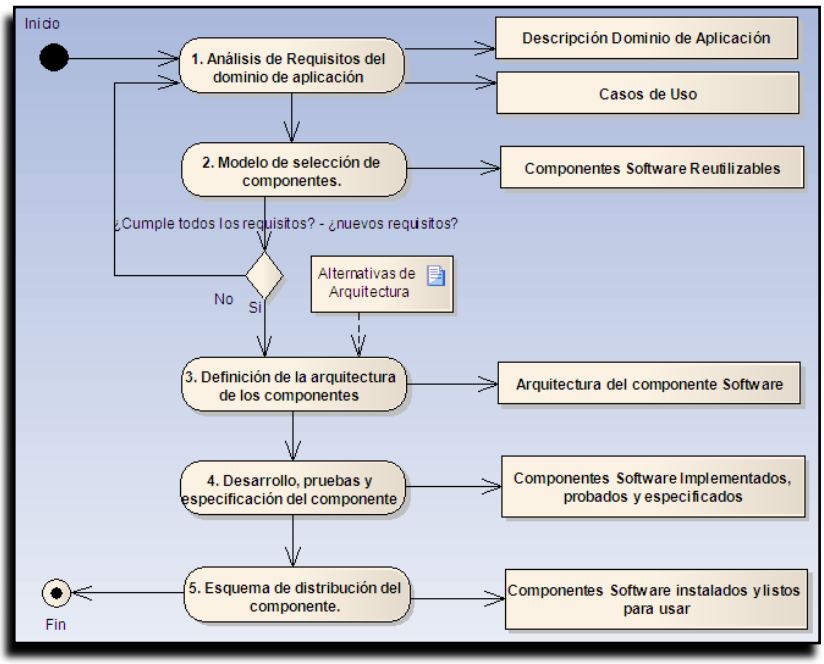
Se puede apreciar, para implementar las entidades se encuentra el API de persistencia de Java, que es el encargado de controlar la comunicación de las entidades con la base de datos; para desarrollar los Enterprise Beans se tiene el API EJB 3.0, que permite la

creación de los controladores de la lógica del negocio para la cual es creado el componente; para los componentes UI se encuentra el API Java Server Faces y el kit de desarrollo de componentes de Richfaces, y por último para exponer los servicios del componente como servicio web, java ofrece el API JAX-WS. Con este conjunto de frameworks que ofrece Java EE 5 es posible implementar el componente en estudio de la presente investigación. En los capítulos posteriores se podrá apreciar más en detalle la utilización de estos frameworks en la implementación que se hizo de un componente para la División de Servicios de Información.

4. PROCESO DE DESARROLLO DE COMPONENTES SOFTWARE REUTILIZABLES

Teniendo identificada la forma y el contenido del componente software reutilizable, se puede definir el planteamiento del proceso de desarrollo de componentes software reutilizables, el cual tiene su fundamento en Pressman[23], Iribarne Martínez[24], Weitzenfeld[25], Bruegge y Dutoit[26]. Las fases propuestas para el proceso de desarrollo son las siguientes: Análisis de requisitos del dominio de aplicación, Modelo de selección de componentes para el dominio de aplicación, Definición de la arquitectura, desarrollo y especificación del componente y por último el esquema de distribución del componente, la propuesta se puede apreciar mejor en la siguiente figura donde se encuentra el diagrama de actividades UML del proceso y se destacan los resultados principales de cada actividad.

Figura 20. Propuesta del proceso de desarrollo de componentes software reutilizables.



[23] PRESSMAN, Roger S. Ingeniería del Software un enfoque práctico. Quinta Edición. Mc Graw Hill. 2002.
[24] IRIBARNE MARTÍNEZ, Luis F. Un Modelo de Mediación para el Desarrollo de Software basado en Componentes COTS. Tesis Doctoral. Universidad de Málaga. España. 2003.
[25] WEITZENFELD, Alfredo. Ingeniería de Software Orientada a Objetos, Teoría y Práctica con UML y Java. México. Itam. 2002.
[26] BRUEGGE, Bernd y DUTOIT, Allen H. Object-Oriented Software Engineering Using UML, Patterns and Java. 2ª Edición. Prentice Hall. 2004.

EL proceso inicia con el análisis de requisitos del área de aplicación de los componentes a desarrollar, por medio del cual, se quiere describir en forma detallada el dominio de aplicación y obtener los diagramas de casos de uso de todas las funcionalidades que se pretende cubrir con los componentes. Después con el modelo de selección de componentes se quiere identificar y clasificar los componentes reutilizables a desarrollar en el dominio de aplicación seleccionado. Luego se debe establecer la arquitectura de cada componente, la cual pretende definir la estructura y organización de los elementos que conforman el componente, teniendo en cuenta las alternativas de arquitectura propuestas. Posteriormente se entra a la fase de desarrollo y especificación del componente cuyo propósito es la implementación de los componentes en el lenguaje seleccionado, pruebas y especificación, la especificación se realiza para poder identificar para qué sirve y qué hace cada componente. Por último se realiza el esquema de distribución del componente lo cual pretende definir la forma de instalar y utilizar los componentes implementados para que los desarrolladores de software los puedan utilizar.

A continuación se describen las fases del proceso de desarrollo de componentes planteado.

4.1. ANÁLISIS DE REQUISITOS DEL DOMINIO DE APLICACIÓN

En esta fase se pretende establecer el dominio de aplicación y el modelo de requisitos para la biblioteca de componentes que se pretende crear.

Para establecer el dominio de aplicación de la biblioteca de componentes es importante identificar y delimitar las áreas posibles de aplicación, estas áreas se definen dependiendo de la necesidad que la empresa de desarrollo de software tenga para implementar componentes y su grado posible de reutilización por los desarrolladores de software.

Por ejemplo para el caso de la División de Servicios de Información de la UIS, se pueden identificar las siguientes aéreas de aplicación: Sistema Financiero, Sistema de Recursos

Humanos, Sistema Académico, entre otras. Estas áreas son muy grandes, es necesario delimitarlas aún más para reducir su complejidad. El área del sistema de Recursos Humanos se puede dividir en las siguientes áreas o aplicaciones: Administración de personal, Administración hoja de vida del personal, Consultas generales[27], Actividad académica administrativa, Programas de desarrollo, entre otras. También es posible tener en cuenta áreas de aplicación comunes a toda la empresa de desarrollo como por ejemplo: Administración de roles, menús y usuarios, generador de listados, tramite de solicitudes.

Después de seleccionar las áreas posibles de aplicación para la cual se van a implementar los componentes software reutilizables es importante evaluar los siguientes criterios y poder decidir el dominio de aplicación de la biblioteca de componentes. Para cada criterio se define una escala y se da un puntaje de evaluación. Esta evaluación la debe realizar el arquitecto software de la empresa o el jefe, esta persona debe conocer la empresa y conocer las necesidades y prioridades que se tienen en la empresa en cuanto al desarrollo de software se refiere.

Tabla 4. Criterios de evaluación de las áreas de aplicación de la biblioteca de componentes.

CRITERIOS DE EVALUACIÓN DE LAS ÁREAS DE APLICACIÓN		
Criterio	**Descripción**	**Evaluación**
1. Grado de Utilización	Define el nivel de reutilización posible de los componentes del dominio de aplicación.	1: Muy bajo, 2: Bajo, 3: Medio, 4: Alto, 5: Muy alto.
2. Dificultad de implementación	Define la posible dificultad para el desarrollo e implementación de los componentes.	5: Muy baja, 4: Baja, 3: Media, 2: Alta, 1: Muy alta.
3. Complejidad	Define la complejidad del componente del área de aplicación de los componentes.	5: Muy baja, 4: Baja, 3: Media, 2: Alta, 1: Muy alta.
4. Cambiabilidad	Define la posibilidad de cambio de los requisitos para los cuales se implementan los componentes.	5: Muy baja, 4: Baja, 3: Media, 2: Alta, 1: Muy alta.
5. Nivel de Abstracción	Define el nivel de abstracción y de las funcionalidades que van a manejar los componentes.	1: Muy bajo, 2: Bajo, 3: Medio, 4: Alto, 5: Muy alto.
6. Necesidad de	Define el grado de necesidad para implementar	1: Muy bajo, 2: Bajo, 3: Medio, 4:

27. Las consultas generales son las consultas que utilizan todos los sistemas de información de la universidad por ejemplo: consultas de unidades académicas y administrativas, consultas de personal, consultas de cargos, consultas de tipos de documento, etc.

Implementación.	el componente en la empresa de desarrollo software.	Alto, 5: Muy alto.
7. Posibilidad de componentes ya desarrollados	Define la posibilidad de encontrar componentes ya desarrollados que se adapten a la mayoría de funcionalidades que va a manejar el área de aplicación.	5: Muy baja, 4: Baja, 3: Media, 2: Alta, 1: Muy alta.

Una vez evaluados los criterios se suman los puntajes dados y se define entre las áreas que más puntaje tienen cuál de ellas implementar. Es posible que las demás áreas se tengan que implementar también, entonces esta clasificación establece la prioridad de desarrollo de cada una. Después de seleccionar el área de aplicación se debe describir el dominio de aplicación seleccionado, resaltando las funcionalidades que se quieren cubrir con la biblioteca de componentes, está tarea se realiza por medio del modelo de requisitos, que se detalla a continuación.

4.1.1. MODELO DE REQUISITOS

Weitzenfeld dice que "el propósito del modelo de requisitos es comprender completamente el problema y sus implicaciones"[28]. El modelo de requisitos se centra principalmente en la información del dominio del problema y en identificar y establecer las funcionalidades que se van a ofrecer con los componentes. Con el modelo de requisitos se pretende:

- Tener una descripción del dominio de aplicación: Es una descripción preliminar de las funcionalidades y necesidades que debe cubrir la biblioteca de componentes a desarrollar.
- Identificar los actores y principales clientes de las funcionalidades del dominio de aplicación.
- Identificar los casos de uso del dominio de aplicación.
- Identificar las restricciones, precondiciones, postcondiciones y requerimientos no funcionales para los casos de uso.

[28] WEITZENFELD, Alfredo. Ingeniería de Software Orientada a Objetos, Teoría y Práctica con UML y Java. Capitulo 6, página 1.

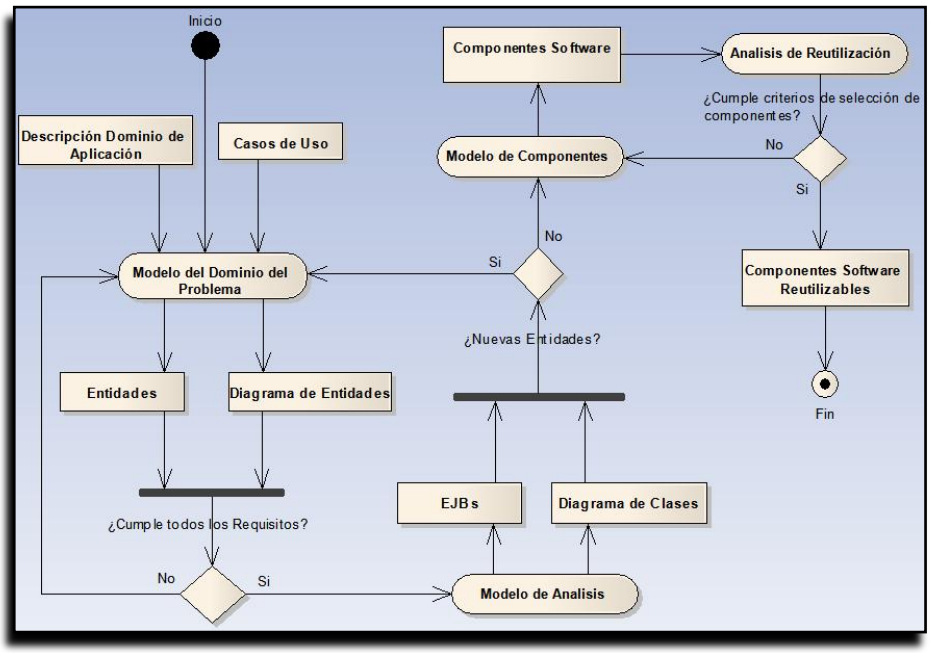
- Analizar las posibilidades de reutilización tanto a nivel de aplicaciones del mismo dominio, como a aplicaciones externas al dominio de aplicación.

Los resultados de este modelo són: la descripción detallada del dominio de aplicación y los diagramas de casos de uso[29].

4.2. MODELO DE SELECCIÓN DE COMPONENTES SOFTWARE REUTILIZABLES.

Con el modelo de selección de componentes se pretende elegir los componentes a implementar para el dominio de aplicación establecido, teniendo en cuenta los casos de uso definidos en la fase anterior.

Figura 21. Modelo de Selección de Componentes.



[29] Para mayor información sobre Modelos de caso de uso puede consultar: SUN MICROSYSTEM. Tutorial: Object-Oriented Analysis and Design Using UML; WEITZENFELD, Alfredo. Ingeniería de Software Orientada a Objetos, Teoría y Práctica con UML y Java; o RUMBAUCH, James; JACOBSON, Ivar y BOOCH, Grady. El lenguaje unificado de modelado. Manual de referencia. Addison Wesley.

El modelo de selección de componentes se puede basar en técnicas comunes de Ingeniería de software, más específicamente en el proceso de desarrollo de software orientado a objetos propuesto por Weitzenfeld[30], haciendo las adaptaciones correspondientes.

En la figura 21 se presenta el diagrama de actividades del modelo de selección de componentes propuesto, el cual consta de 4 actividades principales: modelo del dominio del problema, modelo de análisis, modelo de componentes y análisis de reutilización.

Con el modelo de selección de componentes se pretende identificar los componentes candidatos a implementar y establecer los criterios de selección de componentes para que sean reutilizables, para un dominio de aplicación establecido.

Teniendo la descripción del dominio de aplicación y los diagramas de caso de uso, el modelo de selección de componentes comienza con el modelo del dominio del problema cuyo objetivo principal es identificar las entidades o EJBs de entidad y sus relaciones. Se continúa con el modelo de análisis que pretende establecer los controladores o EJBs de sesión y el diagrama de las clases que manejan la lógica del negocio del área de aplicación. Posteriormente se continúa con el modelo de componentes que pretende encapsular las entidades y los EJBs en componentes, por último, se hace el análisis de reutilización por medio del cual se evalúan los criterios de selección de componentes para garantizar la reutilización de los componentes seleccionados.

A continuación se detallan las actividades que comprende el modelo de selección de componentes:

4.2.1. MODELO DEL DOMINIO DEL PROBLEMA.

Según Weitzenfeld, el modelo del dominio del problema define un modelo de clases común para todos los involucrados. El modelo de dominio se centra principalmente en la

[30] WEITZENFELD, Alfredo. Ingeniería de Software Orientada a Objetos, Teoría y Práctica con UML y Java. Capitulo 6, página 33

identificación de entidades. Una entidad es un objeto de dominio que tiene persistencia en la base de datos. Para identificar las entidades se hace uso de una técnica de ingeniería de textos así:

- Resaltar los sustantivos en la descripción del dominio de aplicación.
- Obtener un listado de entidades candidatas.
- Seleccionar las entidades que tengan significado para el dominio de aplicación que se está trabajando.
- Eliminar las entidades que correspondan aspectos de la interfaz de usuario y las que correspondan a actores del sistema.

Los resultados de este modelo son las entidades del dominio de aplicación y el diagrama de entidades.

El diagrama de entidades establece:

- Atributos y llaves primarias de las entidades.
- Las relaciones existentes entre las entidades.
- La multiplicidad y roles en las relaciones.
- La direccionalidad de las relaciones.

4.2.2. MODELO DE ANÁLISIS.

En este modelo se pretende establecer los Enterprise Java Bean (EJB) del dominio de aplicación. Los EJBs encapsulan la lógica del negocio de la aplicación y actúan como los controladores del componente. Para el modelo de análisis es importante tener en cuenta qué:

- Se debe definir por lo menos un EJB por caso de uso. Dependiendo de la complejidad del caso de uso se pueden utilizar varios EJB.
- Dependiendo de los requerimientos funcionales y no funcionales del caso de uso se decide el tipo de EJB a utilizar: con estado, sin estado o manejador de mensajes.

- Se pueden manejar varios tipos de EJB para un mismo caso de uso, depende de los requerimientos del caso de uso.
- Se deben encapsular los servicios del caso de uso.
- Se deben identificar las interfaces locales y remotas para cada EJB según el caso de uso.
- Se debe adicionar o eliminar las entidades que se estime conveniente.

Los resultados de este modelo son los EJBs y el diagrama de clases donde se relacionan los EJBs y las entidades que manejan; de este diagrama se puede apreciar la dependencia estructural de cada EJB.

Para mayor información puede consultar el libro EJB in Action.

4.2.3. MODELO DE COMPONENTES.

Se utiliza para definir los componentes software candidatos a implementar, se encarga de encapsular las entidades y los EJB dentro de un solo componente. Para realizar esta encapsulación se debe tener en cuenta qué:

- Se deben agrupar los EJB y entidades que manejen información similar y se puedan reutilizar.
- Si se encuentran relaciones de generalización, agregación o composición entre las entidades identificadas, estas entidades deben ir en el mismo componente.
- Las transacciones se deben manejar en lo posible dentro de un solo componente.
- Los EJBs o las entidades que son altamente acopladas deben ir en el mismo componente.

Los resultados de esta actividad son los componentes software candidatos a implementar y un diagrama de componentes que muestra la interacción entre ellos.

4.2.4. ANÁLISIS DE REUTILIZACIÓN.

En esta actividad se pretende asegurar que los componentes resultantes sean reutilizables, para esto es necesario definir unos criterios de selección de componentes. Bruegge y Dutoit [9] definen los objetivos de diseño en su proceso de desarrollo software, los cuales se pueden adaptar para definir los criterios de reutilización y de selección de los componentes. También Bertoa y Vallecillo [15] hacen una adaptación del modelo de calidad ISO 9126 aplicado a los componentes, de los cuales se pueden tomar algunos criterios propuestos por ellos para aplicarlos en los criterios de selección.

En la tabla 5 se pueden ver los criterios de selección de componentes, para cada criterio se presenta su descripción y una escala para realizar su evaluación. La evaluación se realiza con una escala de 1 – 5 indicando el grado de cumplimiento del criterio.

Tabla 5. Criterios de selección de Componentes.

CRITERIOS DE SELECCIÓN DE COMPONENTES		
Criterio	**Descripción**	**Evaluación**
Dependencia estructural.	Se determina por medio de la evaluación del grado de cohesión y el grado de acoplamiento del componente. Debe tener un alto grado de cohesión y un bajo acoplamiento para que la reutilización sea más fácil de obtener.	1- Muy alta. 2- Alta 3- Media 4- Baja 5- Muy baja
Facilidad Mantenimiento	El mantenimiento del componente debe ser transparente, debe afectar lo menos posible a los clientes que lo utilizan. Se mide el grado de dificultad del mantenimiento.	1- Muy Difícil 2- Difícil. 3- Media. 4- Fácil. 5- Muy fácil.
Grado de Adaptabilidad.	Determina la capacidad que tiene el componente para adaptar las funcionalidades o servicios que presta a situaciones imprevistas o no manejadas. El grado de adaptabilidad debe ser alto.	1-Muy Bajo. 2- Bajo. 3- Medio. 4- Alto. 5- Muy alto
Grado extensibilidad.	Determina el grado de dificultad y el impacto de añadir nuevas funcionalidades al componente. Para obtener la reutilización este grado de dificultad debe ser bajo.	1- Muy Alto. 2- Alto. 3- Medio. 4- Bajo. 5- Muy bajo

Grado de Utilización.	Determina el número de posibles clientes o usuarios del componente.	1-Muy Bajo. 2- Bajo. 3- Medio. 4- Alto. 5- Muy alto
Portabilidad.	Determina la capacidad que tiene el componente de utilizarse en diferentes ambientes al que fue implementado. Si se quiere que el componente tenga alta reutilización se debe poder utilizar en diferentes lenguajes, plataformas y sistemas operativos.	1-Muy Baja. 2- Baja. 3- Media. 4- Alta. 5- Muy alta.
Complejidad.	Determina el grado de complejidad de la lógica que maneja el componente. Entre más complejo sea, más difícil será su implementación.	1- Muy Alta. 2- Alta. 3- Media. 4- Baja. 5- Muy baja
Cambiabilidad.	Define la posibilidad de cambio de los requisitos para los cuales fue implementado el componente.	1- Muy Alta. 2- Alta. 3- Media. 4- Baja. 5- Muy baja
Posibilidad de componente ya desarrollado.	Define la posibilidad de encontrar un componente ya desarrollado que se adapte a la mayoría de las funcionalidades que va a manejar el componente a implementar.	1- Muy Alta. 2- Alta. 3- Media. 4- Baja. 5- Muy baja

Después de evaluar cada criterio de selección se suma los valores dados para obtener una clasificación con los componentes ordenados de forma descendente. Los componentes para ser candidatos a implementar deben tener una calificación mayor a 30 de 45 puntos.

Los resultados de esta actividad son los componentes software reutilizables listos para ser implementados.

4.3. ALTERNATIVAS DE ARQUITECTURA DE LOS COMPONENTES SOFTWARE REUTILIZABLES

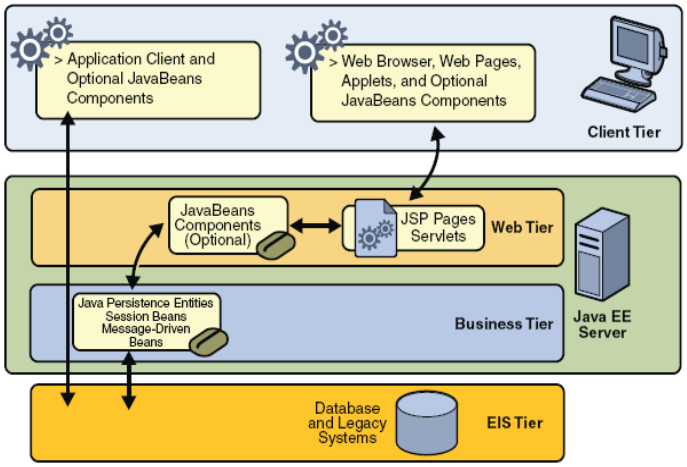
Una vez definidos y seleccionados los componentes, es necesario definir el estilo arquitectónico que pueden seguir para poderlos implementar correctamente.

Bruegge y Dutoit[31] definen el término arquitectura como la forma de organizar e interrelacionar los diversos componentes de un sistema informático. La arquitectura del sistema incluye descomposición del sistema, control del flujo global, manejo de condiciones de frontera y protocolos de intercomunicación de subsistemas.

Desde el punto de vista de la ISBC y para la presente investigación la arquitectura software de los componentes define la forma como se distribuyen e interrelacionan los elementos estructurales del componente (Entidades, EJBs, Componentes UI y Servicios) y define el medio de comunicación entre los componentes.

Los componentes software reutilizables que se pretenden desarrollar se deben basar en el modelo arquitectónico Java EE 5 y deben seguir el modelo de componentes Java EE 5. La arquitectura básica y general de las aplicaciones empresariales se puede apreciar en la figura 22

Figura 22. Arquitectura Aplicaciones Java EE 5[7].



[31] BRUEGGE, Bernd y DUTOIT, Allen H. Object-Oriented Software Engineering Using UML, Patterns and Java. 2ª Edición. Prentice Hall. 2004.

Las aplicaciones Java EE 5 constan de tres capas principales: La capa cliente, la capa del servidor Java EE, que se divide en dos: la capa web y de lógica del negocio, y la capa de información empresarial.

La ingeniería de software define diferentes estilos arquitectónicos a utilizar en el desarrollo de aplicaciones software, del análisis de los estilos arquitectónicos y de la arquitectura de Java EE 5, se pueden definir las siguientes alternativas de arquitectura para los componentes software reutilizables:

- Arquitectura modelo – vista – controlador.
- Arquitectura por capas.
- Arquitectura orientada a servicios (SOA).

En la siguiente tabla se muestran las principales características de los estilos arquitectónicos resaltando sus ventajas, desventajas y cuando utilizar cada uno[32].

Tabla 6. Comparación de las Alternativas de arquitectura de los componentes.

COMPARACIÓN DE LAS ALTERNATIVAS DE ARQUITECTURA DE LOS COMPONENTES		
CARACTERÍSTICAS PRINCIPALES		
M.V.C.	**CAPAS**	**SOA**
Modelo: Administra el comportamiento y los datos del dominio de aplicación. **Vista:** Maneja la visualización de la información. **Controlador:** Interpreta las acciones sobre la vista, informando al modelo y/o a la vista para que cambien según resulte apropiado. La vista y el controlador dependen del	Se puede definir la arquitectura en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa superior y se sirve de las prestaciones que le brinda la capa inferior. Existe en teoría un bajo acoplamiento entre las capas.	Un servicio es una entidad de software que encapsula funcionalidad de negocios y proporciona dicha funcionalidad a otras entidades a través de interfaces públicas bien definidas. La arquitectura orientada a servicios pretende la exposición y uso de servicio por parte de los sistemas informáticos.

[32] GUIJUN, Wang; CASEY K, Fung. Artículo: Architecture Paradigms and Their Influences and Impacts on Component-Based Software Systems. Proceedings of the 37th Hawaii International Conference on System Sciences – 2004

modelo, el modelo no depende de otros componentes. Esto permite construir y probar el modelo independiente de la representación visual. Mantiene un acoplamiento entre la vista y el modelo, y el controlador y el modelo.	Un ejemplo típico de una arquitectura por capas es el modelo OSI.	Esta arquitectura se fundamenta principalmente en los servicios web. Un servicio web es un sistema de software diseñado para soportar interacción máquina – a - máquina sobre una red. Posee una interfaz descrita en un formato procesable por máquina (específicamente WSDL). Otros sistemas interactúan con el Web service de una manera prescripta por su descripción utilizando mensajes SOAP, típicamente transportados usando HTTP con una serialización en XML.
VENTAJAS		
M.V.C.	**CAPAS**	**SOA**
- Soporte de vistas múltiples, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente y en diferentes dispositivos. - Adaptación al cambio, dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo. - Bajo acoplamiento: desacopla las vistas de los modelos y desacopla los modelos de la forma en que se muestran e ingresan los datos. - Alta cohesión: Cada elemento del patrón está altamente especializado en su tarea. - Claridad en el diseño - Alta escalabilidad. - Se puede presentar como una adaptación de la arquitectura por capas.	- Soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. - Admite muy naturalmente optimizaciones y refinamientos. - Proporciona amplia reutilización, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces. - Facilita la migración. El acoplamiento con el entorno está localizado en las capas inferiores. - Permite trabajar en varios niveles de abstracción. Para implementar los niveles superiores no necesitamos conocer en entorno subyacente, solo las interfaces que proporcionan los niveles inferiores.	- Se puede reemplazar un servicio sin tener que preocuparse por la tecnología fundamental; la interface es lo que importa, y está definida en un estándar universal en servicios Web y XML. - Es una arquitectura distribuida, la carga de las transacciones se pueden dividir en diferentes servidores comunicándose todos por medio del los servicios web. - Los elementos de esta alternativa están débilmente acoplados. - El servicio puede recibir requerimientos de cualquier origen. - La funcionalidad del servicio se puede ampliar o modificar sin rendir cuentas a quienes lo requieran. - Los componentes que requieran un servicio pueden descubrirlo y utilizarlo dinámicamente. - Alta interoperabilidad con otras plataformas.
DESVENTAJAS		
M.V.C.	**CAPAS**	**SOA**
- Complejidad, esta alternativa	- Muchos problemas no admiten	- La velocidad de intercambio de

introduce nuevos niveles de indirección y por lo tanto aumenta ligeramente la complejidad. También se profundiza la orientación a eventos del código de la interfaz de usuario, que puede llegar a ser difícil de depurar. - Costo de actualizaciones frecuentes. Si el modelo cambia frecuentemente puede desencadenar muchos requerimientos de actualización en las vistas que utilizan el modelo.	un buen mapeo en una estructura jerárquica. Incluso cuando un sistema se puede establecer lógicamente en capas, consideraciones de rendimiento pueden requerir acoplamientos específicos entre capas de alto y bajo nivel. A veces es también extremadamente difícil encontrar el nivel de abstracción correcto. - Al establecer el llamado solo de capas superiores a capas inferiores es posible que el rendimiento no sea bueno.	información es más lenta que una conexión directa entre los elementos. - Su implantación es compleja y se requiere de grandes esfuerzos en su desarrollo. - Los clientes necesitan saber las operaciones y su semántica antes del uso.
CUÁNDO UTILIZAR CADA ALTERNATIVA		
M.V.C.	**CAPAS**	**SOA**
- Se recomienda utilizar este estilo de arquitectura cuando se necesitan muchas vistas o presentaciones del mismo modelo. - También es recomendada utilizar esta alternativa cuando se necesiten elaborar interfaces de usuario complejas y para diferente clase de dispositivos. - Este tipo de arquitectura es el más usado en aplicaciones web.	- Se puede utilizar la arquitectura por capas, cuando se pude estratificar los elementos que conforman el componente en diferentes categorías y no exista un marcado acoplamiento entre estas categorías. - Es útil en sistemas complejos, que manejen muchas transacciones y que puedan cambiar, al dividir por capas es más fácil cambiar una capa sin afectar notablemente las otras. - La arquitectura propuesta por Java para aplicaciones empresariales es por capas, implementa las siguientes capas: nivel de persistencia, nivel de servicios, nivel de aplicación (Web) y el nivel de presentación.	- Se recomienda utilizarla cuando se necesite distribuir los componentes en diferentes maquinas y que funcionen en diferentes plataformas a las cuales fueron implementados. - Es ideal utilizar este estilo arquitectónico cuando el sistema o componente es basado en mensajes, en especial mensajes asíncronos. - Para sistemas multiplataforma es ideal utilizar esta alternativa, debido a que los servicios web se basan en un estándar XML unificado.

Es importante resaltar que en muchos casos, la mejor solución para determinar que arquitectura utilizar en el desarrollo de un sistema informático o un componente es un hibrido de las tres arquitecturas propuestas tomando lo mejor de cada una y haciendo los ajustes necesarios.

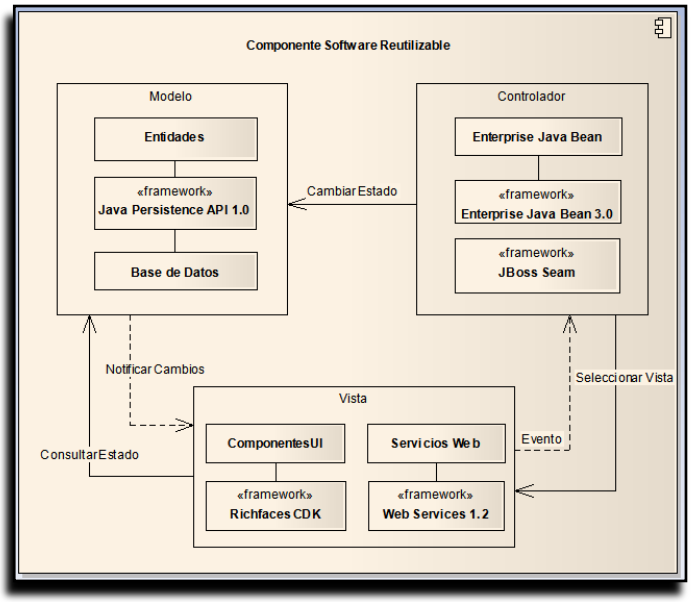
A continuación se presenta la adaptación correspondiente de los estilos de arquitectura planteados para los componentes software reutilizables propuestos en la presente investigación.

4.3.1. ARQUITECTURA MODELO–VISTA–CONTROLADOR.

Franky[33] describe esta arquitectura de la siguiente manera: ”El Modelo, maneja las reglas del negocio y las estructuras de datos; la vista, maneja presentación de los datos del sistema al usuario; y el controlador: Transforma pedidos del usuario en operaciones sobre los objetos del modelo y selecciona la vista para mostrar los resultados al usuario”. La adaptación de este estilo arquitectónico para los componentes software reutilizables se puede apreciar en la figura 23.

En una parte del modelo estarían las entidades soportadas por el API de persistencia de Java que se encarga de mapear las entidades en java con la base de datos relacional, también se utiliza el framework Jboss Seam.

Figura 23. Arquitectura Modelo – Vista – Controlador



[33] FRANKY, María Consuelo. Curso: Desarrollo de aplicaciones en Java EE 5. CincoSOFT LTDA. 2007.

En la vista se ubican los componentes o controles para la interfaz de usuario, desarrollados por medio del kit de desarrollo de componentes de Richfaces y los servicios web, que exponen las funcionalidades del componente como servicios web, lo cual permite utilizar al componente de forma remota y por otras aplicaciones así no sean elaboradas en java.

El control seria manejado por los EJBs soportados por los frameworks Enterprise Java Beans 3.0 y Jboss Seam. Los controladores son los encargados de manejar la lógica del negocio del componente, se encargan de procesar la información suministrada por los clientes y presentar los resultados.

4.3.2. ARQUITECTURA POR CAPAS.

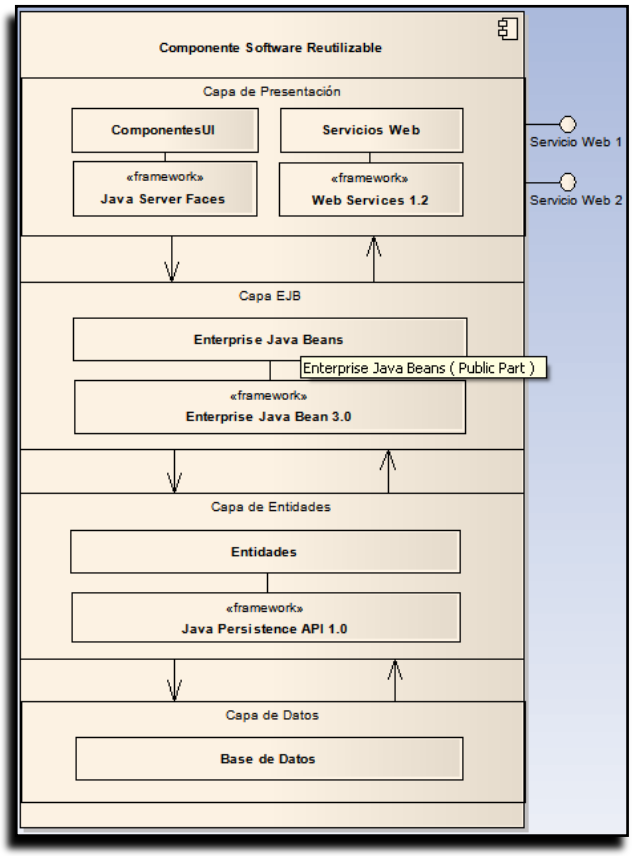
Este tipo de arquitectura pretende dividir los elementos del software en capas que se comunican entre sí hasta entregar el resultado al usuario. Cada capa solo se puede comunicar con las capas subyacentes.

Para los componentes de estudio, se definieron las siguientes capas: capa de datos, capa de entidades, capa de EJBs y la capa de presentación. Este estilo de arquitectura se puede apreciar mejor en la figura 24.

Capa de Datos: Esta capa se encarga de la conectividad con la base de datos, define el modelo físico de datos, el cual va estar ligado a las entidades.

Capa de Entidades: Definen las clases de entidad que van a tener persistencia en la base de datos, Esta capa está soportada por el Api de persistencia de java y el JBoss Seam, que se encargan de mapear las entidades en java con las tablas de la base de datos permitiendo la persistencia automática.

Figura 24. Arquitectura por Capas



Capa de EJBs: Son los elementos que establecen la lógica del negocio del área de aplicación, definen los servicios y funcionalidades que va a manejar el componente.

Capa de Presentación: Es la encargada de utilizar los servicios propuestos por los EJB para exportarlos a clientes como controles de usuario personalizados para el área de aplicación o como servicios web para poder utilizarlos remotamente o por cualquier otro sistema.

4.3.3. ARQUITECTURA ORIENTADA A SERVICIOS.

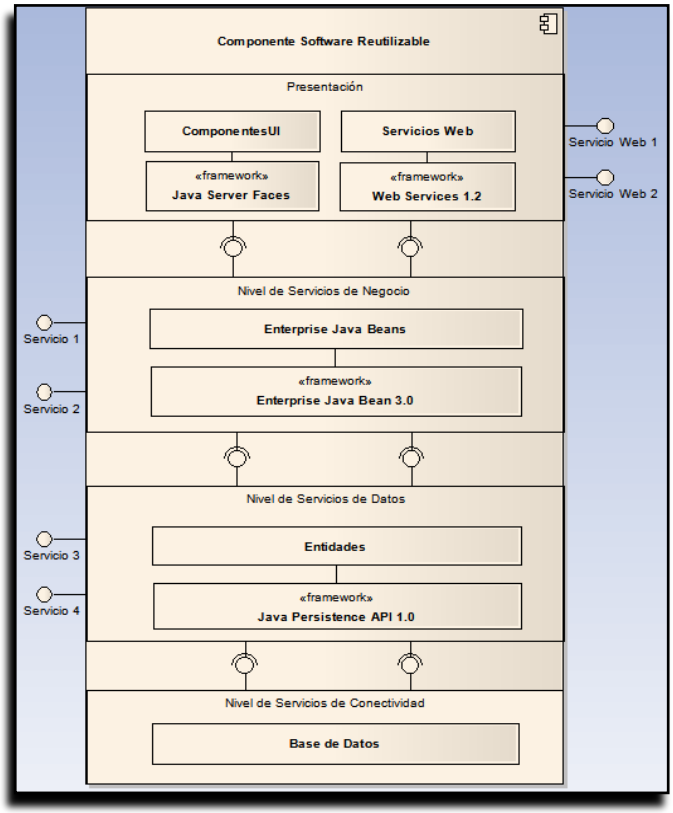
Según Toro[34], la arquitectura orientada a servicios consiste en la exposición y uso de servicios por parte de los sistemas informáticos. Los sistemas informáticos se dividen en

[34] TORO, Víctor Manuel. Conferencia: Panorama sobre la Ingeniería del Software. CincoSOFT LTDA. 2007.

varios niveles de servicios: servicios de conectividad con otros sistemas, servicios de datos, servicios de negocio, procesos de negocio y presentación de servicios para ser consumidos por diferentes sistemas y dispositivos.

En la figura 25 se aprecia la aplicación de la arquitectura orientada a servicios en los componentes software reutilizables.

Figura 25. Arquitectura orientada a servicios.



En este estilo de arquitectura se dividen los elementos estructurales del componente en niveles de servicios que son exportados y usados por el nivel superior. Los niveles de servicios del componente son:

Nivel de Servicios de Conectividad: Expone los mecanismos y servicios de conectividad con cualquier base de datos.

Nivel de Servicios de Datos: En este nivel se encuentran las entidades soportadas por el API de persistencia de Java. En este nivel se encuentran los servicios de creación, modificación, eliminación y control de los datos.

Nivel de Servicios de Negocio: se ofrecen los servicios de la lógica del negocio del dominio de aplicación de los componentes; se encarga de manejar las transacciones y procesar las peticiones de los clientes.

Nivel de Presentación: En este nivel se encuentran los componentes para la interfaz de usuario y la exportación de los servicios ofrecidos por los EJBs como servicios Web. Estos elementos usan los servicios encapsulados en los EJBs del nivel de servicios de negocio o los servicios de datos propuestos por las entidades.

Cada nivel de servicios puede exponer al desarrollador directamente los servicios que implementa, el desarrollador puede directamente acceder a los servicios ofrecidos por las entidades o utilizar los servicios de los EJBs directamente.

4.4. PROCEDIMIENTO PARA EL DESARROLLO, PRUEBAS y ESPECIFICACIÓN DE LOS COMPONENTES

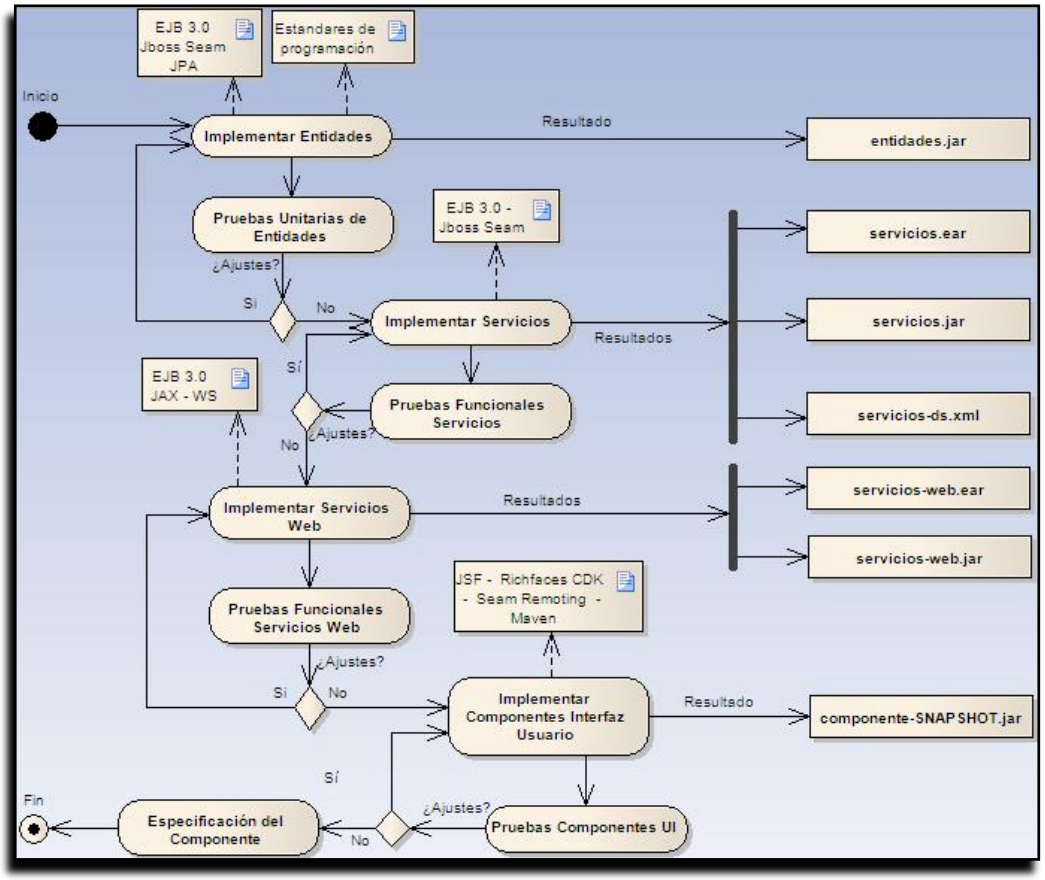
Con el procedimiento de desarrollo, pruebas y especificación se pretende definir los pasos necesarios para implementar los componentes software reutilizables garantizando su calidad, como también describir detalladamente las funcionalidades, propiedades y elementos del componente por medio de su especificación. En la figura se puede apreciar el procedimiento propuesto, el cual consta de 9 actividades principales las cuales se detallan más adelante.

Antes de iniciar el modelo se deben tener bien definidos los estándares de desarrollo de software para ser utilizados en la fase de desarrollo, pruebas e implementación del

componente software reutilizable. Los estándares de desarrollo básicamente deben contener la definición de estándares de nombres de variables, atributos y métodos, buenas prácticas y estándares de codificación. Para la presente investigación se utilizaron los estándares de nombres y programación definidos por la División de Servicios de Información para el desarrollo de los sistemas empresariales.

También es muy importante tener definidas las herramientas que se van a utilizar para la codificación, prueba y distribución del componente cómo son: servidor de aplicaciones, controlador de versiones, IDE de desarrollo (eclipse, netbeans, jcreator, etc), herramientas para compilación, distribución y pruebas.

Figura 26. Procedimiento de Desarrollo, Prueba y Especificación de componentes.



El modelo comienza con la implementación de las entidades, actividad que pretende realizar el mapeo objeto – relacional entre las clases java y la base de datos relacional, continua con las pruebas unitarias de este mapeo, con las cuales se pretende garantizar que el mapeo quedó bien realizado. Luego se realiza la implementación de los servicios que va a ofrecer el componente por medio del los frameworks EJB 3.0 y JBoss Seam, después se realizan la pruebas funcionales de estos servicios para verificar que quedaron bien implementados. Se continúa con el desarrollo de los servicios web, con los cuales se garantiza la interoperabilidad del componente en diferentes plataformas y lenguajes, luego se realizan las pruebas respectivas a los servicios web. Posteriormente se implementan los controles personalizados para la interfaz de usuario, los cuales permiten encapsular y reutilizar la lógica del negocio del componente en diferentes aplicaciones web. Por último se realiza la especificación del todo el conjunto de funcionalidades que ofrece el componente. Cada actividad se define con más detalle a continuación.

4.4.1. IMPLEMENTACIÓN DE ENTIDADES (EJB DE ENTIDAD).

El objetivo principal de esta fase es convertir el modelo de entidades o modelo del dominio del problema en las clases de entidad implementadas y mapeadas con la respectiva base de datos relacional. El mapeo se realiza en Java versión empresarial 5, utilizando el api de persistencia de Java, el JBoss Seam y el framework EJB 3.0. En la figura siguiente se presenta el ejemplo de la implementación de la entidad ClaseUnidad[35] en la cual se puede apreciar las principales anotaciones que se utilizan para definir una entidad y realizar su mapeo con la base de datos.

[35] La entidad ClasesUnidad hace referencia a las clases de unidad académico administrativas de la universidad, como son: división, sección, facultad, escuela, entre otras.

Figura 27. Implementación de la entidad: ClaseUnidad.

```
/**
 * Entidad que representa las clases de unidad de la universidad.
 * Su tabla de persistencia es:clases_unidad
 * @author FreVe
 * @version 1.0
 */
@Entity
@Table(name="recursos_humanos:clases_unidad")  (1)
@Name("ClaseUnidad")
public class ClaseUnidad implements Serializable{

    private static final long serialVersionUID = 1L;

    @Id
    @Column(name="codigo_clase_uni")  (2)
    private Integer codigo;

    @Column(name="descrip_clase_uni", length=30)  (3)
    private String descripcion;

    @OneToMany(mappedBy="claseUnidad", fetch=FetchType.LAZY)  (4)
    @OrderBy("nombre")
    private List<Unidad> unidades;

    public ClaseUnidad(){
        super();
        codigo=0;
        descripcion="";
    }
```

En (1) se define la clase como entidad con la anotación *@Entity*, con la anotación *@Table* se mapea la clase con la tabla *clases_unidad* en la base de datos recursos_humanos, y la anotación *@Name* de seam se le da un nombre a la entidad para ser conocida en el contenedor de EJBs y en el servidor de aplicaciones.

En (2) se define la llave primaria de la entidad con la anotación *@Id* y se mapea el atributo de la clase *codigo* con el campo *codigo_clase_uni* de la tabla.

En (3) se mapea el atributo *descripcion* al campo *descrip_clase_uni* y se le define una longitud de 30 caracteres.

En (4) aparece un listado que representa una relación de uno a muchos con otra entidad, se define por medio de la anotación *@OneToMany* con los atributos *mappedBy* y *fetch* que indican el atributo de la entidad con el cual se relaciona y el modo de carga del listado que en este caso es *Lazy* (peresoso) esto quiere decir que no realiza la consulta del listado cuando se instancia la entidad sino cuando se hace el llamado al método *get* correspondiente al atributo; la anotación *@Orderby* ordena el listado por el nombre de la unidad.

El anterior es un ejemplo básico de cómo implementar una entidad, la cual consta principalmente de los elementos mencionados con anterioridad: anotaciones de definición de la entidad en la cabecera de la clase, atributos mapeados a los campos de la tabla que representan, llave primaria y relaciones con otras entidades, en ocasiones la llave primaria es una llave compuesta por varios campos, para mapear esta llave primaria se debe definir otra clase, la cual debe ir como atributo de la entidad que estamos mapeando y se debe utilizar la anotación *@EmbeddedId* para indicar la llave primaria. También utilizando anotaciones es posible incluir validaciones de datos, de rangos, de formatos, entre otros. En la tabla siguiente se presentan las anotaciones principales para mapear una entidad. En el libro EJB in Action, capitulo 7 página 217, se encuentra una guía detallada de cómo mapear las entidades y como implementar el modelo del dominio del problema en Java versión empresarial.

Tabla 7. Anotaciones principales para el mapeo de las Entidades.

ANOTACIONES PARA EL MAPEO DE ENTIDADES		
ANOTACIONES DE DEFINICIÓN DE ENTIDADES		
Anotación	**Clase**	**Descripción**
@Entity	javax.persistence.Entity	Define una clase o pojo como entidad.
@Embeddable	javax.persistence.Embeddable	Define una clase como embebida, es decir, se puede utilizar como atributo de otra entidad.
@Embedded	javax.persistence.Embedded	Especifica que un campo de persistencia de una entidad o una propiedad es una clase embebida.
@Id	javax.persistence.Id	Denota a un campo de persistencia o propiedad como identificador único para la entidad.
@EmbeddeId	javax.persistence.EmbeddedId	Denota un objeto embebido como llave primaria de la entidad.
ANOTACIONES DE DEFINICIÓN DE DATOS DE LAS ENTIDADES		
Anotación	**Clase**	**Descripción**
@Transient	javax.persistence.Transient	Marca un campo o propiedad como no persistente, es decir cuando se registre en la base de datos este campo no se guarda y no está mapeado a ningún campo de la tabla

@Temporal(TemporalType.*TIPO*)	javax.persistence.Temporal	Especifica el tipo del atributo como un java.util.Date o java.util.Calendar, se utiliza para definir un campo como fecha corta, o tipo Time. Los valores definidos en la enumeración TemporalType son DATE, TIME y TIMESTAMP para hacer corresponder con las java.sql.Date, java.sql.Time y java.sql. Timestamp.
@ Enumerated	javax.persistence.Enumerated	Denota un atributo de la clase como tipo *enumerated*.
ANOTACIONES PARA EL MAPEO DE DATOS DE LAS ENTIDADES		
Anotación	**Clase**	**Descripción**
@Table(name = "bd : tabla", catalog, schema, UniqueConstraints())	javax.persistence.Table	Define la tabla principal a la cual una entidad es mapeada.
@ SecondaryTable (name, catalog, schema, pkJoinColumns(), uniqueConstraints())	javax.persistence.SecondaryTable	Define una tabla secundaria a la cual es mapeada la entidad.
@Column(name="nombre_columna", length=30)	javax.persistence.Column	Define el mapeo entre una columna de la tabla con un atributo o propiedad de la entidad. Las propiedades de la anotación son: name, unique, nullable, insertable, updatable, columnDefinition, table, length, precision, scale.
@GeneratedValue (GenerationType.Tipo)	javax.persistence.GeneratedValue	Se usa para la generación automática de valores. Es usado típicamente en las llaves primarias. GenerationType puede ser: TABLE, SEQUENCE, IDENTITY, AUTO.
ANOTACIONES PARA LAS RELACIONES ENTRE LAS ENTIDADES		
Anotación	**Clase**	**Descripción**
@OneToOne	javax.persistence.OneToOne	Define la relación de Uno a Uno, que significa que una entidad se relaciona con solo una instancia de otra entidad.
@ManyToOne(fetch = FetchType.*LAZY*)	javax.persistence.ManyToOne	Define la relación de Muchos a Uno, en la cual muchas instancias de una entidad se relación con una sola instancia de otra entidad. Define el lado dueño de la relación.
@OneToMany (mappedBy = “campo", fetch = FetchType.*LAZY*)	javax.persistence.OneToMany	Define la relación de Uno a Muchos, en la cual una sola instancia de una entidad se relaciona con muchas instancias de otra entidad. Define el lado inverso de la relación.
@ ManyToMany(fetch =	javax.persistence.ManyToMany	Establece la relación de Muchos a

FetchType.LAZY) – Lado Dueño @ManyToMany (mappedBy = “campo", fetch = FetchType.LAZY) – Lado inverso de la relación.		Muchos, por medio de la cual una instancia de una entidad se relaciona con muchas instancias de la otra entidad y una instancia de la otra entidad se relaciona con muchas instancias de esta entidad. Cualquier lado de la relación puede ser el dueño o el inverso de la relación.
@JoinColumn(name = "columna", referencedColumnName = "columna relacion", nullable = **true**)	javax.persistence.JoinColumn	Denota el nombre de la columna de la tabla a la cual es mapeada la entidad, y con la cual se establece la relación. En referencedColumnName va el nombre de la columna que se referencia de la tabla con la cual se relaciona la entidad.
@JoinColumns({ @JoinColumn(name = “columna ", referencedColumnName = " columna "), @JoinColumn(name = " columna ", referencedColumnName = " columna ") })	javax.persistence.JoinColumns	Define el mapeo de la columna con la entidad por medio de llaves compuestas.
@OrderBy("atributo_clase")	javax.persistence.OrderBy	Define el campo por el cual se orden un listado establecido por las relaciones Uno a Mucho y Muchos a Muchos.

Es importante reescribir los métodos hashcode(), equals() y toString de cada entidad. Para los métodos hascode() y equals() se deben utilizar los atributos propios de la entidad sin tener en cuenta los atributos de las relaciones porque podrían causar una referencia circular y causar una sobrecarga del servidor.

Para la manipulación de las entidades se utiliza el JPQL, el cual es un lenguaje de consulta estructurado, es similar al SQL pero se opera sobre los objetos[36].

Una vez implementadas las entidades es necesario crear el archivo de configuración *orm.xml*, donde se deben declarar las entidades para ser interpretadas por el servidor de aplicaciones. El framework Hibernate tiene herramientas muy útiles que permiten la

[36] Para obtener una referencia completa de utilización de este lenguaje se recomienda estudiar el libro EJB in Action, capitulo 10 página 340.

creación automática de este archivo y la conversión de una base de datos relacional a las entidades mapeadas.

El resultado de esta fase es un archivo .jar que contiene el conjunto de clases de entidad y los archivos necesarios de configuración para ser distribuido en el servidor de aplicaciones.

4.4.2. PRUEBAS UNITARIAS DEL MAPEO DE ENTIDADES.

Teniendo implementadas y mapeadas las entidades es necesario realizar pruebas unitarias a cada entidad para garantizar que el mapeo quedó realizado correctamente y garantizar también la calidad de los objetos del dominio del problema a utilizar en el desarrollo de todo el componente. Para la realización de las pruebas unitarias se puede utilizar el api java llamado JUnit que automatiza las pruebas generando las clases específicas de prueba y los métodos para su ejecución. En la tabla siguiente se detalla una guía para la realización de las pruebas de unidad independientemente si se utiliza el JUnit. En la tabla se destacan las pruebas principales a realizar, el objetivo de cada prueba y el resultado esperado.

Antes de iniciar las pruebas planteadas es necesario introducir datos de prueba a las tablas de la base de datos relacional sobre la cual se realiza el mapeo. Las sentencias JPQL y los métodos necesarios para realizar las pruebas se pueden implementar utilizando el JUnit o realizando una aplicación web de prueba, en la cual se crea una página xhtml, un EJB y se hacen los métodos respectivos para cada prueba. Con las herramientas del framework JBoss Seam es fácil crear una aplicación web de prueba.

Tabla 8. Pruebas unitarias del mapeo de las entidades.

PRUEBAS UNITARIAS DEL MAPEO DE LAS ENTIDADES		
Descripción Prueba	**Objetivo**	**Resultado Esperado**
Probar mapeo de la entidad con la tabla asociada: Se debe crear una sentencia simple JPQL que consulte los datos de la entidad. Ejemplo:	Verificar que el mapeo de la entidad con la tabla asociada fue hecho correctamente.	Al ejecutar la consulta JPQL se debe obtener como resultado los registros de prueba introducidos. Si se presentó algún error en el mapeo no

SELECT c FROM ClaseUnidad.		se devuelve ningún registro.
Probar las relaciones de la entidad con otras entidades: Crear un método que permita realizar la prueba de las relaciones mapeadas en la entidad. Ejemplo: La entidad ClaseUnidad tiene mapeada una relación uno a muchos con la entidad Unidad. El código de prueba seria: EntityManager em; ClaseUnidad clase = em.merge(c); List<Unidad> u = clase.getUnidades(); System.out,println(u); Donde c es la clase de unidad que se recibe como parámetro en el método.	Verificar que el mapeo de las relaciones de entidades se realizó correctamente.	Al ejecutar el método se deben obtener los registros de las entidades relacionadas. Si se presenta error en el mapeo no se obtiene ningún registro o se evidencia alguna excepción.

Probar la inserción de un registro en la tabla de persistencia asociada: Hacer un método que implemente un persist sobre la entidad. Ejemplo:
 	
EntityManager em;
ClaseUnidad c = new ClaseUnidad();
c.setCodigo(1);
c.setDescripcion(“Decanatura”);
em.persist(c);
	Verificar que la inserción de datos o la persistencia automática funcionan correctamente.	Al ejecutar la prueba el registro quedó insertado en la base de datos.

Probar la eliminación de un registro de la tabla de persistencia: Elaborar un método que implemente la eliminación de un registro. Ejemplo: EntityManager em; em.remove(entityManager.merge(c)); Donde c es la Clase de unidad recibida como atributo en el método.	Verificar que la eliminación de datos de la tabla de persistencia se realiza correctamente.	Al ejecutar la prueba el registro fue eliminado de la base de datos.
Probar la actualización de un registro de la tabla de persistencia: Elaborar un método	Verificar que la actualización de datos en la tabla de persistencia se realiza correctamente.	Al ejecutar la prueba el registro es actualizado en la base de datos.

que implemente la actualización de un registro. Ejemplo:
 	
EntityManager em;
ClaseUnidad c = new ClaseUnidad();
c.setCodigo(1);
c.setDescripcion("Escuela");
em.merge(c);		

Una vez finalizadas y aprobadas las pruebas de todas las entidades se puede pasar a la implementación de los servicios que va a prestar el componente. Es importante resaltar que al realizar cambios sobre las entidades es necesario realizar o correr las pruebas descritas en la tabla anterior nuevamente.

4.4.3. IMPLEMENTACIÓN DE SERVICIOS (EJB DE SESIÓN).

En esta fase se pretende crear los servicios que va prestar el componente a sus clientes, en primer lugar es importante aclarar la definición de servicio, según Wang y Fung[37] "un servicio es una unidad software que encapsula funcionalidades del negocio y provee las funcionalidades a sus clientes a través de interfaces bien definidas y publicadas". Por lo tanto un servicio puede ser un método de un EJB, que va ser utilizado por sus clientes a través de una interfaz bien definida.

La forma de implementar los servicios en Java EE 5 es utilizando el framework EJB 3.0 y JBoss Seam. Estos frameworks contienen un conjunto de clases que facilitan la tarea de implementación, distribución y utilización de EJBs. En el libro EJB in Action se encuentra una guía detallada para la implementación y uso de EJBs.

Dentro de las buenas prácticas planteadas por los autores del libro EJB in Action plantean dos patrones para la implementación de los EJBs: el patrón Facade y el patrón EAO (Entity Access Object) los cuales permiten una mejor distribución e independencia de los

[37] GUIJUN, Wang; CASEY K, Fung. Artículo: Architecture Paradigms and Their Influences and Impacts on Component-Based Software Systems. Proceedings of the 37th Hawaii International Conference on System Sciences – 2004

elementos que se utilizan dentro del los EJBs reduciendo el acoplamiento y aumentado la cohesión de los servicios implementados. En la siguiente figura se puede apreciar el flujo de datos por medio de estos patrones.

Figura 28. Patrones Facade y EAO.

El cliente hace peticiones de los servicios invocando métodos del EJB que implementa el patrón Facade, y recibe la respuesta a través de este mismo EJB. La clase que implementa el patrón Facade se encarga de desarrollar la lógica del negoció de los servicios que va a prestar el componente, sin implementar los métodos específicos de manipulación de datos. El EAO se encarga de controlar el acceso a los métodos de manipulación de las entidades, es decir los métodos de persistencia, modificación, eliminación y consulta de datos, el EAO divide la lógica del negoció de las operaciones que tienen que ver con la base de datos. Las clases que implementan cada patrón tienen su interfaz bien definida, por lo tanto permite cambiar la implementación de la clase sin afectar al cliente respectivo.

Para hacer la implementación tanto del patrón Facade como del EAO se deben utilizar EJBs de sesión. En la figura se presenta la implementación del patrón facade, se pueden apreciar la estructura básica de un EJB de sesión con las anotaciones fundamentales que se pueden utilizar.

Figura 29. Código para la implementación del patrón Facade.

```
import java.util.ArrayList;

@Stateless(name="ConsultarUnidades")
@Name("ConsultarUnidades")
public class ConsultarUnidadesEJB implements ConsultarUnidades {

    @EJB
    private UnidadEAO unidadEAO;

    private Log log = new Log("recursosHumanosServicios");
    private List<Unidad> unidades;
    private List<ClaseUnidad> clasesUnidad;
    private List<FondoPresupuestal> fondosPresupuestales;
    private List<TipoUnidad> tiposUnidad;
    private List<TipoCostoUnidad> tiposCostoUnidad;

    /**
     * Método que permite consultar las unidades que pertenecen a una clase de unidad determir
     * El resultado de la consulta se devuelve en el atributo unidades del EJB.
     * @param aClaseUnidad - La clase de unidad a la cual se le van a consultar las unidades.
     * @return Listado de Unidades.
     */
    public List<Unidad> getUnidades(ClaseUnidad aClaseUnidad) throws Exception{
        unidades = new ArrayList<Unidad>();
        unidades = unidadEAO.consultarPorClase(aClaseUnidad);
        return this.unidades;
    }
```

En (1) se definen las anotaciones de definición de EJB de sesión, en este caso se utiliza *@Stateless(name="ConsultarUnidades")* indicando un sesión bean sin estado y el name indica el nombre del EJB en el contenedor. La etiquete *@Name* de JBoss Seam, permite definir un nombre para el EJB con el cual es reconocido por JDNI, en el contenedor Seam y en el servidor de aplicaciones EJB.

En (2) se implementa la interfaz ConsultarUnidades, es la interfaz que sirve de especificación de los métodos que tiene el EJB. Todo EJB debe poseer una interfaz bien definida.

En (3) se instancia el EJB UnidadEAO que es la implementación del patrón EAO, el cual permite el acceso a los métodos de persistencia de las entidades.

En (4) se definen los atributos propios del EJB.

En (5) se encuentra la definición de los métodos especificados en la interfaz. Se puede apreciar que se utiliza el patrón EAO para consultar las unidades por la clase de unidad, el cual devuelve un listado de unidades, logrando la separación del acceso a datos de la lógica del negocio, en caso que llegue a cambiar la forma de consultar las entidades solo se modifica el EAO sin afectar al Facade.

En la figura presentada a continuación se puede apreciar la implementación de la interfaz utilizada para el ejemplo anterior.

Figura 30. Definición de la interfaz para el EJB.

```
package co.edu.uis.recursosHumanos.servicios;

import java.util.List;

@Local (1)
public interface ConsultarUnidades {
    public List<Unidad> getUnidades(Unidad aCriteriosUnidad) throws Exception;
    public Unidad getUnidad(Integer aCodigoUnidad)throws Exception;
    public List<Unidad> getUnidades(ClaseUnidad aClaseUnidad)throws Exception; (2)
    public List<Unidad> getUnidades(FondoPresupuestal aFondoPresupuestal)throws Exception;
    public List<Unidad> getUnidades(TipoUnidad aTipoUnidad)throws Exception;
    public List<Unidad> getUnidades(TipoCostoUnidad aTipoCosto)throws Exception;
    public List<Unidad> consultarUnidadesACargo(Unidad aUnidadSuperior)throws Exception;
    public List<Unidad> consultarUnidadesDelPrograma(Unidad aPrograma)throws Exception;
```

En (1) se define la interfaz como local por medio de la anotación @Local, la cual indica que el EJB se va utilizar en la misma máquina de los clientes que lo van a utilizar, en caso que el acceso fuera remoto se utilizaría @Remote.
En (2) se tiene la definición de los servicios que va implementar el EJB.

A continuación se presenta la implementación del patrón EAO

Figura 31. Código para la implementación del Patrón EAO.

```
package co.edu.uis.recursosHumanos.servicios;

import java.text.DateFormat;

@Stateless(name="UnidadEAO")
@Name("UnidadEAO") (1)
public class UnidadEAOImp implements UnidadEAO { (2)

    @PersistenceContext
    private EntityManager em; (3)

    private Log log = new Log("recursosHumanosServicios"); (4)

     * Método que permite consultar las unidades de la Universidad por los siguientes criterios:
    public List<Unidad> consultar(Unidad aCriteriosUnidad) throws Exception{

    /**
     * Método que permite consultar una unidad de la Universidad por su código. (5)
     * @param aCodigoUnidad - Código de la unidad a buscar.
     * @return Unidad - La unidad instanciada que tiene el código suministrado. La unidad
     * irá nulla si el código no fue encontrado.
     */
    public Unidad consultarPorCodigo(Integer aCodigoUnidad) throws Exception{
        Unidad unidad = new Unidad();
        unidad = em.find(Unidad.class, aCodigoUnidad); (6)
        return unidad;
    }
```

En (1) se definen las anotaciones de definición de EJB de sesión, en este caso se utiliza *@Stateless(name="UnidadEAO")* indicando un sesión bean sin estado y el name indica el nombre del EJB en el contenedor. La etiqueta @Name de jboss seam, define el nombre para el ejb.
En (2) se implementa la interfaz UnidadEAO, es la interfaz que sirve de especificación de los métodos de acceso a datos que tiene el EJB.
En (3) se instancia el EntityManager el cual es una interfaz que permite la conexión con la base de datos y la manipulación de las entidades.
En (4) se definen los atributos propios del EAO.
En (5) se encuentran los métodos de acceso a las entidades.
En (6) se ejecuta el método find del entityManager, el cual permite consultar una entidad por su llave primaria.

Los resultados de esta fase son:

- Archivo *.jar con las clases implementadas.
- Archivo *.ear con los EJBs y archivos de configuración necesarios para utilizar los EJBs.
- Archivo ds.xml, datasource que define la cadena de conexión con la base de datos.

4.4.4. PRUEBAS FUNCIONALES DE LOS SERVICIOS.

Esta fase pretende en primer lugar garantizar que los servicios fueron implementados correctamente y que cumplen con la totalidad de los requisitos funcionales planteados para el presente componente. En la tabla siguiente se describen las pruebas a realizar sobre los servicios.

Tabla 9. Pruebas funcionales de los servicios.

PRUEBAS FUNCIONALES DE LOS SERVICIOS		
Descripción Prueba	**Objetivo**	**Resultado Esperado**
Prueba Unitaria del EAO: Esta	Verificar que los métodos de acceso	Cada método se debe ejecutar sin

prueba consiste en probar cada uno de los métodos de acceso a entidades implementados en el EAO.	a entidades están implementados correctamente.	generar ninguna excepción ni error y los resultados devueltos deben ser los correctos.
Prueba Unitaria del Facade: Esta prueba consiste en probar uno por uno los servicios que ofrece el componente y están implementados en el Facade.	Verificar que los servicios del componente están implementados correctamente.	Cada método se debe ejecutar sin generar ninguna excepción ni error y los resultados devueltos deben ser los correctos.
Prueba de requisitos: Esta prueba consiste en verificar las funcionalidades implementadas contra los casos de uso definidos en la etapa inicial, para asegurar que el componente cumple con la totalidad de los requisitos planteados. Se puede realizar por medio de una lista de chequeo, donde se especifique cada requisito.	Asegurar que el componente cumple con todos los requisitos planteados en los casos de uso.	Cumplimiento de la totalidad de los requisitos.

Las pruebas unitarias del EAO y del Facade se puede realizar utilizando JUnit, el cual permite automatizar las pruebas, también se pueden probar al igual que en el caso de las entidades por medio de una aplicación de prueba donde se implementen y ejecuten los métodos necesarios para realizar dichas pruebas.

4.4.5. IMPLEMENTACIÓN DE SERVICIOS WEB.

Esta fase pretende exponer los servicios que se requieran, creados por medio de los EJBs en la fase anterior como servicios web, se exportan como servicio web aquellos métodos que no manejen una transacción que necesite interacción permanente con el usuario y no requieran mantener su estado o el estado de los procesos que esté manejando el cliente o el usuario.

La implementación en java de los servicios web se hace por medio del api JAX-WS (Java API para XML Web Services). JAX-WS es una tecnología para construir servicios web y sus clientes que se comunican usando XML. Permite a los desarrolladores escribir servicios web orientados por mensajes y servicios web orientado por RPC.

En JAX-WS, una invocación a una operación de un servicio web es representada por un protocolo basado en XML, este protocolo es SOAP. La especificación SOAP define las reglas de codificación y las convenciones para representar las invocaciones y respuestas de los servicios web. Estas llamadas y respuestas son transmitidas como mensajes SOAP (archivos XML) sobre HTTP. Los mensajes SOAP son complejos de implementar, el API JAX-WS esconde esta complejidad al desarrollador. En el servidor el desarrollador especifica las operaciones del servicio web definiendo métodos en una interface escrita en java. El desarrollador define una o más clases que implementen estos métodos. Los clientes de servicios web también son fáciles de codificar. Utilizando JAX-WS los clientes y los servicios web tiene una gran ventaja: la independencia de la plataforma, es decir un cliente puede invocar un servicio web que no esté corriendo en la plataforma java y viceversa.

Un EJB para ser expuesto como servicio web debe cumplir con los siguientes requisitos:

- La clase debe estar anotada con la anotación javax.jws.WebService o javax.jws.WebServiceProvider.
- Los métodos que son expuestos como servicio web deben estar anotados con la anotación javax.jws.WebMethod y no deben ser declarados como *static* o *final*.
- El EJB no debe ser declarado como *final*, no debe ser abstracto, debe tener un constructor público por defecto, debe ser un Ejb de sesión sin estado, es decir debe tener la anotación *@Stateless* y no debe definir el método *finalize.*
- Los parámetros y tipos de retorno de los métodos que son expuestos al cliente servicio web deben ser compatibles con JAXB. En la tabla siguiente se definen los tipos de dato compatibles por defecto.

Tabla 10. Mapeo en XML de los tipos de datos Java.

MAPEO EN XML DE LOS TIPOS DE DATOS JAVA	
Tipo xml	**Tipo de dato java**
xsd:string	Java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd:long	long

xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:doublé	doublé
xsd:boolean	boolean
xsd:byte	byte
xsd:Qname	javax.xml.namespace.QName
xsd:dateTime	javax.xml.datatype.XMLGregorianCalendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:unsignedInt	long
xsd:unsignedShort	int
xsd:unsignedByte	short
xsd:time	javax.xml.datatype.XMLGregorianCalendar
xsd:date	javax.xml.datatype.XMLGregorianCalendar
xsd:anySimpleType	java.lang.Object
xsd:anySimpleType	java.lang.String

4.4.6. PRUEBAS DE LOS SERVICIOS WEB.

Para probar los servicios web es necesario crear un cliente que consuma los servicios web y verificar que los resultados obtenidos son los esperados. Se deben realizar pruebas remotas y con clientes implementados en otras plataformas.

4.4.7. IMPLEMENTACIÓN DE LOS COMPONENTES PERSONALIZADOS PARA LA INTERFAZ DE USUARIO (COMPONENTES UI).

Con la creación de los componentes personalizados para la interfaz de usuario se pretende reutilizar la lógica del negocio encapsulada en el componente en aplicaciones web e intranet. El objetivo principal de esta fase es crear los componentes UI para la lógica de la aplicación seleccionada.

La idea fundamental de desarrollar componentes personalizados para la interfaz de usuario, es implementar como componentes JSF lógica del negocio que se pueda reutilizar en muchas aplicaciones de una forma sencilla y transparente para el

desarrollador. Esta lógica de negocio que se puede implementar como componente UI comúnmente son consultas donde el usuario selecciona un elemento de un conjunto de datos para luego aplicar un proceso sobre el elemento seleccionado o manejarle una acción, Por ejemplo, en la universidad, una consulta de unidades académico – administrativas que se filtre por diferentes criterios para buscar y seleccionar una unidad. Luego de tener la unidad seleccionada el desarrollador puede programar sobre ella diferentes acciones como: consultar el personal de esa unidad, relacionar la unidad seleccionada a una vinculación laboral como la unidad donde va a laborar la persona, consultar las solicitudes de auxiliaturas hechas por esa unidad, etc. Este tipo de componente es muy útil y se reutilizaría mucho en las aplicaciones desarrolladas por la universidad.

Con este tipo de componentes UI al desarrollador se le ahorrarían muchas líneas de código y esfuerzos, no tendría que repetir o incluir las mismas líneas de código de una aplicación a otra.

En el tutorial de java EE 5[38] se definen unos criterios para identificar cuando es necesario implementar un componente UI y se detallan a continuación:

- Cuando se necesite adicionar un nuevo comportamiento a un componente estándar de jsf o de richfaces, tal como generar un tipo de evento adicional.
- Cuando se necesite crear un nuevo componente no implementado que tiene su propio comportamiento. Un ejemplo es un seleccionador de fechas que consiste en tres combos, o la selección de una ciudad buscando por país y departamento.
- Cuando se quiera reutilizar lógica de negocio en muchas aplicaciones, típicamente consultas para seleccionar un dato o una entidad.
- Cuando se necesita un componente que es soportado por un cliente HTML pero no está actualmente implementado por la tecnoligia JSF.

[38] SUN MICROSYSTEM, The Java EE 5 tutorial. 2008

El método de implementación de los componentes personalizados para la interfaz de usuario se definió en el capítulo 2 sección 2.3. Para una mayor referencia se recomienda consultar CDK developer guide[39].

En el capítulo 5 se elabora un componente utilizando el kit de desarrollo de componentes de Richfaces para consultar y seleccionar una unidad académico administrativa, definiendo los parámetros de consulta que puede utilizar el usuario desde la etiqueta puesta en la página.

4.4.8. PRUEBAS DE LOS COMPONENTES UI.

Las pruebas a realizar de los componentes personalizados para la interfaz de usuario se enfocan principalmente en probar que la implementación del componente quedó bien realizada y cumple con los requisitos planteados. Las pruebas a realizar se definen el la siguiente tabla.

Tabla 11. Pruebas de los Componentes UI.

PRUEBAS DE LOS COMPONENTES UI		
Descripción Prueba	**Objetivo**	**Resultado Esperado**
Prueba funcional: Con esta prueba se pretende probar todas las funcionalidades implementadas en el componente. Se realiza suministrando los datos necesarios para probar cada funcionalidad.	Verificar que las funcionalidades implementadas en el componente se ejecutan correctamente.	Cada funcionalidad se debe ejecutar sin generar ninguna excepción ni error y los resultados devueltos deben ser los correctos.
Prueba de eventos: Con esta prueba se quiere detectar posibles fallas en la manipulación de eventos que hace el componente. Se deben probar los eventos que puede manejar el componente. Se deben crear casos de prueba por evento donde se especifique una acción a realizar y un mensaje a mostrar.	Verificar que el componente responde de forma adecuada a los eventos que maneja.	Se ejecuta el evento correctamente y muestra el mensaje suministrado en el caso de prueba.

[39] SERGEY, Smirnov. CDK Developer Guide. Jboss Comunitiy. 2008

Prueba de utilización: Con esta prueba se debe garantizar que el componente se puede utilizar correctamente con varias instancias en la misma página.	Verificar que el componente se puede utilizar correctamente con varias instancias en la misma página.	Que cada instancia del componente funcione correctamente, asignando el valor seleccionado en lugar que le corresponde, sin interferir con las demás instancias.

En Maven al generar el proyecto de desarrollo del componente, se generan las clases respectivas para realizar las pruebas unitarias con JUnit, se tiene que implementar cada caso de prueba y en el momento de instalar el componente antes de crear el jar. Maven ejecuta estas pruebas para verificar que el componente está implementado correctamente.

Con la realización de las pruebas se garantiza que el componente UI se puede utilizar en muchos proyectos de desarrollo software y que funciona correctamente.

4.4.9. ESPECIFICACIÓN DEL COMPONENTE SOFTWARE REUTILIZABLE.

Un componente tiene que mostrar información suficiente para poder ser implementado y usado. La especificación del componente se centra en:

- Descripción del componente.
- La especificación de su interfaz: muestra todo lo que el cliente debe saber para su uso en tiempo de ejecución.
- La especificación de la implementación: Define la información necesaria del componente en tiempo de diseño, para ser apropiadamente implementado, distribuido o instalado.
- Especificación del las características funcionales del componente: Especificar de los casos de uso que maneja.
- Especificación de las características no funcionales, como son los recursos requeridos, el rendimiento, la fiabilidad, la concurrencia, etc.
- Especificación de las características más importantes del diseño del componente.
- Especificación de las pruebas realizadas al componente.

La especificación del componente software reutilizable en estudio en la presente investigación se divide en las siguientes partes:

- Documentación: Se realizará utilizando los javaDocs, para describir las funcionalidades implementadas en las clases java que hacen parte del componente.
- Ficha de especificación, en la cual se describirá la especificación de implementación, de recursos, de factores internos al componente, entre otros. Se puede apreciar en la tabla mostrada a continuación. Se basa principalmente en los estudios hechos por Geisterfer y Ghosh[40].
- Modelos UML: Se adjuntan los modelos UML realizados en la creación del componente incluye: Diagrama de casos de uso, modelo del dominio del problema, diagramas de clases y el diagrama de componentes del área de aplicación.

Tabla 12. Ficha de Especificación de los componentes reutilizables.

FICHA DE ESPECIFICACIÓN DEL COMPONENTE					
NOMBRE:	Nombre del componente.	**FECHA:**	Fecha de creación	**VERSIÓN:**	1.0
DESCRIPCIÓN:	Descripción breve del componente, que hace, las funcionalidades que maneja, domino de aplicación al cual pertenece.				
PALABRAS CLAVE:	Palabras que identifiquen al componente.				
ESPECIFICACIÓN DE IMPLEMENTACIÓN					
TECNOLOGIA:	Tecnología en la cual fue implementado el componente, junto con sus versiones.				
INTERFAZ:	Nombre de la interfaz que define los servicios del componente.				
SERVICIOS:	Define la signatura de los servicios que presta el componente.				
PRECONDICIONES:	Definen los requisitos que se deben cumplir antes de entrar a utilizar el componente.				
POSTCONDICIONES:	Definen los requisitos que se deben cumplir después de utilizar el componente.				
DEPENDENCIAS:	Hace referencia a los elementos de los cuales el componente depende para su correcto funcionamiento.				
ESPECIFICACIÓN NO FUNCIONAL					
RECURSOS:	Define los recursos necesarios para que el componente funcione correctamente y los recursos computacionales necesarios para su funcionamiento.				
RESTRICCIONES:	Define los alcances del componente, qué hace, hasta donde lo hace y qué no hace.				
NIVEL DE SEGURIDAD:	Define el grado de seguridad manejada por el componente, también define las recomendaciones de seguridad que se deben tener en cuenta al utilizar el componente.				
RENDIMIENTO:	Se especifican las recomendaciones de configuración o de cualquier tipo para que el				

[40] GEISTERFER, CJ Michael; GHOSH, Sudipto. Artículo: Software component Specification: A Study in Perspective of Component Selection and Reuse. Proceedings of the Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems. IEEE 2006.

	rendimiento del componente sea el mejor. Se especifica también si tiene limitaciones de concurrencia que puedan afectar el rendimiento.
PORTABILIDAD:	Define las plataformas en las cuales funciona correctamente el componente.
GRADO DE REUTILIZACIÓN:	Es el puntaje obtenido por el componente en el análisis de reutilización realizado en el modelo de selección de componentes.
DISTRIBUCIÓN:	Define la forma como se debe instalar y utilizar el componente en las plataformas soportadas.

Con la elaboración de la ficha de reutilización se tiene una especificación completa para el componente software reutilizable. Lo ideal sería sistematizar las fichas de especificación, las cuales podrían quedar almacenadas en una base de datos y los desarrolladores por medio de una aplicación web consultar los componentes disponibles en el repositorio por diferentes criterios: por palabras clave, por descripción, por nombre, por grado de reutilización, etc. El resultado de la consulta serían los componentes que cumplen con esos criterios y poder ver y analizar la ficha de especificación de cada uno de ellos, para escoger el que mejor se adapte a las necesidades del desarrollador.

4.5. ESQUEMA DE DISTRIBUCIÓN DEL COMPONENTE

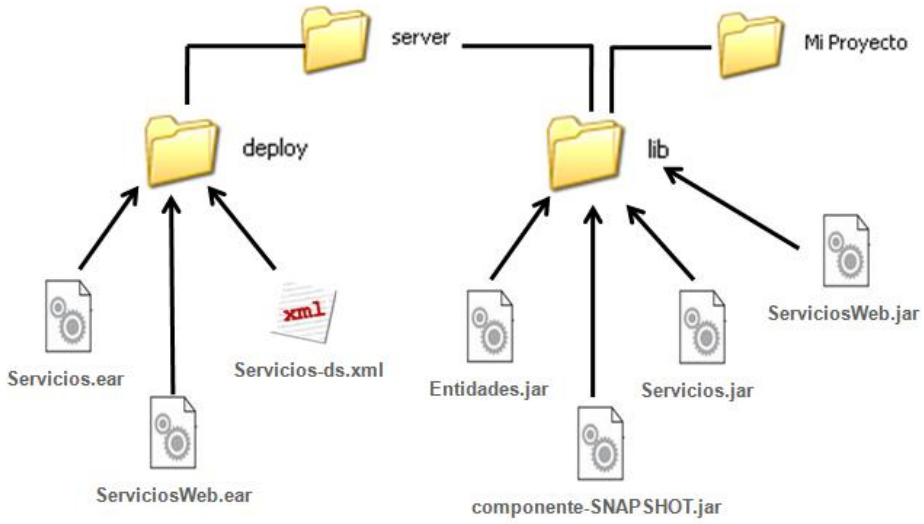
Terminada la implementación del componente se procede a realizar su distribución para que los desarrolladores de software lo puedan utilizar. Los resultados de la implementación del componente son:

- Jar con el mapeo de las entidades.
- Jar con la implementación de los servicios.
- ear con la implementación de los servicios.
- dataSource.xml, archivo de configuración de la conexión con la base de datos.
- Jar con la implementación de los servicios web.
- Jar con la implementación de los componentes para la interfaz de usuario.
- JavaDocs de las clases java implementadas.
- Ficha de especificación.
- Modelos UML del componente.

Para realizar la distribución del componente primero se debe crear un archivo build.xml de la herramienta ANT por medio del cual se pueda instalar en el servidor de aplicaciones el componente de forma automática, luego empaquetar los archivos en un archivo *.tar o *.zip para ser colocado en el servidor de versiones de la empresa de desarrollo.

En la siguiente figura se presenta el esquema de distribución en un servidor de aplicaciones del componente software reutilizable.

Figura 32. Esquema de distribución del componente reutilizable.



Se puede apreciar que los archivos *.jar deben ir en las librerías del servidor y del proyecto de desarrollo donde se vayan a utilizar. Los archivos *.ear van en la carpeta de distribución del servidor de aplicaciones: server / deploy, con el archivo *-ds.xml que contiene la conexión a las base de datos.

Al realizar la distribución de esta manera se puede comenzar a utilizar el componente en los proyectos de desarrollo de la División de Servicios de Información.

En el servidor de versiones de la empresa de desarrollo el programador encontrará un único archivo con toda la documentación y especificación del componente. También encontrara un archivo build.xml, que automáticamente instala el componente en el servidor de aplicaciones por medio de la herramienta ANT.

5. DISEÑO E IMPLEMENTACIÓN DEL COMPONENTE PARA LA DIVISIÓN DE SERVICIOS DE INFORMACIÓN.

En la División de Servicios de Información (DSI) de la Universidad Industrial de Santander se está realizando el proyecto de desarrollo de las nuevas versiones de los sistemas de información Académico, Financiero y de Recursos Humanos en Java versión empresarial 5. Estos sistemas dan soporte a los procesos más importante que se llevan a cabo en la Universidad, son utilizados principalmente por estudiantes, profesores y por personal administrativo para la toma de decisiones. Este proyecto es llevado a cabo por un grupo de trabajo de la DSI, dedicado a la investigación y desarrollo de estos sistemas. Actualmente el proyecto se encuentra iniciando la fase de implementación.

La presente investigación hace parte del proyecto de las nuevas versiones de los sistemas de información de la UIS. Se pretende aportar con sus resultados un proceso de desarrollo de componentes software reutilizables, por medio del cual el desarrollo de los sistemas informáticos se realice de forma más rápida a la forma tradicional y con mejor calidad. Es importante resaltar que con el avance de la investigación se van utilizando y probando los resultados por el grupo de desarrollo de las nuevas versiones de los sistemas de información, recibiendo aportes fundamentales para refinar el proceso de desarrollo; en forma reciproca con el avance de la investigación se hicieron aportes importantes al proyecto mencionado.

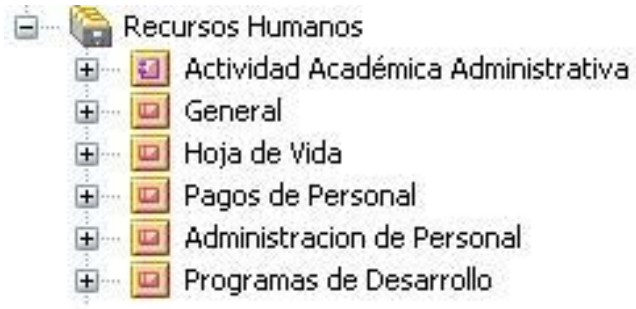
Es muy importe resaltar que este tipo de investigaciones trae grandes beneficios a la empresa de desarrollo software, en este caso a la DSI, la cual es el laboratorio de pruebas de los planteamientos teóricos y prácticos que se realizan en la investigación, así los resultados se están probando y refinando en la implementación de software real, aplicable y que requiere calidad. Los resultados obtenidos por este tipo de investigaciones son más acordes a la realidad y se garantizan que funcionan correctamente, no se quedan solo en planteamientos teóricos que nunca son probados en el desarrollo de software empresarial.

Para la aplicación del proceso de desarrollo planteado en los capítulos anteriores se seleccionó el área de aplicación en el Sistema de Recursos Humanos de la Universidad

Industrial de Santander, área que administra la información del personal que labora en esta institución, los cargos, las unidades académico - administrativas, la actividad académica y administrativa del empleado, los programas de desarrollo, entre muchas otras.

En la siguiente figura se pueden apreciar las principales aplicaciones que contiene el sistema de Recursos Humanos.

Figura 33. Aplicaciones del Sistema de Recursos Humanos[41].



Cada aplicación tiene un número grande de aplicaciones complejas, por ejemplo, la aplicación de administración de personal contiene:

- Administración de aprendices.
- Administración de jubilados y sustitutos.
- Administración de servicios prestados.
- Auxiliaturas.
- Becas de sostenimiento.
- Administración de docentes cátedra.
- Administración de personal planta.

Se puede apreciar que el área de aplicación es muy grande y es necesario delimitarla, para seleccionar el dominio de aplicación para el cual se va a utilizar el proceso de desarrollo de componentes software reutilizables.

[41] La carpeta General contiene los casos de uso generales que son utilizados por las demás aplicaciones de recursos Humanos y las demás aplicaciones de la Universidad.

5.1. ANALISIS DE REQUISITOS DEL DOMINIO DE APLICACIÓN.

Como se definió en el capitulo anterior con esta fase se pretende establecer el dominio de aplicación y el modelo de requisitos para la biblioteca de componentes que se pretende crear. El resultado de esta fase son: la descripción del dominio de aplicación y el modelo de casos de uso que definen las funcionalidades que deben manejar la biblioteca de componentes.

En la siguiente tabla se muestran las áreas de aplicación candidatas, estas áreas fueron establecidas en conjunto con el jefe de la DSI y con el líder del área del sistema de Recursos Humanos, en la tabla también se realiza la aplicación de los criterios de evaluación definidos para clasificar las áreas de aplicación del proceso de desarrollo de componentes.

Los criterios que se evaluaron fueron:

1. **Grado de Utilización.** Escala: 1: Muy bajo, 2: Bajo, 3: Medio, 4: Alto, 5: Muy alto.
2. **Dificultad de implementación.** Escala: 5: Muy baja, 4: Baja, 3: Media, 2: Alta, 1: Muy alta.
3. **Complejidad:** Escala: 5: Muy baja, 4: Baja, 3: Media, 2: Alta, 1: Muy alta.
4. **Cambiabilidad:** Escala: 5: Muy baja, 4: Baja, 3: Media, 2: Alta, 1: Muy alta.
5. **Nivel de Abstracción:** Escala: 1: Muy bajo, 2: Bajo, 3: Medio, 4: Alto, 5: Muy alto.
6. **Necesidad de Implementación:** Escala: 1: Muy bajo, 2: Bajo, 3: Medio, 4: Alto, 5: Muy alto.
7. **Posibilidad de componente ya desarrollado:** Escala: 5: Muy baja, 4: Baja, 3: Media, 2: Alta, 1: Muy alta.

Tabla 13. Evaluación áreas de aplicación.

AREA DE APLICACIÓN	CRITERIOS DE EVALUACIÓN							
	1	2	3	4	5	6	7	Puntos
Administración y consulta de personal: Encapsula la lógica de negocio para la vinculación y consulta del personal teniendo en cuenta los diferentes tipos de nómina que tiene la Universidad.	5	1	2	4	4	5	4	25
Tramite Solicitudes: Componente que generaliza el trámite de cualquier	5	2	2	2	4	3	4	22

solicitud. Teniendo en cuenta los pasos necesarios para su aprobación, consulta y productos generados a partir de la aprobación de la solicitud.								
Administración de roles, usuarios, menús: Permite la administración de los usuarios, roles y permisos sobre las funcionalidades de cualquier aplicación DSI.	5	3	3	3	3	4	2	23
Generador de Listados: Componente que permite la generación de un reporte en PDF de cualquier entidad, EJB o consulta estableciendo los campos o criterios a mostrar en el informe.	4	3	4	4	4	3	1	23
Consultas Generales de Recursos Humanos: Componente para realizar las consultas sobre unidades académico administrativas, cargos, municipios, departamentos, países, tipos de documento.	5	4	3	4	3	5	4	28
Administración ordenes de trabajo: Componente que permite manejar las órdenes de trabajo generadas a partir de una solicitud o por una persona, teniendo en cuenta secciones, material utilizado y personal que las realiza.	3	2	3	3	2	2	4	19

Conclusión:

Según los criterios de selección establecidos para cada área de aplicación, el área de aplicación candidata a implementar sería: **Consultas Generales del Sistema de Recursos Humanos**, estando en un segundo lugar: Administración y Consulta de Personal.

5.1.1. DESCRIPCIÓN DEL DOMINIO DE APLICACIÓN: CONSULTAS GENERALES DE RECURSOS HUMANOS.

Las consultas generales son las consultas que utilizan todos los sistemas de información de la universidad, para poder implementar las funcionalidades propias de cada sistema. Son datos básicos, representan en su gran mayoría a las tablas tipo del sistema de información de recursos humanos.

Las consultas generales se especifican a continuación:

Consulta de continentes, países, departamentos y municipios: Esta consulta debe permitir buscar:

- Los continentes por: código y por nombre.

- Los países por los siguientes criterios: código, nombre y continente.
- La capital de un país.
- Los departamentos por: código del departamento, por nombre y por país.
- La capital de un departamento.
- Los municipios por: código, nombre, departamento y país.

Consulta de los Tipos de Estado civil: Para buscar los tipos de estado civil de acuerdo a los siguientes criterios:

- Por código del estado civil.
- Por descripción del estado civil.
- Listado de todos los estados civiles.

Consulta de Tipos de Documento: Permite obtener los datos de los tipos de documento por alguno de los siguientes criterios:

- Por código del tipo de documento.
- Por la descripción del tipo de documento.
- Por abreviatura del tipo de documento.
- Por la señal que indica si es documento personal o documento de soporte.

Consulta de las unidades o dependencias de la universidad: Permite buscar las unidades por algunos de los siguientes criterios de consulta:

- Por código de unidad.
- Por clase de Unidad
- Por tipo de Unidad
- Por nombre de unidad
- Por la unidad superior
- Por vigencia de la unidad.
- Por fondo presupuestal

Consulta de Empleados: Permite buscar los datos personales de cualquier empleado de la Universidad, consultar docentes de planta, buscar docentes cátedra, consultar el personal de planta, obtener los jefes de unidad por cualquiera de los siguientes criterios de consulta:

- Por tipo y número de documento.
- Por nombres.
- Por apellidos.
- Por cargo
- Por nivel del cargo
- Por tipo de nómina.
- Por sede
- Por período de vigencia
- Por unidad

Consulta de Cargos: Permite buscar los datos de los cargos por cualquiera de los siguientes criterios de consulta:

- Por código del cargo.
- Por descripción del cargo.
- Por nivel del cargo.
- Por tipo de cargo.
- Por clase de cargo.
- Por vigencia del cargo.

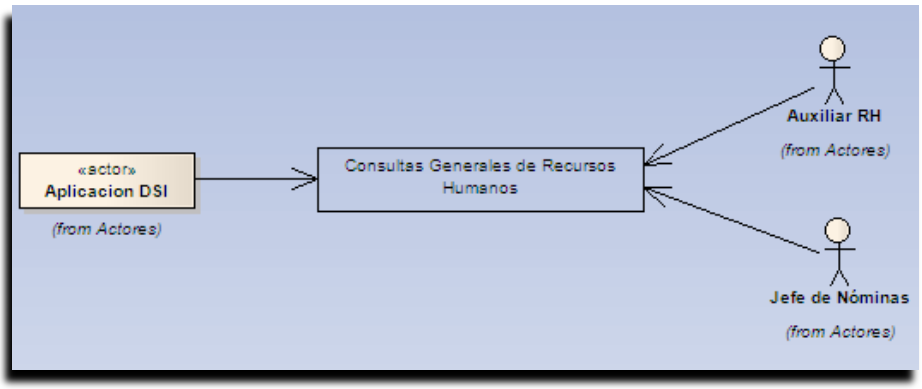
Las personas encargadas de la actualización, creación y modificación de la información base para las consultas generales son: El jefe de nóminas y un auxiliar de la División de Recursos Humanos.

5.1.2. MODELO DE CASOS DE USO.

Siguiendo los planteamientos hechos por Weitzenfeld, en primer lugar se identifican los actores principales del dominio de aplicación seleccionado.

En la figura que aparece a continuación se puede apreciar los actores principales de las consultas generales de Recursos Humanos y se definen a continuación:

Figura 34. Actores de las consultas generales de Recursos Humanos.



Aplicación DSI: Es el actor que representa a todas las aplicaciones de la División de Servicios de información y requiere utilizar alguna consulta general del sistema de Recursos Humanos.

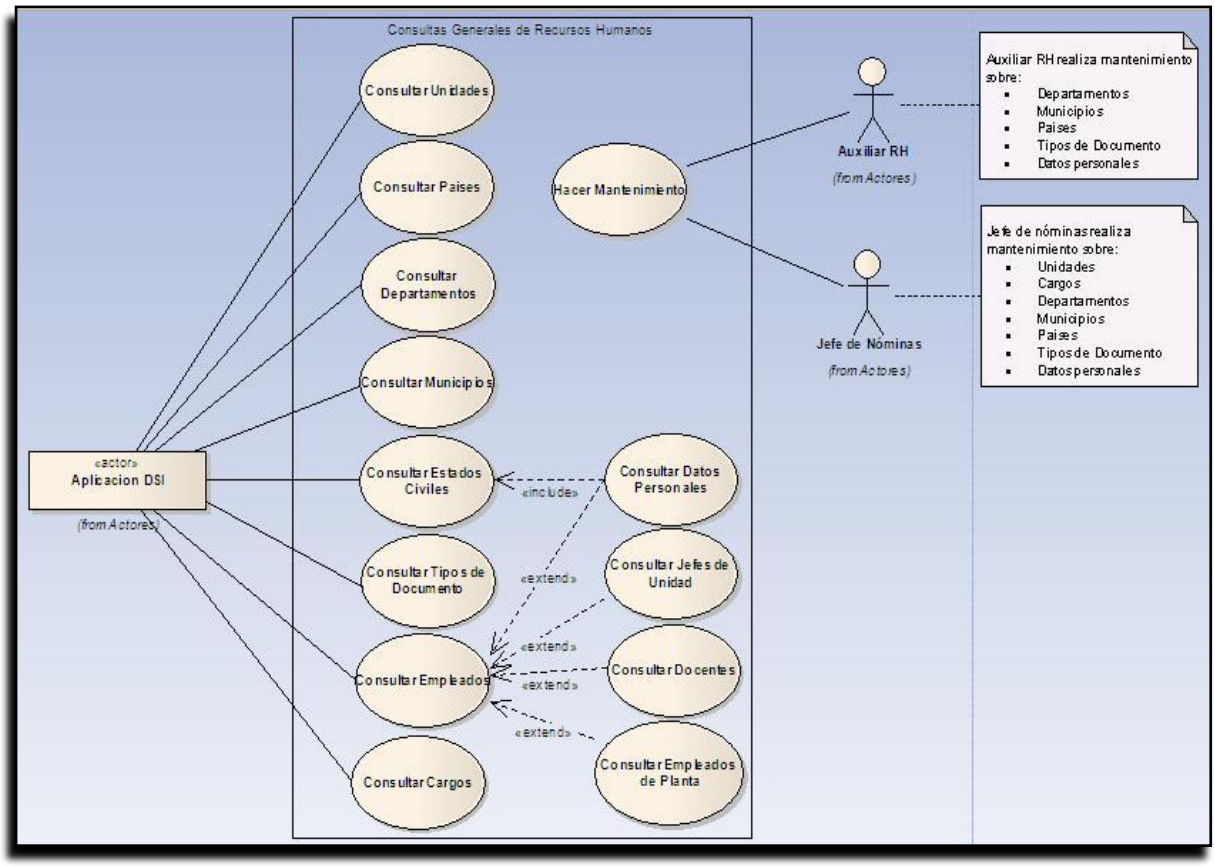
Auxiliar RH: Representa a un auxiliar de la División de Recursos Humanos el cual puede crear, modificar y consultar la información de: Departamentos, Municipios, Países, Tipos de Documento y Datos personales.

Jefe de Nóminas: Representa al jefe de nóminas de la División de Recursos Humanos el cual puede crear, modificar y consultar la información de: Cargos, Unidades, Departamentos, Municipios, Países, Tipos de Documento y Datos personales.

Después de identificar los actores, se procede a la creación de los diagramas de casos de uso, los cuales se presentan en la siguiente figura.

Con la descripción del dominio de aplicación realizada en el numeral anterior y la definición de los casos de uso se da por terminada la fase de análisis de requisitos del área de aplicación y se puede realizar el modelo de selección de componentes.

Figura 35. Modelo de casos de uso de las consultas generales.



5.2. MODELO DE SELECCIÓN DE COMPONENTES: CONSULTAS GENERALES DEL SISTEMA DE RECURSOS HUMANOS.

Con el modelo de selección de componentes se pretende identificar los componentes software y garantizar que sean reutilizables para el dominio de aplicación seleccionado. El modelo se puede apreciar en la sección 4.2.

5.2.1. MODELO DEL DOMINIO DEL PROBLEMA.

Con el modelo del dominio del problema se pretenden identificar las clases de entidad que pertenecen al dominio de aplicación se realiza utilizando la ingeniería de textos sobre la descripción del problema y utilizando los casos de uso. En la tabla que se muestra a continuación se presentan las entidades identificadas para las consultas generales de Recursos Humanos.

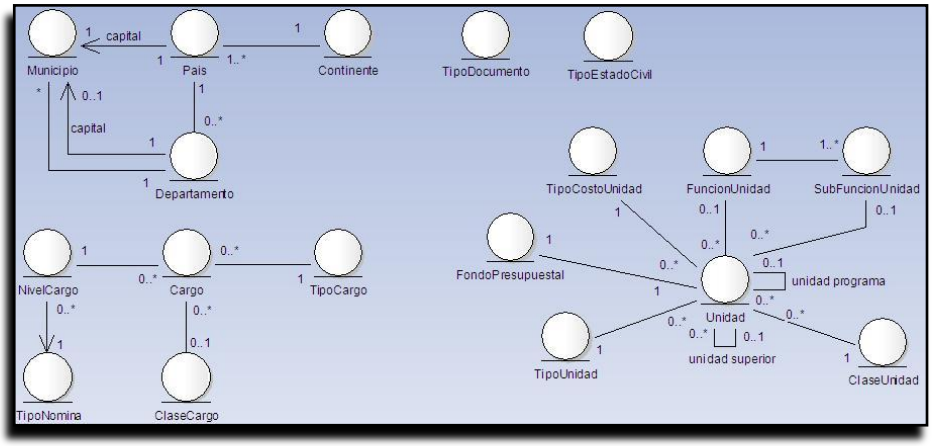
Para la identificación de entidades se tuvo en cuenta el diseño de la base de datos relacional de la versión actual del sistema de información de Recursos Humanos, lo cual facilita la identificación de entidades y de sus atributos.

Tabla 14. Identificación de entidades.

ENTIDADES IDENTIFICADAS			
Municipio	TipoDocumento	TipoCargo	TipoCostoUnidad
Pais	TipoNomina	Persona	FuncionUnidad
Continente	NivelCargo	ClaseUnidad	SubFuncionUnidad
Departamento	ClaseCargo	TipoUnidad	Unidad
TipoEstadoCivil	Cargo	FondoPresupuestal	

Después de tener identificadas las entidades se procede a establecer las relaciones existentes entre ellas y elaborar en diagrama de clases de entidad. En la figura siguiente se puede apreciar el diagrama de clases de entidad.

Figura 36. Diagrama de Clases de Entidad de las consultas generales.



En el modelo de entidades no se tuvo en cuenta la parte del modelado de las consultas de empleados, debido a que su lógica maneja una complejidad más alta y se deben relacionar más entidades, como se pretende mostrar un ejemplo básico y fácil de entender, es suficiente con las entidades planteadas en la presente sección. Esas consultas formarían otro componente que utilizaría muchas de las entidades mostradas en el diagrama de la figura anterior, se tuvo en cuenta en la selección del área de aplicación de la biblioteca de componentes, quedando en segundo lugar.

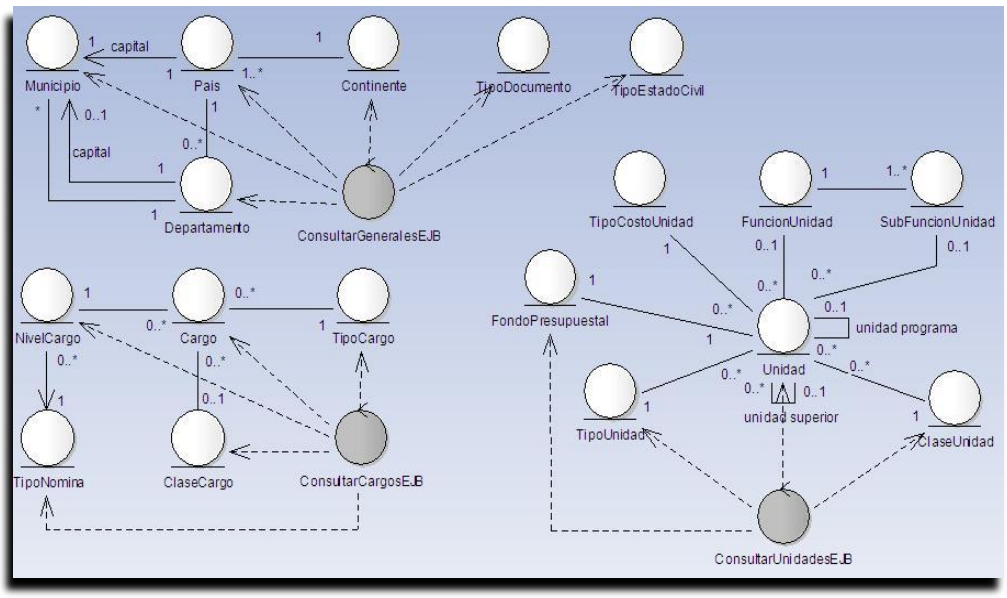
5.2.2. MODELO DE ANÁLISIS.

El modelo de análisis pretende identificar las clases de control necesarias para implementar la lógica del negocio del área de aplicación seleccionada. Para el caso de estudio de la presente investigación se identificaron tres clases de control:

- ConsultarUnidadesEJB
- ConsultarCargosEJB
- ConsultarGeneralesEJB

Por lo tanto el diagrama de clases quedaría de la siguiente manera:

Figura 37. Diagrama de clases: Relación entre entidades y EJBs.



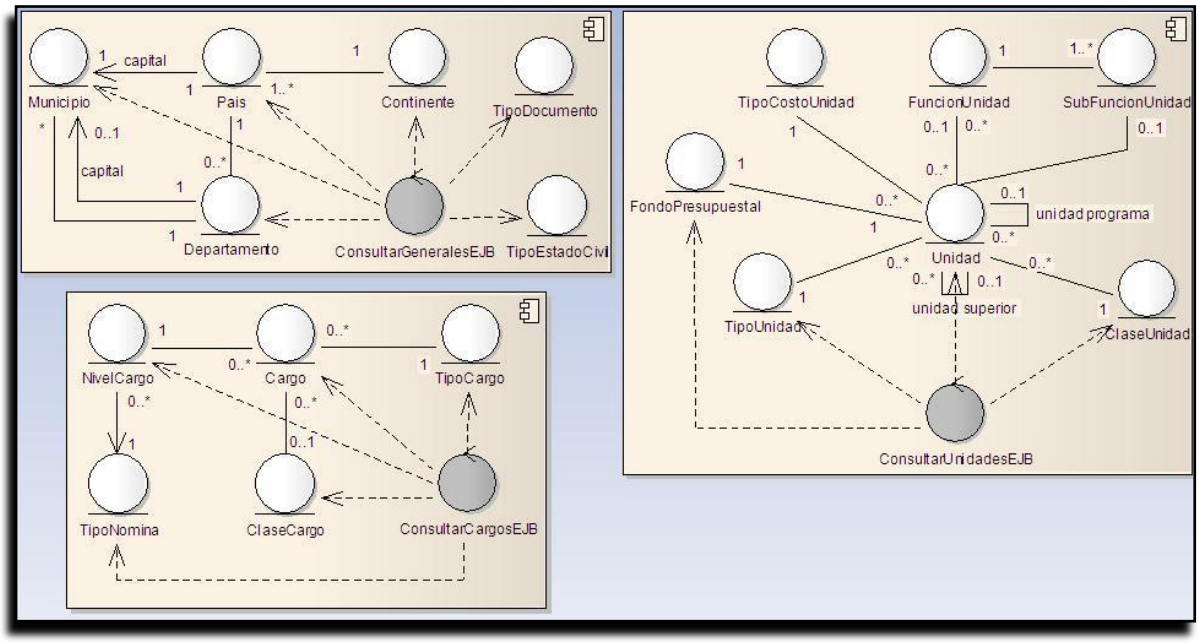
Se puede resaltar que se muestra una relación de dependencia entre los EJB y las entidades que va a manejar.

5.2.3. MODELO DE COMPONENTES.

El objetivo principal del modelo de componentes es agrupar las entidades y los EJB en componentes independientes, de tal forma que se puedan reutilizar en los proyectos de desarrollo de forma independiente. Otra de las ventajas de agrupar en componentes es que la labor posterior del mantenimiento es más fácil solo se tiene que modificar cada componente sin afectar la funcionalidad total del sistema.

En el diagrama de clases anterior es clara la separación de las entidades y es fácil agrupar en componentes las entidades y los ejbs. En la gran mayoría de los casos hacer esta separación no es fácil, implica conocer muy bien la lógica del negocio y marcar bien las dependencias entre componentes. En este caso los componentes identificados no tienen dependencia de los otros. En la figura siguiente se muestra el diagrama de componentes para el caso de estudio. En el caso que existieran relaciones entre los componentes estas relaciones pasan a ser las dependencias del componente y se tendría que analizar que tan marcada es cada dependencia, si es muy marcada la dependencia se podrían unir los dos componentes en uno solo. Este análisis lo debe hacer un arquitecto software con la suficiente experiencia para definir los componentes y saber que implicaciones tiene agrupar de una forma u otra las entidades y los EJBs.

Figura 38. Diagrama de componentes inicial del área de aplicación.



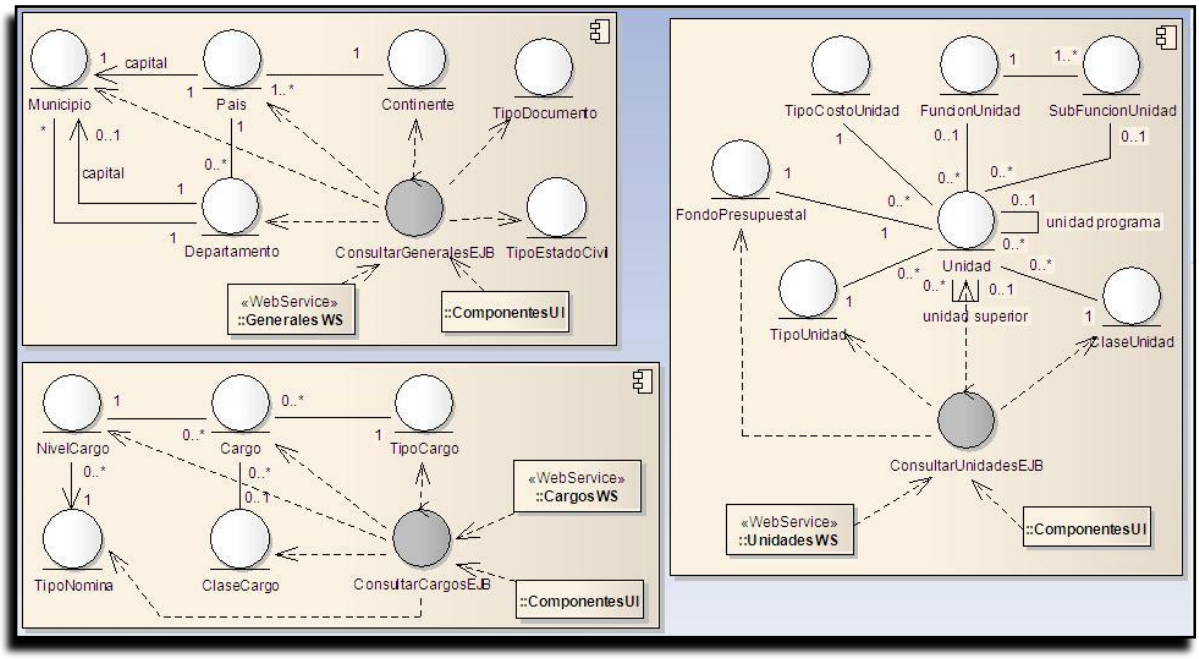
El resultado de esta fase fue la identificación de los tres componentes mostrados:

- Componente Generales.
- Componente Cargos.
- Componente Unidades.

Hasta el momento los componentes identificados constan solo de entidades y EJBs, falta definir los servicios web y los componentes personalizados para la interfaz de usuario, con estos elementos los componentes quedarían completos, cumpliendo con los planteamientos hechos desde el principio de la investigación.

Por lo tanto el diagrama de componentes mostrado en la figura anterior cambia porque se deben agregar estos elementos. El diagrama de componentes completo para el área de aplicación seleccionada se puede apreciar en la siguiente figura.

Figura 39. Diagrama de componentes del área de aplicación.



5.2.4. ANÁLISIS DE REUTILIZACIÓN.

Una vez identificados los componentes se proceden a aplicar los criterios de selección y de reutilización para asegurar que sean reutilizables en muchas aplicaciones. Los criterios de selección pueden determinar el orden de implementación de los componentes y aseguran que son reutilizables. Se definieron en la sección 4.2.4. son los siguientes:

1. **Dependencia estructural.** Escala: 1- Muy alta, 2- Alta, 3- Media, 4- Baja, 5- Muy baja.
2. **Facilidad Mantenimiento**. Escala: 1- Muy Difícil, 2- Difícil, 3- Media, 4- Fácil, 5- Muy fácil.
3. **Grado de Adaptabilidad.** Escala: 1-Muy Bajo, 2- Bajo, 3- Medio, 4- Alto, 5- Muy alto.
4. **Grado extensibilidad.** Escala: 1- Muy Alto, 2- Alto, 3- Medio, 4- Bajo, 5- Muy bajo.
5. **Grado de Utilización.** Escala: 1-Muy Bajo, 2- Bajo, 3- Medio, 4- Alto, 5- Muy alto.
6. **Portabilidad.** Escala: 1-Muy Baja, 2- Baja, 3- Media, 4- Alta, 5- Muy alta.
7. **Complejidad.** Escala: 1- Muy Alta, 2- Alta, 3- Media, 4- Baja, 5- Muy baja.
8. **Cambiabilidad.** Escala: 1- Muy Alta, 2- Alta, 3- Media, 4- Baja, 5- Muy baja.

9. **Posibilidad de componente ya desarrollado.** Escala: 1- Muy Alta, 2- Alta, 3- Media, 4- Baja, 5- Muy baja.

Tabla 15. Análisis de reutilización de los componentes identificados.

COMPONENTE	CRITERIOS DE EVALUACIÓN									
	1	2	3	4	5	6	7	8	9	Puntos
Generales: Componente que permite administrar y consultar la información correspondiente a continentes, países, departamentos, ciudades, tipos de documentos y tipos de estado civil.	5	4	3	3	5	3	5	5	2	35
Unidades: Componente que permite la administración y consulta de las unidades de la universidad.	5	4	3	3	5	2	4	4	4	34
Cargos: Componente que permite la administración y consulta de los cargos de la universidad.	5	4	3	3	4	2	4	3	4	32

Se puede apreciar que los tres componentes pasan de los 30 puntos cumpliendo con los requisitos planteados para garantizar que son reutilizables. Por lo tanto el resultado de esta fase son 3 componentes software reutilizables para el área de aplicación seleccionada.

Los puntajes entre cada componente son similares debido a que son desarrollados bajo la misma filosofía y son muy parecidos. Estos criterios de evaluación y selección de componentes se pueden aplicar en el desarrollo basado en componentes, para seleccionar y clasificar los componentes a utilizar para cumplir cierto requisito en el sistema que se esté desarrollando. También sirven para comparar componentes ya desarrollados (COTS) y mirar cuál de ellos es mejor.

Para los capítulos posteriores se va a trabajar sobre el componente de Unidades, porque con él se pretende mostrar los pasos necesarios para realizar su implementación, prueba y distribución, se considera que mostrando cómo se desarrolla un componente los otros se pueden desarrollar de forma similar. Antes de entrar a la fase de implementación es necesario definir la arquitectura a utilizar en el desarrollo del componente.

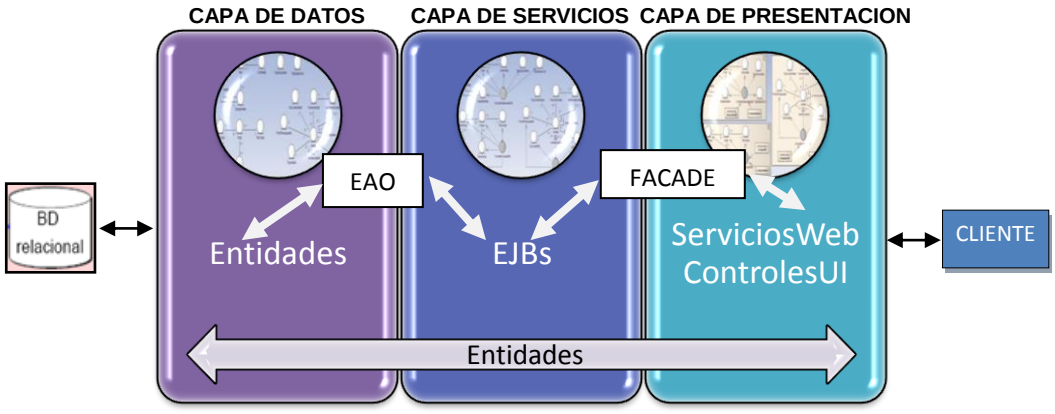
5.3. DEFINICIÓN DE LA ARQUITECTURA DEL COMPONENTE A IMPLEMENTAR.

Las alternativas de arquitectura fueron definidas en la sección 4.3, la arquitectura define la estructura del componente, la cual está formada por las entidades, EJBs, servicios web, componentes UI y sus relaciones. El diseño arquitectónico debe satisfacer los requisitos funcionales y no funcionales definidos para el dominio de aplicación.

Partiendo del modelo de componentes del área de aplicación se pueden ver los elementos que conforman el componente y sus relaciones. La arquitectura seleccionada es la arquitectura por capas, basada en la arquitectura de las aplicaciones Java EE 5 en la figura que se presenta a continuación se define la arquitectura para los componentes reutilizables a implementar.

Se puede resaltar que la comunicación entre las capas de datos y de servicios se realiza por medio del patrón EAO (Entity Access Object), es decir cada operación que necesite realizar los EJBs sobre las entidades se deben hacer por medio de este patrón. Por otro lado la comunicación entre la capa de presentación y la capa EJB se realiza por medio del patrón FACADE, el cual permite mejorar el rendimiento de las operaciones realizadas sobre los EJBs. La flecha de la parte inferior indica el flujo de información entre las capas e indica también que las entidades se pueden instanciar en cualquier capa, siguiendo la filosofía de los POJOs[42] planteados en la versión Java EE 5.

Figura 40. Arquitectura del componente a implementar.



[42] Play Old Java Objects – Jugar con los objetos java antiguos, simples y útiles.

En la sección 4.3 se definieron cada una de las capas presentadas en la figura anterior.

5.4. DESARROLLO, PRUEBAS Y ESPECIFICACIÓN DEL COMPONENTE.

Teniendo definida la arquitectura y los modelos que definen el componente a desarrollar, antes de entrar a implementar los elementos que conforman el componente es importante definir las herramientas a utilizar para la codificación, pruebas y distribución. Las herramientas utilizadas en la investigación fueron las utilizadas en la División de Servicios de Información para el desarrollo de aplicaciones empresariales, las cuales se detallan a continuación:

- Servidor de aplicaciones: JBoss aplication server versión 4.2.0.GA
- IDE de desarrollo: JBoss Developer Studio. Version: 2.0.0.Beta1
- Servidor de versiones: subclipse-site-1.4.7, ajdt_1.6.1a_for_eclipse_3.4, +org.tmatesoft.svn_1.2.1.eclipse
- Herramienta de compilación y distribución: Apache ANT.
- Base de datos relacional: Informix 10.0.

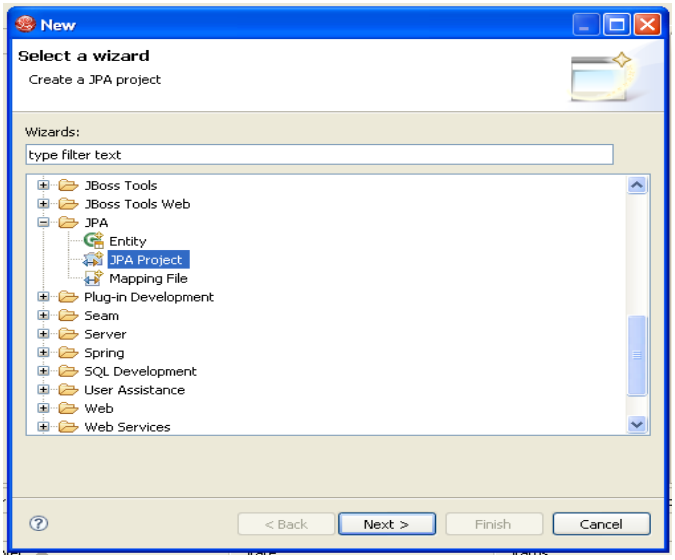
También se utilizaron los estándares de programación de la División de Servicios de Información. La implementación se realiza siguiendo el procedimiento propuesto en la sección 4.4 de este documento.

5.4.1. IMPLEMENTACIÓN DE ENTIDADES.

Para realizar la implementación de las entidades en primer lugar se debe crear un proyecto JPA (Java Persistence API) en el JBoss developer studio, los pasos para crear el proyecto JPA son:

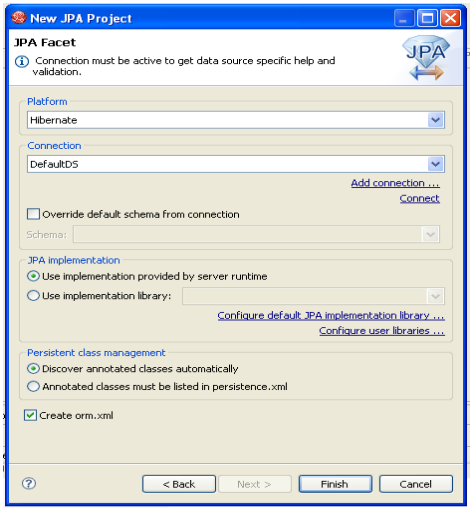
1. Seleccionar el menú file / new / other y abre la ventana mostrada en la siguiente figura, en la cual se selecciona JPA y JPA Project.

Figura 41. Seleccionar Proyecto JPA.



2. Clic en el botón *Next*, luego se abre otra ventana donde se introduce el nombre del proyecto en este caso sería UnidadesEntidades y se da clic en *Next*. Luego se muestra la siguiente ventana:

Figura 42. Parámetros del Proyecto JPA.



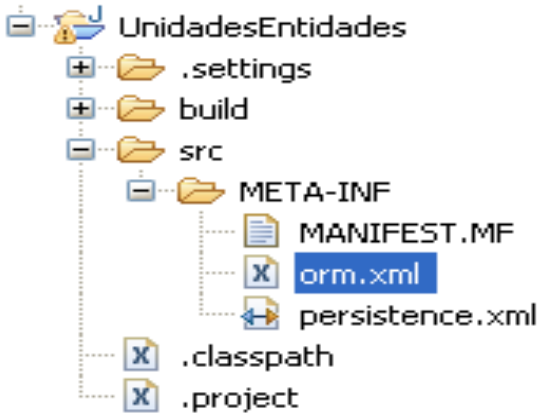
3. Se especifica:

- plataforma = Hibernate.
- Connection = DefaultDS.
- JPA Implementacion = Use implementation provided by server runtime.
- Persist class management = Discover annotated classes automatically.
- Seleccionar Create orm.xml.

4. Clic en *finish*.

El JBoss developer studio crea la estructura del proyecto de entidades con los archivos de configuración y de distribución. Esta estructura se muestra a continuación.

Figura 43. Estructura del Proyecto JPA.



En la estructura se puede destacar el directorio src, en el cual se debe crear el paquete donde quedaran implementadas las entidades, en este caso sería: co / edu / uis / recursosHumanos / entidades.

En el directorio src / META-INF se pueden apreciar dos archivos de configuración importantes el orm.xml y el persistence.xml, el orm.xml se utiliza para mapear las entidades en xml contra la base de datos relacional; mientras el persistence.xml define la configuración de la unidad de persistencia utilizada en el proyecto.

Después de tener la estructura del proyecto lista se procede a crear una por una las entidades definidas en la sección 5.2.1. Para crear una entidad se debe ubicar en la

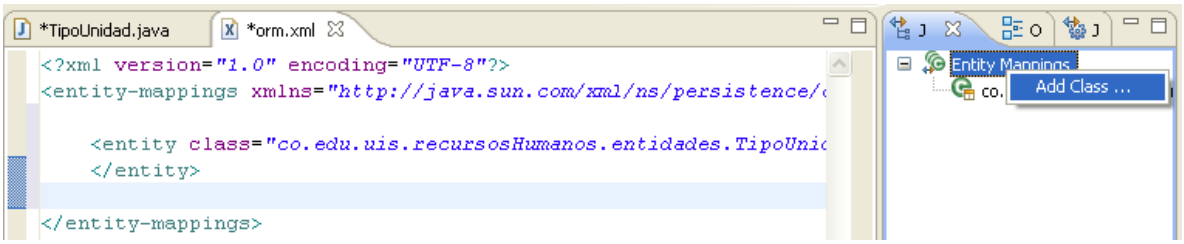
carpeta del paquete donde va quedar la entidad hacer clic derecho / new / class. Se especifican los datos requeridos y clic en finalizar.

La forma de implementación de la clase de entidad se definió en la sección 4.4.1 y en la sección 2.2.1.

Después de tener hecha la implementación de las entidades se procede a mapearlas en el archivo de configuración orm.xml, Para incluir las entidades se debe realizar:

- Abrir el archivo orm.xml
- En la perspectiva JPA del JBoss developer Studio, se encuentra la vista JPA Structure (ver siguiente figura), al hacer clic derecho sobre el titulo Entity Mappings, se pueden ir agregando una por una las entidades.
- En el código generado, se debe colocar el atributo name=”Nombre”, que se definió para la entidad, dentro de la etiqueta <entity>.

Figura 44. Mapear entidades en el archivo orm.xml.



Luego de tener mapeadas las entidades en el archivo orm.xml se puede compilar y generar el *.jar de las entidades. Para realizar la compilación y creación se debe utilizar ANT, creando una archivo buil.xml donde se definen los comandos que debe hacer para compilar y generar el *.jar. Se genera el archivo unidades-entidades.jar.

5.4.2. PRUEBA DE ENTIDADES.

Para realizar las pruebas se creó un proyecto web por medio de la herramienta Seam-gen, la cual crea el esqueleto de una aplicación web, generando las páginas principales y los archivos de configuración[43].

Las pruebas se realizaron de la siguiente manera:

- Incluir el archivo unidades-entidades.jar en la carpeta lib del proyecto web creado.
- Crear una página para realizar las pruebas, llamada probarUnidades.xhtm.
- Crear un EJB para realizar los métodos de prueba, propuestos en la sección 4.4.2.
- Compilar y distribuir la aplicación de prueba
- Entrar a la página de prueba.
- Verificar los resultados obtenidos al ejecutar cada método.
- Documentar los resultados.

En la tabla siguiente se puede apreciar los resultados de las pruebas realizadas. Se probaron los métodos de relación con las otras entidades, creando una instancia y haciendo el llamado al método respectivo. Se hizo un método para obtener los listados de las relaciones uno a muchos o muchos a muchos, y son mostrados en una tabla.

Tabla 16. Pruebas unitarias del mapeo de las entidades.

PRUEBAS UNITARIAS DEL MAPEO DE LAS ENTIDADES		
Entidad	**Métodos probados**	**Resultado**
Unidad	unidad.getClaseUnidad()	OK
	unidad.getFondoPresupuestal()	OK
	unidad.getFuncion()	OK
	unidad.getPrograma()	OK
	unidad.getSubFuncion()	OK
	unidad.getTipoCostoUnidad()	OK
	unidad.getTipoUnidad()	OK

[43] Para ver más información de la generación de proyectos web por medio de Seam puede consultar: DAN, Allen; Seam in Action. Manning Publications. 2008

	unidad.getUnidadesACargo()	OK
	unidad.getUnidadesPrograma()	OK
	unidad.getUnidadSuperior()	OK
	getListadoUnidades()	OK
	insertarUnidad()	OK
	modificarUnidad()	OK
	eliminarUnidad()	OK
TipoUnidad	tipo.getUnidades();	OK
	getListadoTipos();	OK
	insertarTipoUnidad();	OK
	modificarTipoUnidad()	OK
	eliminarTipoUnidad()	OK
ClaseUnidad	clase.getUnidades()	OK
	getListadoClasesUnidad();	OK
	insertarClaseUnidad();	OK
	modificarClaseUnidad()	OK
	eliminarClaseUnidad()	OK
FondoPresupuestal	fondo.getUnidades();	OK
	getListadoFondos();	OK
	insertarFondoPresupuestal();	OK
	modificarFondoPresupuestal ();	OK
	eliminarFondoPresupuestal ();	OK
TipoCostoUnidad	tipo.getUnidades();	OK
	getListadoTiposCosto;	OK
	insertarTipoCosto();	OK
	modificarTipoCosto ();	OK
	eliminarTipoCosto ();	OK
FuncionUnidad	funcion.getSubFunciones();	OK
	getListadoFuncioesUnidad();	OK
	insertarFuncion();	OK
	modificarFuncion ();	OK
	eliminarFuncion ();	OK
SubFuncionUnidad	subFuncion.getFuncion();	OK
	getListadoSubFunciones	OK
	insertarSubFuncion();	OK
	modificarSubFuncion ();	OK
	eliminarSubFuncion ();	OK

La ventaja de utilizar las pruebas de esta forma, es que la aplicación web de prueba queda disponible para que los desarrolladores la puedan consultar y cerciorarse que se cumplen en su totalidad.

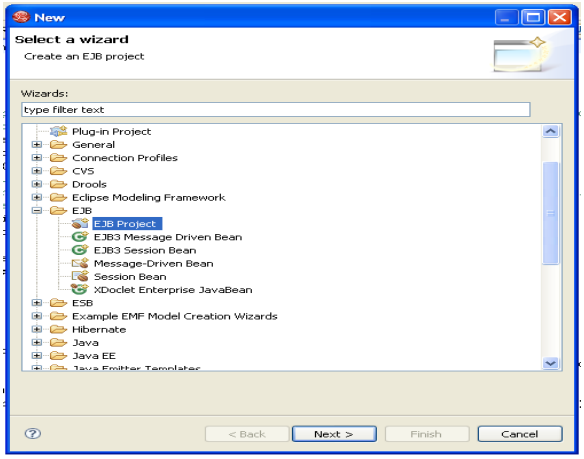
Con las pruebas se puede concluir que el mapeo de las entidades que conforman el componente Unidades se realizó correctamente.

5.4.3. IMPLEMENTACIÓN DE SERVICIOS.

Para realizar la implementación de los servicios que va a prestar el componente por medio del JBoss Developer Studio es necesario crear un proyecto modulo EJB (EJB-Module), de la siguiente manera:

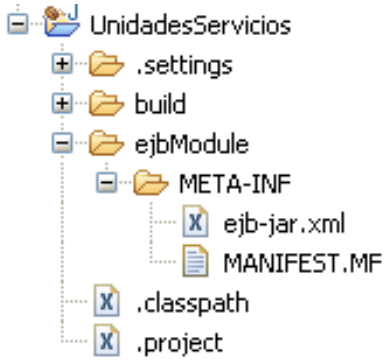
1. Estando en JBoss developer studio ir a file / new / other. En la ventana de creación del nuevo proyecto seleccionar EJB, como se muestra en la figura y dar clic en Next.

Figura 45. Crear proyecto EJB.



2. Especificar el nombre del proyecto, para este caso sería UnidadesServicios, y dar clic en *Next*.
3. Seleccionar la casilla que dice generate deployment descriptor, y clic en *finish*.

Figura 46. Estructura generada para el proyecto EJB.



Con los pasos anteriores se crea el proyecto para realizar un módulo EJB, la estructura generada por el JBoss es la mostrada en la figura anterior.

Es necesario crear los siguientes directorios y archivos para su correcto funcionamiento y poder así implementar los servicios.

- Directorio lib, necesario para incluir las librerías requeridas para implementar los servicios.
- En META-INF, crear los siguientes archivos, con el contenido mostrado a continuación.

Aplication.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/application_5.xsd"
        version="5">

  <display-name>unidades-servicios</display-name>

  <module>
```

```
    <ejb> unidades-servicios.jar</ejb>
  </module>

  <!-- Seam and EL -->
  <module>
     <ejb>jboss-seam.jar</ejb>
  </module>

</application>
```

Components.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns="http://jboss.com/products/seam/components"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:core="http://jboss.com/products/seam/core"
            xsi:schemaLocation="
    http://jboss.com/products/seam/components
    http://jboss.com/products/seam/components-2.0.xsd">
</components>
```

Persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
                        http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
            <persistence-unit name="RecursosHumanosServicios">
                        <jta-data-source>java:/UnidadesServiciosDatasource
                        </jta-data-source>
                        <jar-file>unidades-entidades.jar</jar-file>
                        <properties>
        <property name="hibernate.dialect"
                        value="org.hibernate.dialect.InformixDialect"/>
        <property name="hibernate.show_sql" value="true"/>
```

```
        <property name="hibernate.format_sql" value="true"/>
        <property name="jboss.entity.manager.factory.jndi.name"
          value="java:/ UnidadesServiciosEntityManagerFactory"/>
      </properties>
          </persistence-unit>
</persistence>
```

- En la raíz del proyecto crear el archivo que define la conexión a la base de datos (dataSource).

Unidades-servicios-ds.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE datasources
   PUBLIC "-//JBoss//DTD JBOSS JCA Config 1.5//EN"
   "http://www.jboss.org/j2ee/dtd/jboss-ds_1_5.dtd">
<datasources>
  <local-tx-datasource>
    <jndi-name>UnidadesServiciosDatasource</jndi-name>
    <connection-url>
        jdbc:informix-sqli://ipserver:puerto/BD:INFORMIXSERVER=Schema</connection-url>
    <driver-class>com.informix.jdbc.IfxDriver</driver-class>
    <user-name>login</user-name>
    <password>clave</password>
  </local-tx-datasource>
</datasources>
```

- En la raíz del proyecto también crear el archivo build.properties con la siguiente línea de código: jboss.home =C\:\\jboss-4.2.0.GA. Que define la ruta del servidor de aplicaciones, en este caso está en la carpeta raíz. También se crea el archivo build.xml que es un archivo con comandos de ANT para realizar la compilación y distribución de los servicios.

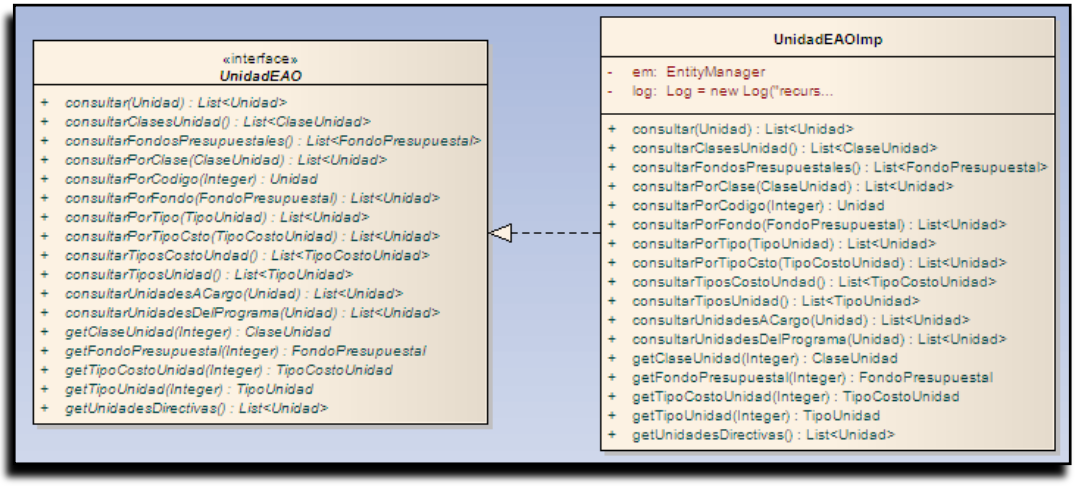
Después de tener los archivos mencionados de configuración se comienza con el desarrollo de los EJB que implementan los servicios del componente software reutilizable, la pautas y guias para desarrollar los EJB se definieron en la sección 4.4.3.

El orden de implementación de los servicios es el siguiente:

- Codificar la interfaz del EAO: UnidadEAO.
- Codificar la clase que implementa el EAO: UnidadEAOImp.
- Codificar la interfaz del facade: ConsultarUnidades.
- Codificar la clase que implementa el facade: ConsultarUnidadesEJB.

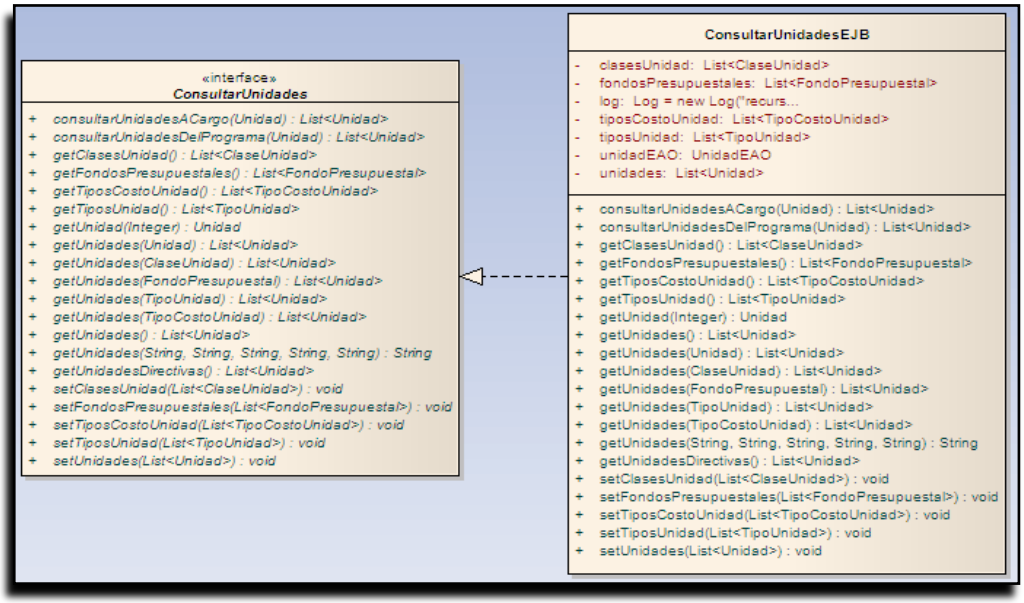
En la figura siguiente se muestra los métodos implementación en el patrón EAO.

Figura 47. Implementación del patrón EAO.



Después de tener el EAO, se procede a desarrollar los EJBs necesarios para implementar los servicio que va a prestar el componente y el patrón Facade. En este caso solo se necesita un EJB y en ese mismo EJB se implementa el patrón Facade. En la figura que aparece a continuación se presenta el EJB implementado.

Figura 48. Implementación del patrón Facade.



Con la implementación del EJB ConsultarUnidades se cumplen con los requisitos planteados en los casos de uso y en la descripción del dominio de aplicación. Los resultados son el archivo unidades-servicios.jar, que contiene las clases implementadas, el unidades-servicios.ear, que es el archivo para distribuir en el servidor y el archivo unidades-servicios-ds.xml, que define la conexión con la base de datos y la unidad de persistencia para el componente.

Después de terminar la implementación de los servicios se procede a realizar la prueba funcional para garantizar que quedaron bien implementados.

5.4.4. PRUEBAS FUNCIONALES SERVICIOS.

Para realizar las pruebas de los servicios, se retoma la aplicación web creada para realizar la prueba de las entidades, se crea una nueva página probarUnidadesServicios.xhtm donde se realizaran las pruebas, también es necesario crear un EJB dentro del proyecto, para en él instanciar los servicios y hacer el llamado a

todos los métodos implementados, se debe verificar que los resultados obtenidos son los esperados y que no se presenta ningún error.

El procedimiento realizado para las pruebas fue:

- Distribuir en el servidor el archivo unidades-servicios.ear y el archivo unidades-servicios-ds.xml.
- Copiar el archivo unidades-servicios.jar en el lib del servidor y del proyecto de prueba.
- Implementar la página de prueba probarUnidadesServicios.xhtml y el EJB donde se crean los métodos.
- Crear los métodos de prueba.
- Compilar y distribuir la aplicación de prueba
- Entrar a la página de prueba.
- Verificar los resultados obtenidos al ejecutar cada método.
- Documentar los resultados

Tabla 17. Pruebas unitarias de los servicios implementados.

PRUEBAS UNITARIAS DE LOS SERVICIOS		
EJB	**Métodos probados**	**Result.**
UnidadEAO	consultar(Unidad aCriteriosUnidad);	OK
	consultarPorCodigo(Integer aCodigoUnidad);	OK
	consultarPorClase(ClaseUnidad aClaseUnidad)	OK
	consultarPorFondo(FondoPresupuestal aFondo)	OK
	consultarPorTipo(TipoUnidad aTipoUnidad)	OK
	consultarPorTipoCsto(TipoCostoUnidad aTipo)	OK
	consultarUnidadesACargo(Unidad aUnidad)	OK
	consultarUnidadesDelPrograma(Unidad aProgram)	OK
	consultarClasesUnidad();	OK
	consultarFondosPresupuestales()	OK
	consultarTiposUnidad()	OK
	consultarTiposCostoUndad()	OK
	getTipoUnidad(Integer aCodigoTipo)	OK
	getClaseUnidad(Integer aCodigoClase)	OK
	getFondoPresupuestal(Integer aCodigoFondo)	OK
	getTipoCostoUnidad(Integer aCodigoTipo	OK
	getUnidadesDirectivas()	OK
ConsultarUnidades	getUnidades(Unidad aCriteriosUnidad)	OK

	getUnidad(Integer aCodigoUnidad)	OK
	getUnidades(ClaseUnidad aClaseUnidad)	OK
	getUnidades(FondoPresupuestal aFondo)	OK
	getUnidades(TipoUnidad aTipoUnidad)	OK
	getUnidades(TipoCostoUnidad aTipoCosto)	OK
	consultarUnidadesACargo(Unidad aUnidadSuperior)	OK
	consultarUnidadesDelPrograma(Unidad aPrograma)	OK
	getUnidades();	OK
	setUnidades(List<Unidad> unidades);	OK
	getClasesUnidad()	OK
	setClasesUnidad(List<ClaseUnidad> clasesUnidad);	OK
	getFondosPresupuestales()	OK
	setFondosPresupuestales(List<FondoPresupuestal>	OK
	fondosPresupuestales);	OK
	getTiposUnidad()	
	setTiposUnidad(List<TipoUnidad> tiposUnidad);	OK
	getTiposCostoUnidad();	OK
	setTiposCostoUnidad(List<TipoCostoUnidad>	OK
	tiposCostoUnidad);	OK
	getUnidades(String aCodigoTipo, String	
	aCodigoClase, String aFondo, String	OK
	aUnidadSuperior, String aNombre);	
	getUnidadesDirectivas();	OK

Después de realizar las pruebas unitarias sobre los EJBs implementados es necesario probar que se cumplen los requisitos planteados por el área de aplicación del componente software reutilizable.

Tabla 18. Prueba de cumplimiento de requisitos.

PRUEBA DE CUMPLIMIENTO DE REQUISITOS	
REQUISITO	**EVALUACIÓN.**
Permitir consultar las unidades por algunos de los siguientes criterios de consulta: • Por código de unidad. • Por clase de Unidad • Por tipo de Unidad • Por nombre de unidad • Por la unidad superior • Por vigencia de la unidad.	Se puede apreciar que con los métodos implementados en el EJB se pueden realizar consultas con los criterios requeridos. Por lo tanto el requisito se cumple en su totalidad.

• Por fondo presupuesta.	

Cabe resaltar que solo se evalúa el requisito que tiene que ver con la consulta de unidades porque se está evaluando el componente reutilizable Unidades y ese es el único requisito que tiene.

Con las pruebas realizadas se garantiza que los servicios del componente funcionan correctamente y cumplen con los requisitos establecidos.

5.4.5. IMPLEMENTACIÓN DE SERVICIOS WEB.

Para la implementación de los servicios web que va a manejar el componente, en primer lugar se tiene que crear un proyecto EJB igual que el caso anterior con el nombre UnidadesWS, se deben crear los mismos archivos de configuración e incluir en la carpeta lib del proyecto generado el archivo unidades-servicios.jar para poder utilizar los servicios implementados.

En la siguiente figura se puede ver la implementación del servicio web para realizar la consulta de las unidades por los siguientes criterios: código del tipo de unidad, código clase de unidad, código del fondo presupuestal, código de la unidad superior y por parte del nombre de la unidad.

Figura 49. Implementación del Servicio Web.

```
import javax.ejb.EJB;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebService;

import org.jboss.seam.Component;

import co.edu.uis.recursosHumanos.servicios.ConsultarUnidades;
import co.edu.uis.servicios.Log;

/**
 * Servicio web que permite consultar las Unidades del universidad.
 */
@Stateless
@WebService
public class ConsultarUnidadesWS{

    @EJB
    ConsultarUnidades consultaUnidades;
    Log log = new Log("recursosHumanosServicios");

    public ConsultarUnidadesWS() {
        super();
    }

    @WebMethod
    public String consultarUnidades(String aCodigoTipo, String aCodigoClase, String aFondo,
            String aUnidadSuperior, String aNombre){
        String respuesta="";
        try {
            consultaUnidades = (ConsultarUnidades)Component.getInstance("consultaUnidades");
            respuesta = consultarUnidades(aCodigoTipo, aCodigoClase, aFondo, aUnidadSuperior, aNombre);
        } catch (Exception e) {
            log.getLog().error(e.toString());
        }
        return respuesta;
    }
}
```

En (1) se importan las clases base para generar los servicios web.

En (2) se encuentran: la anotación @WebServices que extiende al EJB como servicio Web y la anotación @Stateless que indica que la clase es un EJB sin estado.

En (3) se instancia al EJB ConsultarUnidades implementado en la sección anterior.

En (4) aparece la anotación @WebMethod la cual marca el método respectivo para ser ejecutado por medio del servicio web.

En (5) se obtiene la instancia del EJB ConsultarUnidades del servidor de aplicaciones y se ejecuta la consulta.

El resultado devuelve un cadena de caracteres que contiene el listado de las unidades que cumplieron los criterios establecidos separadas por el carácter “|” y cada registro viene con su código y el nombre de la unidad.

Al distribuir en el servidor de aplicaciones el servicio web, automáticamente se generan los archivos xml necesarios para ser expuesto como servicio web incluyendo el archivo WSDL.

5.4.6. PRUEBAS FUNCIONALES SERVICIOS WEB

Para realizar la prueba del servicio web es necesario crear un cliente que consuma el servicio web implementado. El cliente debe realizar:

1. Usar la anotación javax.xml.ws.WebServiceRef y declarar la referencia al servicio web. Esta anotación usa el WSDL el elemento Location para especificar la URI del servicio web distribuido. Ejemplo:

   @WebServiceRef(wsdlLocation="http://localhost:8080/UnidadesWS/consultarUnidadesWS?wsdl")
   static UnidadesWS *servicio*;

2. Recibir un proxy o puerto al servicio, se hace invocando el método getConsultarUnidadesWSPort() del servicio.

   ConsultarUnidadesWS port = servicio. getConsultarUnidadesWSPort();

3. Invocar el método consultarUnidades del puerto, pasando los parámetros requeridos.

   String rta = port.consultarUnidades(“”,””,””,””,”Decanatura”);

El resultado de la consulta hecha por medio del servicio web es un String que contiene el listado de las unidades que tiene en el nombre la palabra Decanatura. Si el resultado devuelto es el correcto, la implementación del servicio web fue acertada.

5.4.7. IMPLEMENTACIÓN COMPONENTES UI.

Con la implementación de los componentes personalizados para la interfaz de usuario se pretende tener componentes JSF listos para ser utilizados en el desarrollo de aplicaciones Web. Se implementó un componente que permite: seleccionar una unidad de un listado y se realiza la consulta por cualquiera de los siguientes criterios:

- Tipo de Unidad.
- Clase de Unidad.
- Fondo Presupuestal.
- Unidad Superior.
- Parte del nombre de la Unidad.

Al seleccionar cualquiera de los criterios propuestos se realiza la consulta y automáticamente muestra los resultados en una tabla para que el usuario seleccione la unida que desea y realizar el proceso requerido.

El componente se implementó de acuerdo a lo planteado en la sección 2.3, pero hizo falta aclarar unos puntos muy importantes que se resuelven a continuación.

El componente de Richfaces para JSF y Ajax se llama ConsultarUnidades y los elementos más importantes para su creación fueron:

- UIConsultarUnidades.java
- ConsultarUnidadesRendereBase.java
- htmlConsultarUnidades.jspx
- ConsultarUnidades.xml
- ConsultarUnidades.js

Con este conjunto de elementos se genera las clases necesarias para ejecutar el componente UI por medio de JSF. A continuación se describen las partes fundamentales de cada elemento.

UIConsultarUnidades.java

Es la clase base para el componente, en ella se define el tipo del componente y su familia. También define los métodos set y get de las propiedades que se requiera, deben ser métodos abstractos. Su implementación se define a continuación.

Figura 50. Clase UIConsultarUnidades.java.

```
package co.edu.uis.recursosHumanos.componentes.component;

import javax.faces.component.UIInput;(1)

/**
 * Clase del componente JSF que define la familia del componente, el tipo y la clase base.
 *
 */
public abstract class UIConsultarUnidades extends UIInput {(2)

    public static final String COMPONENT_TYPE = "co.edu.uis.recursosHumanos.componentes.ConsularUnidades";
    public static final String COMPONENT_FAMILY = "co.edu.uis.recursosHumanos.componentes.ConsularUnidades";(3)

    public abstract String getMensaje();
    public abstract void setMensaje(String aMensaje);(4)
```

En (1) se importa la clase JSF base para el componente, la cual genera todos los métodos base para el componente UI.
En (2) se define el nombre de la clase, por estándar de JSF debe comenzar por el prefijo UI y hereda de UIInput, ya que el componente a crear es un elemento de entrada de datos.
En (3) se define la familia y el tipo del componente UI a implementar.
En (4) se definen los métodos de acceso a las propiedades que lo requieran.

ConsultarUnidadesRendererBase.java

Es la clase base para generar la clase Renderer del componente UI, por medio de la cual se puede codificar y decodificar las etiquetas que representan al componente. Esta clase es la utilizada en la plantilla jspx, en ella se definen los métodos que necesite la plantilla

para generar el componenteUI. A continuación se detallan las partes más importantes de la clase y se pueden apreciar en la siguiente figura.

Figura 51. Implementación de la ConsultarUnidadesRendererBase.java

```
package co.edu.uis.recursosHumanos.componentes.renderkit;

import java.util.List;

public abstract class ConsultarUnidadesRendererBase extends HeaderResourcesRendererBase { (1)

    @EJB
    private ConsultarUnidades consultaUnidades; (2)

    public ConsultarUnidadesRendererBase() {
        super();

    }

    @Override
    protected Class<? extends UIComponent> getComponentClass() { (3)
        return UIConsultarUnidades.class;
    }

    @Override
    public void decode(FacesContext context, UIComponent component) {

        ExternalContext external = context.getExternalContext();
        Map requestParams = external.getRequestParameterMap();
        UIConsultarUnidades uiUnidad = (UIConsultarUnidades)component; (4)
        String clientId = uiUnidad.getClientId(context);
        String submittedValue = (String)requestParams.get(clientId);
        if (submittedValue != null) {
            uiUnidad.setSubmittedValue(submittedValue);
        }

    }
```

En (1) se define clase ConsultarUnidadesRendereBase que extiende de HeaderResourcesRendererBase.

En (2) se inyecta el EJB ConsultarUnidades para utilizar los servicios definidos en la fase anterior.

En (3) se sobre escribe el método que define la clase de componenteUI.

En (4) se sobre escribe el método decode() que se encarga de convertir los valores capturados en las etiquetas que representan al componente en los valores respectivos del componente.

En esta clase se definen los siguientes métodos que son utilizados para llenar los combos en el componente UI.

- getTiposUnidad(FacesContext context, UIComponent component).
- getClasesUnidad (FacesContext context, UIComponent component).
- getFondos(FacesContext context, UIComponent component).
- getUnidadesDirectivas(FacesContext context, UIComponent component)

En figura que aparece a continuación se pueden apreciar un ejemplo de cómo se implementa el método.

Figura 52. Metodo getTiposUnidad

```
public String getTiposUnidad(FacesContext context, UIComponent component){
    List<TipoUnidad> tiposUnidad;
    String opciones="";

    ExternalContext external = context.getExternalContext();
    Map requestParams = external.getRequestParameterMap();
    UIConsultarUnidades uiUnidad = (UIConsultarUnidades)component;
    consultaUnidades = (ConsultarUnidades)Component.getInstance("consultaUnidades");

    try {
        tiposUnidad = consultaUnidades.getTiposUnidad();
        for(TipoUnidad tipoUnidad: tiposUnidad){
            opciones += "<option value='"+ tipoUnidad.getCodigo() + "'>"
                     +  tipoUnidad.getDescripcion().trim() + "</option>";
        }
    } catch (Exception e) {
        System.out.println("Error: " + e.toString());
    }
    return opciones;
}
```

htmlConsultarUnidades.jspx

Esta página es la plantilla que se utiliza para generar la clase Renderer, la cual es la clase fundamental para todos los componentes de JSF y se encarga de convertir los valores del componente en etiquetas y viceberza.

Utiliza un lenguaje de Tags, las cuales se convierten en código de la clase Renderer, por este motivo en la versión actual del CDK no permite la inclusión de componentes de Richfaces o JSF, porque son difíciles de traducir en el código respectivo en la clase Renderer; por lo tanto, las herramientas que se utilizan para implementar esta plantilla

son: HTML, JSP, tags, javascript para capturar los eventos, Seam Remoting para procesar los eventos y realizar consultas sobre los EJBs desde javascript por medio de Ajax.

Para obtener una referencia de las tag que se pueden utilizar se puede consultar el documento CDK de richfaces capitulo 11. Para obtener más información sobre la herramienta Seam remoting se puede consultar Seam in Action capítulo 12. En la siguiente figura se puede apreciar la cabecera de la plantilla.

Figura 53. Plantilla htmlConsultarUnidades.jspx.

```
<f:root xmlns:f="http://ajax4jsf.org/cdk/template"
    xmlns:c=" http://java.sun.com/jsf/core"
    xmlns:ui=" http://ajax4jsf.org/cdk/ui"
    xmlns:u=" http://ajax4jsf.org/cdk/u"
    xmlns:x=" http://ajax4jsf.org/cdk/x"
    xmlns:h=" http://ajax4jsf.org/cdk/h"
    xmlns:vcp=" http://ajax4jsf.org/cdk/vcp"
    class="co.edu.uis.recursosHumanos.componentes.renderkit.html.ConsultarUnidadesRenderer"
    baseclass="co.edu.uis.recursosHumanos.componentes.renderkit.ConsultarUnidadesRendererBase"
    component="co.edu.uis.recursosHumanos.componentes.component.UIConsultarUnidades">
    <f:clientid var="clientId" />

    <h:scripts>/co/edu/uis/recursosHumanos/componentes/renderkit/html/scripts/ConsultarUnidades.js</h:scripts>
    <h:styles>/co/edu/uis/recursosHumanos/componentes/renderkit/html/css/estilos.css</h:styles>

    <jsp:scriptlet><![CDATA[
        String scripts = "<script type='text/javascript' src='../seam/resource/remoting/resource/remote.js'"
                       + "></script>";
        scripts += "<script type='text/javascript' "
                + "src='../seam/resource/remoting/interface.js?ConsultarUnidades'></script>";
        scripts += "<script type='text/javascript'>var cliente ='" + component.getClientId(context) + "';"
                + "var vectorDatos = new Array();</script>";
        context.getResponseWriter().write(scripts);]]>
    </jsp:scriptlet>
```

En (1) se especifican las cabeceras de la pagina que son inclusiones a las librerías de las etiquetas y tags que se pueden utilizar, también se define el nombre de la clase que se va a generar: ConsultarUnidadesRenderer, el nombre de la clase base para el renderer: ConsultarUnidadesRendererBase y la clase base del componente: UIConsultarUnidades.
En (2) se incluyen los archivos javascript que procesan los eventos y el archivo de estilos.
En (3) se incluye la referencia a los archivos javascript que permiten la utilización del Seam Remoting, se incluyen de esta manera porque por medio de la etiqueta <h:scripts> no funciona correctamente y la pagina donde se utiliza el componente no los reconoce.

En (4) se utiliza un método del contexto para escribir datos en la página donde se utilice el componente.

A continuación se presenta el código para generar un cuadro de texto con un icono al lado que al ser pulsado despliega los criterios de consulta.

Figura 54. Creación de un cuadro de texto y un icono.

```
<div id="#{clientId}divInput" x:passThruWithExclusions="style,name,id,mensaje,value">
    <f:resource name="/co/edu/uis/recursosHumanos/componentes/renderkit/html/iconimages/icon.gif" var="icon"/>
    <table width="400" border="0" align="left" cellpadding="2">
        <tr>
            <td valign="middle" align="left">
                <input
                    id="#{clientId}"
                    name="#{clientId}"
                    type="text"
                    value="#{this:getValorString(context, component)}"
                    class="#{component.attributes['inputClass']}"
                    style="#{component.attributes['inputStyle']}"
                    size="60"
                 />
                 <a href="#" onClick="abrirDetalle('#{clientId}')">
                    <img src="#{icon}" id="#{clientId}boton" width="15" height="15" align="absmiddle"/>
                 </a>
            </td>
        </tr>
    </table>
</div>
```

Se puede apreciar el input con sus propiedades tradicionales, las cuales son asignadas a valores del componente. El evento del icono se maneja por medio de javaScript y se pasa como parámetro la identificación del cliente, la cual es única para todas las instancias del componente. El código javascript utilizado es:

Figura 55. Código JavaScript que maneja el evento onClick del icono.

```
function abrirDetalle(idCliente){
    cliente = idCliente;
    var div= cliente + "divConsulta";
    var boton = cliente + "divInput";
    var top =  posicionY(document.getElementById(boton)) + 25;
    var left = posicionX(document.getElementById(boton))+ 5;
    var divEncabezado = idCliente + "divEncabezado";

    document.getElementById(div).style.top = top + 'px';
    document.getElementById(div).style.left = left + 'px';
    document.getElementById(div).style.visibility ="visible";
}
```

Se puede apreciar que el código javascript es sencillo y como se utiliza tradicionalmente en cualquier aplicación web. Se define en el archivo ConsultarUnidades.js. A continuación

se presenta el código necesario para crear un combo y manejarle el evento onChange, el cual debe ir al servicio y consultar las unidades según el criterio seleccionado.

Figura 56. Generación de un select.

```
<tr>
    <td width="110" align="left" valign="middle" class="#{component.attributes['etiquetasClass']}">
        Tipo Unidad:
    </td>
    <td width="410" valign="middle" align="left" class="#{component.attributes['datosClass']}">
        <select name="#{clientId}TipoUnidad" id="#{clientId}TipoUnidad"
                onchange="buscarUnidades('#{clientId}')"
                class="#{component.attributes['combosClass']}">
            <option value="-1" selected="selected">Seleccione Tipo...</option>
            <jsp:scriptlet><![CDATA[
                    context.getResponseWriter().write(getTiposUnidad(context, component));
            ]]></jsp:scriptlet>
        </select>
    </td>
</tr>
```

Se puede apreciar que el evento onChange llama a un método javascript buscarUnidades con el parámetro de identificación del cliente. El código javaScript del método se presenta en la figura que aparece a continuación.

Se puede resaltar el llamado que se hace desde javaScript a un método del EJB ConsultarUnidades y el resultado que debe ser un String con el listado de las unidades se procesa por medio de la función mostrarUnidades(unidades), donde la variable unidades contiene el resultado de la consulta. En el método mostrarUnidades se genera una tabla html donde se pintan los resultados y se actualiza un div.

Figura 57. Llamado al método de consultar unidades del EJB desde JavaScript.

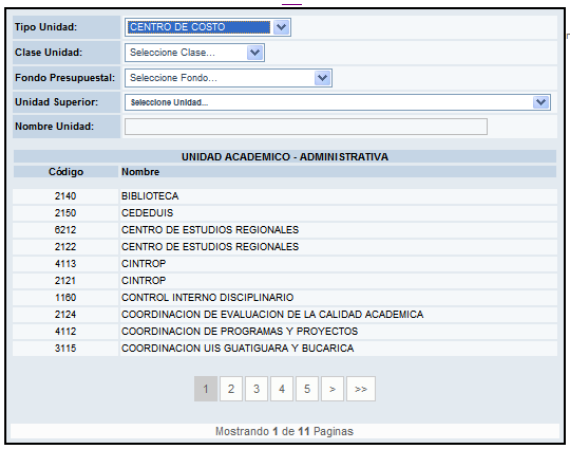
```
function buscarUnidades(clientId){
    cliente = clientId;
    var tipo = document.getElementById(clientId + "TipoUnidad").value;
    var clase = document.getElementById(clientId + "ClaseUnidad").value;
    var fondo = document.getElementById(clientId + "Fondo").value;
    var superior = document.getElementById(clientId + "Superior").value;
    var nombre = document.getElementById(clientId + "Nombre").value;
    Seam.Component.getInstance("ConsultarUnidades").getUnidades(tipo, clase, fondo,
            superior,nombre,mostrarUnidades);
}

function mostrarUnidades(unidades) {
    ...
}
```

El componente generado se puede ver en la figura siguiente. Al incluir la etiqueta <uis:ConsultarUnidades> </uis:ConsultarUnidades> en la página se muestra:

Figura 58. Componente UI: ConsultarUnidades.

Al pulsar sobre el icono se muestran los criterios de consulta, al seleccionar cualquier criterio, se consulta la base de datos y se trae le información correspondiente como se puede apreciar en la siguiente figura.

Figura 59. Componente UI al pulsar el icono.



5.4.8. PRUEBAS COMPONENTES UI.

Para realizar las pruebas del componente personalizado para la interfaz de usuario se utilizó la aplicación de prueba creada en la prueba de las entidades. Se creó otra página de prueba y se realizaron las pruebas mostradas en la siguiente tabla.

Tabla 19. Pruebas de los componentes UI.

PRUEBAS DE LOS COMPONENTES UI		
PRUEBA	**Métodos probados**	**Resultado**
Propiedades: Verificar que las	Id	OK

propiedades del componente UI se asignan y funcionan correctamente.	name	OK
	value	OK
	mensaje	OK
	style	OK
	titulo	OK
	inputStyle	OK
	inputClass	OK
	tituloClass	OK
	datosClass	OK
	combosClass	OK
	etiquetasClass	OK
	verTipoUnidad	OK
	verClaseUnidad	OK
	verFondo	OK
	verUnidadSuperior	OK
	verNombre	OK
Prueba de Métodos de Consulta: Se probaron los método de consultar y generación de tipos.	getTiposUnidad()	OK
	getClasesUnidad()	OK
	getFondos()	OK
	getUnidadesDirectivas()	OK
	consultarUnidades()	OK
Prueba de Utilización: Se crearon varias instancias del componente, añadiendo varias etiquetas en la misma página, se verifico que se asignaran al cuadro de texto correspondiente y funcionara correctamente, se verifico que los resultados de la consulta fueran los correctos.		OK

Con las pruebas realizadas se puede garantizar que el componente funciona correctamente y se puede comenzara a utilizar en los proyectos de desarrollo.

5.4.9. ESPECIFICACIÓN DEL COMPONENTE.

Para la especificación del componente software reutilizable se utilizó el procedimiento planteado en la sección 4.4.9. Por lo tanto para realizar la especificación del componente se proceda a generar los javaDocs e incluirlos en una carpeta llamada especificación, junto con los modelos UML elaborados para el componente y la ficha de especificación, la

cual para el componente implementado se presenta en la tabla que aparece a continuación.

Tabla 20. Ficha de especificación del componente reutilizable Unidades.

FICHA DE ESPECIFICACIÓN DEL COMPONENTE					
NOMBRE:	Unidades	**FECHA:**	Agosto 10 2009	**VERSIÓN:**	1.0
DESCRIPCIÓN:	Componente reutilizable que implementa la consulta de unidades académicas y administrativas de la universidad por los siguientes criterios de consulta: tipo de unidad, clase de unidad, nombre, unidad superior, nombre de la unidad y fondo presupuestal.				
PALABRAS CLAVE:	Consultar unidades, consultar dependencias.				
ESPECIFICACIÓN DE IMPLEMENTACIÓN					
TECNOLOGIA:	jdk1.5, Java ee 5, richfaces 3.3.0.GA, ejb 3.0, seam 2.0.0.GA.				
INTERFAZ:	ConsultarUnidades.java				
SERVICIOS:	Interfaz ConsutarUnidades define los siguiente servicios accedidos vía EJB: • getUnidades(Unidad aCriteriosUnidad) • getUnidad(Integer aCodigoUnidad) • getUnidades(ClaseUnidad aClaseUnidad) • getUnidades(FondoPresupuestal aFondo) • getUnidades(TipoUnidad aTipoUnidad) • getUnidades(TipoCostoUnidad aTipoCosto) • consultarUnidadesACargo(Unidad aUnidadSuperior) • consultarUnidadesDelPrograma(Unidad aPrograma) • getUnidades(); • setUnidades(List<Unidad> unidades); • getClasesUnidad() • setClasesUnidad(List<ClaseUnidad> clasesUnidad); • getFondosPresupuestales() • setFondosPresupuestales(List<FondoPresupuestal> fondosPresupuestales); • getTiposUnidad() • setTiposUnidad(List<TipoUnidad> tiposUnidad); • getTiposCostoUnidad(); • setTiposCostoUnidad(List<TipoCostoUnidad> tiposCostoUnidad); • getUnidades(String aCodigoTipo, String aCodigoClase, String aFondo, String aUnidadSuperior, String aNombre); • getUnidadesDirectivas(); Por medio de servicio web se pueden utilizar los siguientes servicios: • consultarUnidades(String aCodigoTipo, String aCodigoClase, String aFondo, String aUnidadSuperior, String aNombre). Componentes personalizados para la interfaz de Usuario: • ConsultarUnidades: xmlns:uis=*https://www.uis.edu.co/ConsultarUnidades,* su etiqueta es: <uis:ConsultarUnidades></ConsultarUnidaes>				

PRECONDICIONES:	Ninguna.
POSTCONDICIONES:	Ninguna.
DEPENDENCIAS:	Para que funcione el componente correctamente, más específicamente la librería jboss seam remoting y conozca los servicios, se debe referenciar el jar que contiene los servicios en el archivo aplication.xml del proyecto donde se vaya a implementar, en la sección de módulos.
ESPECIFICACIÓN NO FUNCIONAL	
RECURSOS:	Archivo de definición del origen de datos (dataSource), el cual debe ir en la carpeta de distribución del componente. Se incluye en el componente.
RESTRICCIONES:	Ninguna.
NIVEL DE SEGURIDAD:	Medio.
RENDIMIENTO:	Ninguna recomendación.
PORTABILIDAD:	Funciona en aplicaciones Java que soporten EJB 3.0, JSF, JSP y richfaces. En otras plataformas funciona utilizando los servicios web implementados.
GRADO DE REUTILIZACIÓN:	34
DISTRIBUCIÓN:	Se deben instalar los archivos *.jar en la carpeta lib del proyecto o en la parte donde se referencien las librerías externas a la aplicación, también deben ir estos archivos en la carpeta lib del servidor de aplicaciones. Los archivos *.ear deben ir la carpeta de distribución de aplicaciones del servidor, junto con el archivo *-ds.xml.

5.5. DISTRIBUCIÓN DEL COMPONENTE.

Los resultados de la implementación del componente fueron:

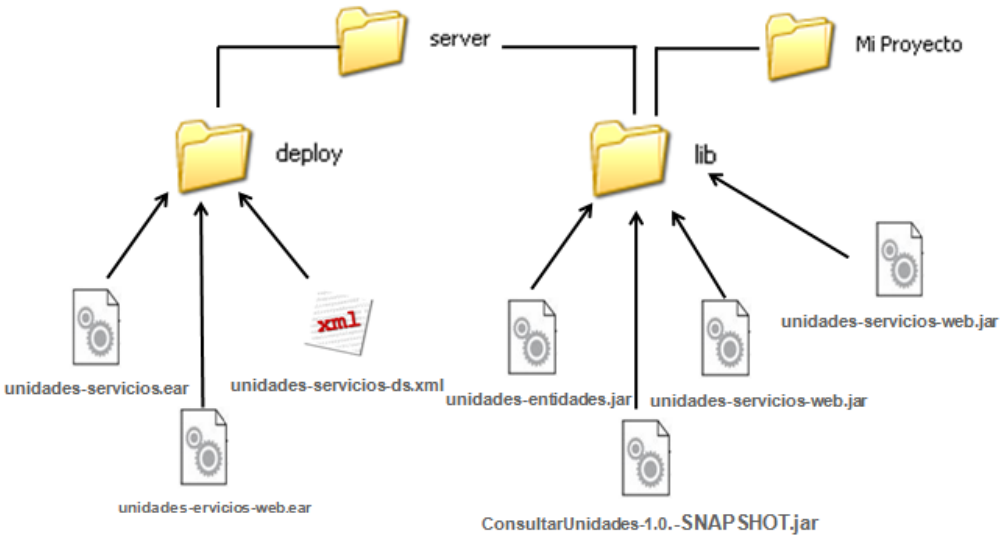
- Archivo unidades-entidades.jar.
- Archivo unidades-servicios.jar, unidades-servicios.ear
- Archivo unidades-ds.xml
- Archivo unidades-servicios-web.jar, unidades-servicios-web.ear
- Archivo ConsultarUnidades-1.0-SNAPSHOT.jar
- Ficha de especificación.
- JavaDocs
- Modelos UML de la implementación del componente.

Se deben empaquetar estos elementos en un archivo .zip o .tar para ser distribuidos en el servidor de aplicaciones de la DSI. De allí todos los desarrolladores pueden bajar el componente y utilizarlo en sus proyectos de desarrollo. En la siguiente figura se muestra

el esquema de distribución en el servidor del componente y en el proyecto de desarrollo donde se va a utilizar.

Para hacer la instalación del componentes se incluye en el archivo .jar un archivo build.xml de tareas de la herramienta ANT que se encarga de copiar y registrar el componente software reutilizable en el servidor de aplicaciones.

Figura 60. Esquema de distribución del componente.



6. CONCLUSIONES

Con la elaboración de componentes software reutilizables se puede lograr la reutilización de estos en muchos proyectos de desarrollo. La reutilización de software trae beneficios muy importantes para la actividad de desarrollo profesional de software, ya que permite reutilizar elementos ya elaborados, haciendo así más rápida su implementación. Los beneficios más importantes de la reutilización de software propuestos por Meyer[44] son:

- **Oportunidad**: Se tiene menos software que hacer y se puede construir con mayor rapidez.
- **Mantenimiento**: Disminución de los esfuerzos de mantenimiento.
- **Fiabilidad**: Al tener buenos componentes ya usados y probados con anterioridad se puede mejorar el desarrollo de nuevos sistemas.
- **Eficiencia**: Los componentes en muchas ocasiones son desarrollados por expertos en el dominio que se desarrollaron, usando los mejores algoritmos, patrones y estructuras de datos.
- **Consistencia**: Los componentes deben tener un énfasis estricto en un diseño regular y coherente, ya que el estilo del componente influye en el estilo del sistema desarrollado.
- **Inversión**: El componente no se desarrolla solo para un proyecto, tiene la oportunidad de utilizarlo en muchos proyectos. Se puede preservar en el componente el conocimiento de los mejores desarrolladores.

Con la elaboración del proceso de desarrollo de componentes software reutilizables se beneficiarían principalmente las empresas desarrolladoras de software empresarial, en el contexto de este trabajo, la División de Servicios de Información de la Universidad, dispone ahora de una forma más detallada, basada en el estándar Java EE 5, para elaborar componentes software reutilizables que se puedan usar en el desarrollo de los sistemas empresariales.

[44] MEYER, Bertrand. Construcción de Software Orientado A Objetos. Prentice-hall 2002.

Por medio del modelo de selección de componentes se pueden identificar los componentes candidatos a implementar para un dominio de aplicación particular, asegurando por medio del análisis de reutilización la posibilidad de utilizarse en varios proyectos de desarrollo software.

Se definieron las alternativas de arquitectura para la implementación de componentes software reutilizables, se hizo una comparación entre las tres alternativas planteadas: arquitectura por capas, arquitectura MVC y arquitectura orientada a servicios. Se tienen los fundamentos base para decidir que arquitectura se puede utilizar en la implementación de un componente específico.

Con la presente investigación se plantea un procedimiento para el desarrollo, prueba y especificación de componentes basados en el estándar Java EE 5, se detalla la forma de implementar el mapeo de las entidades, la elaboración de los servicios del área de aplicación utilizando EJBs 3.0, exposición de esos servicios como servicios web y componentes personalizados para la interfaz de usuario. Con los elementos mencionados, se puede construir cualquier software empresarial, sin la necesidad de ser basado en componentes, por lo tanto se tiene una guía para realizar software empresarial en la versión Java EE 5.

La utilización de componentes personalizados para la interfaz de usuario se reduce la cantidad de código que tiene que generar un programador para realizar una consulta y selección de un elemento, con solo un par de líneas y un archivo de configuración se puede traer la consulta.

Se definió el esquema de distribución y utilización del componente software reutilizable, permitiendo así la utilización por parte de los desarrolladores en los proyectos software de la División de Servicios de Información.

Con el desarrollo de los componentes software reutilizables se garantiza la reutilización de lógica del negocio de la empresa de desarrollo de software en muchos proyectos, el componente no se centra sólo en resolver un problema especifico para la interfaz de

usuario, sino que por el contrario se puede reutilizar un conjunto de requisitos software en varios proyectos de desarrollo.

El proceso de desarrollo de componentes propuesto esta correcto y completo, se probó en el desarrollo de software empresarial en la División de servicios de información y funciona de forma adecuada, las empresas de desarrollo de software pueden acoger esta propuesta para el desarrollo de componentes software que garantizan la reutilización de lógica del negocio en muchas aplicaciones.

7. RECOMENDACIONES

- Se recomienda probar el proceso de desarrollo de componentes software reutilizables en la elaboración de varios componentes más complejos, para refinar aún más el proceso y garantizar que se puede utilizar en cualquier caso.

- Se recomienda elaborar un sistema para la catalogación y consulta de los componentes reutilizables que se van desarrollando, para que los desarrolladores de software los puedan utilizar e identificar de forma rápida.

- Es importante aclarar y definir el proceso de desarrollo software basado en componentes, identificando claramente las etapas de selección, evaluación, adaptación y ensamblaje de los compones en el sistema informático a desarrollar.

8. TRABAJO FUTURO

Con el presente trabajo se abre una línea de investigación interesante y prometedora, la cual no ha sido explorada en la Universidad, el trabajo realizado en conjunto con la División de Servicios de información abre varios campos y trabajos pendientes de realizar:

- Elaboración de un sistema de catalogación y consulta por diferentes parámetros de los componentes software reutilizables ya elaborados para que el desarrollador los pueda utilizar.

- En el proyecto de desarrollo de las nuevas versiones de los sistemas de información de la universidad se requiere la elaboración de un gran número de componentes reutilizables, medio por el cual se puede refinar el proceso de desarrollo planteado.

- Es importante profundizar y detallar mejor el proceso de desarrollo de software basado en componentes, identificando las actividades que se deben realizar en cada una de las etapas que se deben llevar a cabo para la elaboración de software empresarial basado en componentes ya elaborados.

BIBLIOGRAFIA

PRESSMAN, Roger S. Ingeniería del Software un enfoque práctico. Quinta Edición. Mc Graw Hill. 2002.

SZYPERSKY, C. Component Software. Beyond Object-Oriented Programming. Addison-Wesley. 1998.

MEYER, Bertrand. The Significance of Components. Beyond Objects column, Software Development. 1999.

SUN MICROSYSTEM. Tutorial: Object-Oriented Analysis and Design Using UML. Copyright 2003.

IRIBARNE MARTÍNEZ, Luis F. Un Modelo de Mediación para el Desarrollo de Software basado en Componentes COTS. Tesis Doctoral. Universidad de Málaga. España. 2003.

SCHNEIDER, Jean Guy y HAN, Jun. Artículo: Components – the Past, the Present ,and the Future. School of Information Technology, Swinburne University of Technology. Hawthorn, Victoria, Australia.

SUN MICROSYSTEM, The Java EE5 tutorial. 2008

WEITZENFELD, Alfredo. Ingeniería de Software Orientada a Objetos, Teoría y Práctica con UML y Java. México. Itam. 2002.

BRUEGGE, Bernd y DUTOIT, Allen H. Object-Oriented Software Engineering Using UML, Patterns and Java. 2ª Edición. Prentice Hall. 2004.

FRANKY, Maria Consuelo. Curso: Desarrollo de aplicaciones en Java EE 5. CincoSOFT LTDA. 2007.

TORO, Víctor Manuel. Conferencia: Panorama sobre la Ingeniería del Software. CincoSOFT LTDA. 2007.

MEYER, Bertrand. Construcción de Software Orientado A Objetos. Prentice-hall 2002.

RUMBAUCH, James; JACOBSON, Ivar y BOOCH, Grady. El lenguaje unificado de modelado. Manual de referencia. Addison Wesley. Madrid – España. 2000.

DRAKE, José M; MEDINA, Julio Luís y GONZALEZ HARBOUR, Michael. Artículo: Entorno para el Diseño de Sistemas Basados en Componentes de Tiempo Real. Grupo de Computadores y Tiempo Real. Universidad de Cantabria. España.

BERTOA, Manuel F. y VALLECILLO, Antonio. Calidad de Componentes Software. Departamento de Lenguajes y Ciencias de la Computación. Universidad de Málaga. 2004

DEBU, Panda; REZA, Rahman y DEREL Lane. EJB in Action. Manning Publications Co. 2007.

GUIJUN, Wang; CASEY K, Fung. Artículo: Architecture Paradigms and Their Influences and Impacts on Component-Based Software Systems. Proceedings of the 37th Hawaii International Conference on System Sciences – 2004

SERGEY, Smirnov. Component Development Kit of RichFaces. Jboss Comunitiy. 2008

KITO D. Mann. Java Server Faces in action. Manning Publications. 2008

Tracz, W; Where Does Reuse Start? Proc. Realities of Reuse Workshop, Syracuse University CASE Center, Enero de 1990.

GEISTERFER, CJ Michael; GHOSH, Sudipto. Artículo: Software component Specification: A Study in Perspective of Component Selection and Reuse. Proceedings of the Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems. IEEE 2006.

DAN, Allen; Seam in Action. Manning Publications. 2008

Nota: Los diagramas UML presentados fueron elaborados por medio de la herramienta software Enterprise Architect de Sparx System, licenciada por la División de Servicios de Información de la UIS.