CONSTRUCCIÓN DE UNA APLICACIÓN WEB PARA LA ELABORACIÓN DE DIAGRAMAS DE CLASE, ORIENTADA BAJO EL MODELO DE SERVICIOS DE INTERNET CLOUD COMPUTING

(Proyecto de Grado Dirigido)

GABRIEL LEONARDO DÍAZ CÁRDENAS

UNIVERSIDAD FRANCISCO DE PAULA SANTANDER

FACULTAD DE INGENIERIAS

PLAN DE ESTUDIOS DE INGENIERIA DE SISTEMAS

SAN JOSÉ DE CÚCUTA

2013

CONSTRUCCIÓN DE UNA APLICACIÓN WEB PARA LA ELABORACIÓN DE DIAGRAMAS DE CLASE, ORIENTADA BAJO EL MODELO DE SERVICIOS DE INTERNET CLOUD COMPUTING.

GABRIEL LEONARDO DÍAZ CÁRDENAS

Código 0152685

Trabajo de grado para ostentar el título de Ingeniero de Sistemas de la Universidad Francisco de Paula Santander

DIRECTOR

MARCO ANTONIO ADARME JAIMES

INGENIERO DE SISTEMAS

MAGISTER EN CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD FRANCISCO DE PAULA SANTANDER

FACULTAD DE INGENIERIAS

PLAN DE ESTUDIOS DE INGENIERIA DE SISTEMAS

SAN JOSÉ DE CÚCUTA

2013

**TABLA DE CONTENIDO**

[LISTADO DE FIGURAS 8](#_Toc368845428)

[LISTADO DE DIAGRAMAS 10](#_Toc368845429)

[LISTADO DE TABLAS 11](#_Toc368845430)

[INTRODUCCIÓN 12](#_Toc368845431)

[1. PRESENTACIÓN DEL ANTEPROYECTO 13](#_Toc368845432)

[1.1 TITULO 13](#_Toc368845433)

[1.2 PLANTEAMIENTO DEL PROBLEMA 13](#_Toc368845434)

[1.3 JUSTIFICACIÓN 14](#_Toc368845435)

[1.4 OBJETIVOS 16](#_Toc368845436)

[1.4.1 Objetivo General 16](#_Toc368845437)

[1.4.2 Objetivos Específicos 16](#_Toc368845438)

[1.5 ALCANCES Y DELIMITACIONES 17](#_Toc368845439)

[1.5.1 Alcance 17](#_Toc368845440)

[1.5.2 Delimitaciones 17](#_Toc368845441)

[2. MARCO REFERENCIAL 19](#_Toc368845442)

[2.1 MARCO DE ANTECEDENTES 19](#_Toc368845443)

[2.1.1 Draw.io 19](#_Toc368845444)

[2.1.2 Creately 19](#_Toc368845445)

[2.1.3 GWTUML 20](#_Toc368845446)

[2.1.4 JoinJS 20](#_Toc368845447)

[2.2 MARCO CONCEPTUAL 21](#_Toc368845448)

[2.2.1 Herramientas CASE 21](#_Toc368845449)

[2.2.2 UML 23](#_Toc368845450)

[2.2.3 Diagramas de Clase 24](#_Toc368845451)

[2.2.4 Meta-Programación 25](#_Toc368845452)

[2.2.5 Cloud Computing 26](#_Toc368845453)

[2.3 MARCO TEORICO 28](#_Toc368845454)

[2.3.1 Java 28](#_Toc368845455)

[2.3.2 Frameworks 29](#_Toc368845456)

[2.3.3 Servidor de Aplicaciones 31](#_Toc368845457)

[2.3.4 Extreme Programming (XP) 32](#_Toc368845458)

[2.3.5 SVG 34](#_Toc368845459)

[2.3.6 HTML, JavaScript y CSS 35](#_Toc368845460)

[2.4 MARCO DE LEGAL 36](#_Toc368845461)

[2.4.1 Acuerdos de uso 36](#_Toc368845462)

[2.4.2 Contrato de licencia de código binario, SUN MICROSYSTEMS 36](#_Toc368845463)

[2.4.3 GNU General Public Licence 37](#_Toc368845464)

[2.4.4 Propiedad Intelectual 37](#_Toc368845465)

[3. DISEÑO METODOLÓGICO 38](#_Toc368845466)

[3.1 TIPO DE INVESTIGACIÓN 38](#_Toc368845467)

[3.2 FUENTES DE INFORMACIÓN 38](#_Toc368845468)

[3.2.1 Fuentes de información primaria 38](#_Toc368845469)

[3.2.2 Fuentes de información secundaria 38](#_Toc368845470)

[3.3 RECOLECCIÓN Y ANALISIS DE INFORMACIÓN 39](#_Toc368845471)

[4. ANALISIS DE LAS HERRAMIENTAS CASE 41](#_Toc368845472)

[4.1 ENTERPRISE ARCHITECT 41](#_Toc368845473)

[4.1.1 Historia 42](#_Toc368845474)

[4.1.2 Características Generales 43](#_Toc368845475)

[4.1.3 Funciones para Diagramas de Clase 44](#_Toc368845476)

[4.2 STARUML 60](#_Toc368845477)

[4.2.1 Historia 60](#_Toc368845478)

[4.2.2 Características Generales 61](#_Toc368845479)

[4.2.3 Funciones para Diagramas de Clase 62](#_Toc368845480)

[4.3 ARGOUML 70](#_Toc368845481)

[4.3.1 Funciones para Diagramas de Clase 70](#_Toc368845482)

[5. DIAGRAMAS DE CLASES 73](#_Toc368845483)

[5.1 CLASIFICADORES 74](#_Toc368845484)

[5.1.1 Clase 75](#_Toc368845485)

[5.1.2 Atributos 77](#_Toc368845486)

[5.1.3 Operaciones 78](#_Toc368845487)

[5.1.4 Interfaz 80](#_Toc368845488)

[5.1.5 Enumeración 81](#_Toc368845489)

[5.1.6 Paquete 82](#_Toc368845490)

[5.2 RELACIONES 83](#_Toc368845491)

[5.2.1 Asociación 83](#_Toc368845492)

[5.2.2 Generalización 85](#_Toc368845493)

[5.2.3 Realización 86](#_Toc368845494)

[5.2.4 Agregación 88](#_Toc368845495)

[5.2.5 Composición 90](#_Toc368845496)

[5.2.6 Clase Asociación 93](#_Toc368845497)

[5.2.7 Dependencia 93](#_Toc368845498)

[5.3 CONCEPTOS AVANZADOS 94](#_Toc368845499)

[5.3.1 Multiplicidad 94](#_Toc368845500)

[5.3.2 Navegabilidad 95](#_Toc368845501)

[5.3.3 Visibilidad 95](#_Toc368845502)

[5.3.4 Roles 95](#_Toc368845503)

[6. CLASIFICACIÓN DE LA INFORMACIÓN 96](#_Toc368845504)

[6.1 MATRIZ DE CARACTERISTICAS 96](#_Toc368845505)

[6.2 CARACTERISTICAS PROPIAS 97](#_Toc368845506)

[6.3 LISTA DE FUNCIONALIDADES 97](#_Toc368845507)

[7. METODOLOGÍA DE DESARROLLO 98](#_Toc368845508)

[7.1 EXPLORACIÓN 98](#_Toc368845509)

[7.1.1 Identificación de Roles del Proyecto 98](#_Toc368845510)

[7.1.2 Identificación de Actores del Sistema 99](#_Toc368845511)

[7.1.3 Historias de Usuario 102](#_Toc368845512)

[7.1.4 Arquitectura de la Aplicación 112](#_Toc368845513)

[7.1.5 Definición de la Tecnología 115](#_Toc368845514)

[7.1.5.1 JSF 115](#_Toc368845515)

[7.1.5.2 Facelets 117](#_Toc368845516)

[7.1.5.3 PrimeFaces 117](#_Toc368845517)

[7.1.5.4 EJB 118](#_Toc368845518)

[7.1.5.5 Java Persistence API 119](#_Toc368845519)

[7.1.5.6 MxGraph 120](#_Toc368845520)

[7.1.5.7 Glassfish 122](#_Toc368845521)

[7.1.5.8 MySQL 122](#_Toc368845522)

[7.1.5.9 Eclipse 123](#_Toc368845523)

[7.2 PLANIFICACIÓN 123](#_Toc368845524)

[7.2.1 Priorización de Historias de Usuario 123](#_Toc368845525)

[7.2.2 Estimación de Historias de Usuario 125](#_Toc368845526)

[7.2.3 Definición de Iteraciones 125](#_Toc368845527)

[7.2.4 Metáfora del Sistema 126](#_Toc368845528)

[7.3 IMPLEMENTACIÓN 127](#_Toc368845529)

[7.3.1 Patrones de Desarrollo 127](#_Toc368845530)

[7.3.1.1 Modelo Vista Controlador 127](#_Toc368845531)

[7.3.1.2 Intercepting Filter 128](#_Toc368845532)

[7.3.1.3 Front Controller 129](#_Toc368845533)

[7.3.1.4 Session Facade 130](#_Toc368845534)

[7.3.1.5 Singleton 131](#_Toc368845535)

[7.3.1.6 Data Access Object 132](#_Toc368845536)

[7.3.2 Estructura del Código del Proyecto 133](#_Toc368845537)

[7.3.3 Interfaz Gráfica de Usuario 134](#_Toc368845538)

[7.3.4 Iteración 1 137](#_Toc368845539)

[7.3.4.1 Actividades a Realizar 137](#_Toc368845540)

[7.3.4.2 Diagrama de Clases 138](#_Toc368845541)

[7.3.4.3 Modelo de Datos 140](#_Toc368845542)

[7.3.4.4 Interfaz grafica 143](#_Toc368845543)

[7.3.4.5 Controles 145](#_Toc368845544)

[7.3.4.6 Servicios 145](#_Toc368845545)

[7.3.4.7 Pruebas de unidad 156](#_Toc368845546)

[7.3.5 Iteración 2 159](#_Toc368845547)

[7.3.5.1 Actividades a Realizar 159](#_Toc368845548)

[7.3.5.2 Diagrama de Clases 159](#_Toc368845549)

[7.3.5.3 Modelo de Datos 160](#_Toc368845550)

[7.3.5.4 Interfaz grafica 161](#_Toc368845551)

[7.3.5.5 Seguridad 163](#_Toc368845552)

[7.3.5.6 Servicios 165](#_Toc368845553)

[7.3.5.7 Pruebas Unitarias 167](#_Toc368845554)

[7.3.6 Iteración 3 167](#_Toc368845555)

[7.3.7 Iteración 4 167](#_Toc368845556)

[7.3.8 Iteración 5 167](#_Toc368845557)

[7.3.9 Iteración 6 167](#_Toc368845558)

[7.3.10 Iteración 7 167](#_Toc368845559)

[7.3.11 Iteración 8 167](#_Toc368845560)

[7.3.12 Iteración 9 167](#_Toc368845561)

[7.4 DESPLIEGUE 167](#_Toc368845562)

[CONCLUSIONES 169](#_Toc368845563)

[RECOMENDACIONES 170](#_Toc368845564)

[BIBLIOGRAFIA 171](#_Toc368845565)

# LISTADO DE FIGURAS

[Figura 1. Jerarquía de los Diagramas de UML 2.0 [4] 24](#_Toc368915703)

[Figura 2. Ejemplo de diagrama de clases con Enterprise Architect 45](#_Toc368915704)

[Figura 3. Caja de herramientas para diagramas de clase de Enterprise Architect 46](#_Toc368915705)

[Figura 4. Explorador del proyecto de Enterprise Architect 47](#_Toc368915706)

[Figura 5. Panel de propiedades de Enterprise Architect 48](#_Toc368915707)

[Figura 6. Elementos de diagramas de clase de Enterprise Architect 48](#_Toc368915708)

[Figura 7. Menú contextual de una clase en Enterprise Architect 50](#_Toc368915709)

[Figura 8. Representación de una clase en Enterprise Architect 51](#_Toc368915710)

[Figura 9. Ventana de creación de atributos para una clase en Enterprise Architect 51](#_Toc368915711)

[Figura 10. Ventana de creación de operaciones para clases en Enterprise Architect 52](#_Toc368915712)

[Figura 11. Ventana de generación de código en Enterprise Architect 54](#_Toc368915713)

[Figura 12. Ventana de opciones de disposición de Enterprise Architect 56](#_Toc368915714)

[Figura 13. Ventana de importación de archivos XMI en Enterprise Architect 59](#_Toc368915715)

[Figura 14. Ventana de exportación de archivos XMI en Enterprise Architect 60](#_Toc368915716)

[Figura 15. Caja de herramientas para diagramas de clase en StarUML 63](#_Toc368915717)

[Figura 16. Distribución de los paneles principales en StarUML 64](#_Toc368915718)

[Figura 17. Menú contextual de una clase en StarUML 65](#_Toc368915719)

[Figura 18. Menú contextual para formato de elementos en StarUML 66](#_Toc368915720)

[Figura 19. Asistente para generación de código Java en StarUML 67](#_Toc368915721)

[Figura 20. Primer paso del asistente de ingeniería inversa en StarUML 68](#_Toc368915722)

[Figura 21. Ultimo paso del asistente de ingeniería inversa en StarUML 68](#_Toc368915723)

[Figura 22. Ventana de importación de archivos XMI en StarUML 69](#_Toc368915724)

[Figura 23. Ventana de exportación de archivos XMI en StarUML 69](#_Toc368915725)

[Figura 24. Distribución de los paneles principales de ArgoUML 71](#_Toc368915726)

[Figura 25. Ventana de generación de imágenes en ArgoUML 71](#_Toc368915727)

[Figura 26. Ventana de exportación XMI de ArgoUML 72](#_Toc368915728)

[Figura 27. Diagrama de clasificadores del meta modelo UML [13] 74](#_Toc368915729)

[Figura 28. Notación UML para una clase 75](#_Toc368915730)

[Figura 29. Paquete de clases del meta modelo UML [13] 76](#_Toc368915731)

[Figura 30. Notación UML para una clase con atributos 77](#_Toc368915732)

[Figura 31. Notación UML de una clase con atributos y operaciones 78](#_Toc368915733)

[Figura 32. Notación UML para una interfaz 80](#_Toc368915734)

[Figura 33. Notación UML para una enumeración 81](#_Toc368915735)

[Figura 34. Notación UML para un paquete 82](#_Toc368915736)

[Figura 35. Notación UML para una Asociación simple 84](#_Toc368915737)

[Figura 36. Notación UML para una Generalización 85](#_Toc368915738)

[Figura 37. Notación UML para una Realización 87](#_Toc368915739)

[Figura 38. Notación UML para una Agregación Normal 89](#_Toc368915740)

[Figura 39. Notación UML para una Agregación Compartida 90](#_Toc368915741)

[Figura 40. Notación UML para una Composición 91](#_Toc368915742)

[Figura 41. Notación UML para una Clase Asociación 93](#_Toc368915743)

[Figura 42. Notación UML para una Dependencia 94](#_Toc368915744)

[Figura 43. Arquitectura de la aplicación CLASS Modeler 113](#_Toc368915745)

[Figura 44. Arquitectura de la libreria mxGraph 121](#_Toc368915746)

[Figura 45. Patrón Modelo-Vista-Controlador 128](#_Toc368915747)

[Figura 46. Estructura y distribución del código de la aplicación CLASS Modeler 134](#_Toc368915748)

[Figura 47. Plantillas principales para la interfaz gráfica de la aplicación 135](#_Toc368915749)

[Figura 48. Jerarquía de las páginas basadas en plantillas 136](#_Toc368915750)

[Figura 49. Interfaz gráfica principal de la aplicación CLASS Modeler 143](#_Toc368915751)

[Figura 50. Interfaz gráfica para la creación de cuenta de diagramador 144](#_Toc368915752)

[Figura 51. Interfaz gráfica para solicitud de reasignación de contraseña 144](#_Toc368915753)

[Figura 52. Interfaz gráfica para reasignar la contraseña 145](#_Toc368915754)

[Figura 53. Resultados de las pruebas unitarias del servicio *UserServiceTest* 158](#_Toc368915755)

[Figura 54. Resultados de las pruebas unitarias del servicio *SecurityServiceTest* 158](#_Toc368915756)

[Figura 55. Resultados de las pruebas unitarias del servicio *EmailServiceTest* 158](#_Toc368915757)

[Figura 56. Formulario de inicio de sesión 162](#_Toc368915758)

[Figura 57. Tablero de control para listar los diagramas del usuario 162](#_Toc368915759)

[Figura 58. Formulario de creación de diagramas. 164](#_Toc368915760)

[Figura 59. Diagrama de secuencia de la historia de usuario H9 169](#_Toc368915761)

[Figura 60. Resultados de las pruebas de unidad del servicio SessionService 171](#_Toc368915762)

[Figura 61. Resultados de las pruebas unitarias del servicio DiagramService 171](#_Toc368915763)

# LISTADO DE DIAGRAMAS

[Diagrama 1. Diagrama de actores de la aplicación CLASS Modeler 102](#_Toc368915764)

[Diagrama 2. Diagrama de clases del dominio de la iteración 1 139](#_Toc368915765)

[Diagrama 3. Modelo de datos para la iteración 1 141](#_Toc368915766)

[Diagrama 4. Diagrama de clases de los servicios para la iteración 1 146](#_Toc368915767)

[Diagrama 5. Diagrama de clases de implementacion para servicos de la iteración 1 148](#_Toc368915768)

[Diagrama 6. Diagrama de secuencia para creación de cuenta de diagramador H1 149](#_Toc368915769)

[Diagrama 7. Diagrama de secuencia para activación de cuenta de diagramador H2 151](#_Toc368915770)

[Diagrama 8. Diagrama de secuencia para solicitud de nueva contraseña H3 153](#_Toc368915771)

[Diagrama 9. Diagrama de secuencia para reasignación de contraseña H3 154](#_Toc368915772)

[Diagrama 10. Diagrama de clases de la iteración 2 160](#_Toc368915773)

[Diagrama 11. Modelo de datos de la iteración 2 161](#_Toc368915774)

[Diagrama 12. Proceso de filtro de peticiones hacia las páginas de la aplicación 165](#_Toc368915775)

[Diagrama 13. Diagrama de clases de los servicios para la iteración 2 166](#_Toc368915776)

[Diagrama 14. Diagrama de secuencia de la historia de usuario H6 167](#_Toc368915777)

# LISTADO DE TABLAS

[Tabla 1. Características principales de Enterprise Architect 44](#_Toc368915778)

[Tabla 2. Relaciones para diagramas de clase en Enterprise Architect 53](#_Toc368915779)

[Tabla 3. Características principales de StarUML 62](#_Toc368915780)

[Tabla 4. Tipos de multiplicidad de una relación UML 94](#_Toc368915781)

[Tabla 5. Matriz de características para diagramas de clase 96](#_Toc368915782)

[Tabla 6. Especificación del actor *Visitante* 99](#_Toc368915783)

[Tabla 7. Especificación del actor *Usuario* 100](#_Toc368915784)

[Tabla 8. Especificación del actor *Invitado* 100](#_Toc368915785)

[Tabla 9. Especificación del actor *Diagramador* 101](#_Toc368915786)

[Tabla 10. Formato de historia de usuario H1 103](#_Toc368915787)

[Tabla 11. Formato de historia de usuario H2 104](#_Toc368915788)

[Tabla 12. Formato de historia de usuario H3. 104](#_Toc368915789)

[Tabla 13. Formato de historia de usuario H4 105](#_Toc368915790)

[Tabla 14. Formato de historia de usuario H5 106](#_Toc368915791)

[Tabla 15. Formato de historia de usuario H6 106](#_Toc368915792)

[Tabla 16. Formato de historia de usuario H7 106](#_Toc368915793)

[Tabla 17. Formato de historia de usuario H8 107](#_Toc368915794)

[Tabla 18. Formato de historia de usuario H9 107](#_Toc368915795)

[Tabla 19. Formato de historia de usuario H10 108](#_Toc368915796)

[Tabla 20. Formato de historia de usuario H11 108](#_Toc368915797)

[Tabla 21. Formato de historia de usuario H12 109](#_Toc368915798)

[Tabla 22. Formato de historia de usuario H13 109](#_Toc368915799)

[Tabla 23. Formato de historia de usuario H14 110](#_Toc368915800)

[Tabla 24. Formato de historia de usuario H15 110](#_Toc368915801)

[Tabla 25. Formato de historia de usuario H16 111](#_Toc368915802)

[Tabla 26. Formato de historia de usuario H17 111](#_Toc368915803)

[Tabla 27. Formato de historia de usuario H18 111](#_Toc368915804)

[Tabla 28. Formato de historia de usuario H19 112](#_Toc368915805)

[Tabla 29. Formato de historia de usuario H20 112](#_Toc368915806)

[Tabla 30. Formato de historia de usuario H21 112](#_Toc368915807)

[Tabla 31. Priorización de historias de usuario 124](#_Toc368915808)

[Tabla 32. Estimación de tiempo para historias de usuario 125](#_Toc368915809)

[Tabla 33. Plan de iteraciones 126](#_Toc368915810)

[Tabla 34. Listado de pruebas de unidad para la iteración 1 156](#_Toc368915811)

[Tabla 35. Listado de pruebas de unidad para la iteración 2 171](#_Toc368915812)

# INTRODUCCIÓN

En el desarrollo de software es común usar herramientas que faciliten las tareas complejas y agilicen el trabajo manual; la mayoría o casi todas las empresas dedicadas al desarrollo de software incorporan aplicaciones a su quehacer diario, permitiendo a sus trabajadores llevar a cabo su labor de manera más eficiente, conjunta y automatizada.

Este grupo de aplicaciones son llamadas Herramientas CASE, debido a las siglas tomadas del nombre en inglés “Computer Aided Software Engineering”, traducido al español como Ingeniería de Software asistida por computadora. Estas herramientas son la base para la creación de todas las aplicaciones software que vemos hoy en día y cada una se especializa en diferentes aspectos, dependiendo de la o las fases del proceso de desarrollo que apoyan. Algunas con un grado de complejidad bastante alto, permiten controlar prácticamente todo el ciclo de vida de una aplicación desde el análisis de requerimientos hasta el despliegue y mantenimiento.

Tras un largo camino en la evolución del software y en metodologías de desarrollo, se han incorporado nuevos modelos y tendencias para la construcción de aplicaciones. Debido al alto impacto que tuvo el internet, ha surgido un modelo de negocio en donde se rompe el concepto tradicional de las aplicaciones que residen estáticas en la máquina del usuario y este a su vez compra los derechos para usar en todo momento determinada aplicación; hoy por hoy hablamos de un esquema en el que las aplicaciones se alojan en lugares remotos y los usuarios hacen uso de ellas bajo demanda mediante la red global Internet y pagando únicamente el consumo que se hace de estas (en el caso del Software Propietario). Desde hace varios años llamado Cloud Computing o Computación en la Nube, este modelo ha cambiado la concepción que se tenía anteriormente del uso de aplicaciones software.

Con el desarrollo de este proyecto se pretende incorporar los conceptos mencionados anteriormente en una herramienta software, aprovechando las bondades que brindan las aplicaciones en la nube y resolviendo una necesidad particular de nuestro ambiente universitario. De este modo los estudiantes podrán disponer de una herramienta básica para su formación y que posea las características necesarias para ser considerada una aplicación en la nube.

El objetivo principal del proyecto es el desarrollo de una herramienta para el modelado de Diagramas de Clase UML, que le permita a los usuarios crear y manipular de manera online los componentes básicos de este tipo de diagrama y llegando hasta obtener archivos de código fuente generados a partir de los elementos del mismo.

Llamada CLASS Modeler (Modelador de Clases) debido a su naturaleza y enfoque, esta herramienta pretende solventar una necesidad detectada, además de servir como apoyo al proceso de aprendizaje de los estudiantes. A lo largo de este documento se expondrán todas las técnicas empleadas para su construcción y la metodología usada como guía del proceso de desarrollo.

# PRESENTACIÓN DEL ANTEPROYECTO

## TITULO

CONSTRUCCIÓN DE UNA APLICACIÓN WEB PARA LA ELABORACIÓN DE DIAGRAMAS DE CLASE, ORIENTADA BAJO EL MODELO DE SERVICIOS DE INTERNET CLOUD COMPUTING.

## PLANTEAMIENTO DEL PROBLEMA

En estudios realizados por el grupo de investigación y desarrollo de software (GIDIS) de la Universidad Francisco de Paula Santander, después convertido en un proyecto de grado [1], se ha determinado que unos de los principales factores involucrados en los altos índices de mortalidad y deserción académica en la carrera de Ingeniería de Sistemas de las universidades de Cúcuta, son la falta de dedicación al estudio por parte de los estudiantes, poca comprensión de temas como abstracción y modelamiento de sistemas, y falta de ambientes prácticos donde los estudiantes tuvieran la posibilidad de llevar a cabo actividades para afianzar los conceptos aprendidos en clase. Teniendo esto en cuenta se hace evidente que se necesitan mecanismos para que los estudiantes ejerciten sus habilidades y de esta manera se disminuya esta problemática que ha afectado considerablemente al programa académico.

Dentro del programa de Ingeniería de Sistemas de la UFPS se tiene un clara tendencia e inclinación hacia el desarrollo de software, es claro para muchos estudiantes que este es uno de los puntos fuertes de la carrera y es por ello que muchos continúan su vida profesional por ese camino; por tal motivo desde los primeros semestres se trata de cultivar estas habilidades en ellos; considerando que el modelamiento y abstracción de sistemas son temas de vital importancia, se hace necesario disponer de herramientas automáticas (software) que permitan adquirir competencias en estas áreas y reforzar conceptos aprendidos en clase, pero que solo se consiguen mediante la práctica; además que permitan al estudiante forjar unos cimientos importantes para su vida profesional en aspectos tales como UML, Diagramas de clase, CASE Tools[[1]](#footnote-1) y razonamiento abstracto.

Por otro lado, se es consciente que actualmente ya existen herramientas muy completas que permiten realizar estas labores y que el programa académico ha venido utilizando desde hace algún tiempo (por ejemplo Enterprise Architect[[2]](#footnote-2), Netbeans UML, StarUML, ArgoUML), además que la universidad dispone de licencias de uso para algunas de ellas (por ejemplo, Enterprise Architect que es software privado), y otras son software libre de modo que pueden ser utilizados de cualquier forma; sin embargo, la mayoría de ellas, por no decir todas, son software que se instala de manera nativa, de modo que carecen de portabilidad y facilidad de acceso, aspectos que son muy importantes y atractivos hoy en día en los software de aplicación. Otras requieren ciertas características mínimas de hardware que aunque no es muy común, no siempre están disponibles y son suficientes para soportar que estas aplicaciones se ejecuten de manera adecuada.

Adicional pero no menos importante, se ha notado mediante observaciones indirectas y manifestaciones explicitas de algunos estudiantes, sobre todo en primeros semestres, que muchos de ellos aún tienen dificultades en la configuración de estas aplicaciones, otros expresan no tener computadoras propias o simplemente que los recursos hardware son insuficientes para el funcionamiento adecuado de las aplicaciones, aspectos que evidencian la necesidad planteada anteriormente e impulsan a diseñar modelos de servicios accesibles a través internet, los cuales podrían brindar ciertas flexibilidades y beneficios extras para los usuarios que no tienen las aplicaciones nativas, beneficios tales como facilidad de acceso, portabilidad, autoconfiguración, capacidad de compartir recursos e información.

## JUSTIFICACIÓN

El programa académico de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander, tiene como objeto de estudio la “Construcción y Gestión de Sistemas de Información”[[3]](#footnote-3), por esta razón dentro del pensum existen muchos cursos que se enfocan en programación e ingeniería de software; esto significa que hay una clara tendencia a tratar temas como el modelado de aplicaciones, procesos de desarrollo y otros conceptos que hacen parte de esta área de la ingeniería de sistemas; de esta manera consideramos que se necesita que los estudiantes pongan en práctica técnicas y métodos para desarrollar y mantener software de calidad, teniendo siempre en mente que la tecnología existe para ser usada como herramienta facilitadora del trabajo.

Lo anterior sugiere la necesidad de disponer de una herramienta que permita realizar prácticas de laboratorio, más específicamente prácticas de abstracción de diagramas de clase, de modo que se puedan aplicar todos los conceptos aprendidos, bajo un ambiente real. Es por ello que nos hemos enfocado específicamente en este aspecto, de tal manera que se pueda obtener una herramienta adicional que sirva de apoyo al proceso de aprendizaje y permita adquirir con seguridad conceptos vitales y que serán de gran ayuda en la vida profesional de los estudiantes.

De igual manera sería ideal que los docentes puedan disponer de un mecanismo de control y supervisión de las actividades realizadas por sus estudiantes, permitiendo de manera fácil y cómoda la revisión de los trabajos de clase o evaluaciones parciales; por tal motivo se pretende construir una aplicación orientada bajo el modelo de servicios “Cloud Computing”[[4]](#footnote-4), e incorporar comportamientos como los desarrollados por “Google Drive”[[5]](#footnote-5), en donde los usuarios pueden compartir información con otros de manera fácil y cómoda a través de un clic. Aplicando este concepto a la aplicación en construcción, se espera que los estudiantes puedan compartir con su docente los diagramas realizados de modo que el docente pueda consultarlos y corregirlos a través de la aplicación. Además es significativo incentivar el autoaprendizaje, permitiendo a los estudiantes el intercambio de conocimientos e información, de manera que puedan reforzar los conceptos, obviamente mediante algunos mecanismos de control que garanticen un proceso limpio y transparente.

Adicionalmente una motivación importante para el desarrollo de este proyecto es mejorar la infraestructura de servicios y de tecnologías de información disponibles para los estudiantes de la UFPS, mediante la construcción de herramientas innovadoras y que generen conocimientos sobre el desarrollo de aplicaciones web.

Por último, otro aspecto importante es la necesidad de impulsar e innovar en el desarrollo de tecnologías, el compromiso que tiene el plan de estudios de Ingeniería de Sistemas con la universidad y con la comunidad en general, para desarrollar alternativas de solución a problemáticas mediante el uso de la tecnología y generar conocimiento en base a investigación e innovación nos ha llevado a esta iniciativa, esperando propulsar el desarrollo de nuevos productos y herramientas que presten servicios dentro y fuera de la universidad, y que sean realizadas por los mismos estudiantes.

## OBJETIVOS

### Objetivo General

Implementar una aplicación web basada en el paradigma orientado a servicios de Cloud Computing, que permita el diseño y construcción de diagramas de clase UML y que sirva como medio de apoyo para proyectos de software realizados por los estudiantes de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander.

### Objetivos Específicos

* Realizar un estudio de las principales características que poseen las herramientas CASE utilizadas por los estudiantes del programa de Ingeniería de Sistemas, para determinar los servicios que prestan en cuanto al diseño de diagramas de clase.
* Definir las funcionalidades que el software debe poseer, determinando las características específicas, los servicios que debe prestar a los usuarios y los procedimientos que debe llevar a cabo.
* Realizar el modelado del sistema, determinando una solución óptima a los procedimientos definidos, de modo que no se vea afectado el rendimiento y la estabilidad del software a construir.
* Realizar pruebas de funcionamiento que garanticen el comportamiento adecuado del software y que permitan verificar que se cumplan los requerimientos definidos.
* Documentar las pruebas realizadas junto con los procedimientos llevados a cabo y los resultados obtenidos.
* Desplegar la aplicación en un servidor de modo que pueda ser accedido por estudiantes y docentes.

## ALCANCES Y DELIMITACIONES

### Alcance

La ejecución de este proyecto pretende alcanzar lo siguiente:

* Obtener un producto software que permita a los usuarios diseñar de manera visual diagramas de clase UML.
* Disponer de una plataforma online que pueda ser utilizada desde cualquier lugar y a cualquier hora a través de internet.
* Disponer de una herramienta que permita a los docentes asesorar de manera continua y cómoda a sus estudiantes en los proyectos desarrollo de software.
* Un software para realizar modelado de objetos que sirva como soporte de los proyectos de software y permita generar código fuente de manera automática, agilizando el proceso de desarrollo.
* Un documento mediante el cual se pueda conocer el funcionamiento básico del software y los servicios que este presta, que sirva como guía a los usuarios sobre la manera en la cual deben utilizar la herramienta (Manual de usuario).

### Delimitaciones

Con el propósito de determinar la viabilidad y posibilidad de desarrollo del proyecto se han establecido las delimitaciones mencionadas a continuación:

* El software será totalmente independiente del Sistema de Información Académico de la Universidad (SIA), de modo que no se validara que los usuarios registrados estén en la base de datos del mismo.
* Solo se podrá realizar diagramas de clase UML, la aplicación no abarca el diseño de otro tipo de diagrama.
* El software no dispondrá de un administrador, de modo que no será necesario disponer de usuarios que realicen configuraciones específicas y únicas en el software.
* El código fuente generado a partir de los diagramas de clase será únicamente en lenguaje Java.
* Considerando que existen varios cursos de programación e Ingeniería de Software dentro del pensum académico actual del programa de Ingeniería de Sistemas, el proyecto no estará dirigido específicamente a un curso en particular, sino que estará disponible para cualquier estudiante en cualquier curso; sin embargo, se tomara una población base de uno o dos cursos para realizar el levantamiento de información. En el diseño metodológico se especificará con más detalle este tema.
* El software no permitirá editar el código fuente generado directamente, para esta labor se hace necesario que el usuario disponga de un editor.

# MARCO REFERENCIAL

## MARCO DE ANTECEDENTES

Después de realizar una investigación de las soluciones existentes a la problemática planteada, se encontraron algunas aplicaciones en el mercado que ofrecen parte de las características que esperamos obtener con el software en construcción. A continuación se mencionan algunas de estas aplicaciones:

### Draw.io

Draw.io es una aplicación de diagramado online desarrollada usando la librería JavaScript mxGraph creada por la compañía JGraph Ltd. Esta empresa fue fundada en el año 2000 y está dedicada únicamente a la comercialización y soporte de componentes gráficos de visualización desde el 2004. Actualmente se ha impuesto en el mercado con su producto líder *mxGraph*, el cual comprende una librería JavaScript que permite dibujar componentes para diagramas en cualquier navegador web usando HTML, CSS, JavaScript y SVG, además que permite realizar el Backend[[6]](#footnote-6) de las aplicaciones en diferentes lenguajes como Java, PHP y .NET.

Draw.io es el sitio de demostración para la tecnología desarrollada por la empresa JGraph y su producto mxGraph, el cual permite dibujar diagramas creativos en el navegador, compartir y editar con otros usuarios en tiempo real. Esta aplicación puede ser accedida directamente desde la URL: <https://www.draw.io/>.

### Creately

Creately es una herramienta online UML para crear diagramas de Casos de uso, Diagramas de clase, Actividad, Secuencia, Modelos Entidad Relación, entre muchos otros. Esta es una herramienta comercial la cual es propiedad de la empresa Cinergix Pty Ltd., es el primer producto de la empresa y se encuentra actualmente en una etapa madura; es posible acceder a él en modo prueba aunque no se pueden guardar los proyectos y las funciones son un poco limitadas; para poder tener a disposición toda la herramienta es necesario comprar una licencia de uso. Esta herramienta permite la generación de código fuente, documentación, ingeniería inversa, entre otras funciones; características que lo hacen muy llamativo aunque cuenta con la limitante de ser software propietario.

### GWTUML

GWTUML es una herramienta open source desarrollada por la empresa francesa Object Direct, dirigido por el ODLabs departament. Esta es una compañía dedicada a la consultoría y el desarrollo de tecnologías de información utilizando metodologías de desarrollo ágil, orientación a objetos y Cloud computing, brindando soporte a proyectos open source.

GWTUML es una herramienta CASE en proceso de desarrollo, la cual permite el modelado visual de diagramas de clase, diagramas de objetos y diagramas de secuencia. El desarrollo de esta herramienta se inició en enero de 2009 y la versión actual aún se encuentra en modo experimental, sin embargo, los desarrolladores han liberado algunas APIs[[7]](#footnote-7) y versiones beta de la aplicación, las cuales pueden ser descargadas y utilizadas en otros proyectos que requieran sus servicios. Adicionalmente existe alguna documentación del código fuente (Javadocs) de modo que los usuarios pueden consultarlos en caso de utilizar las APIs disponibles.

Este proyecto está basado en GWT[[8]](#footnote-8) (Google Web Toolkit) el cual es un Framework para el desarrollo de aplicaciones web en lenguaje java, liberando al desarrollador de la complejidad implícita de varios aspectos de la tecnología AJAX, que a su vez es la herramienta más utilizada hoy en día para la construcción de aplicaciones para la internet

### JoinJS

JoinJS es una librería JavaScript para crear diagramas totalmente interactivos en la web. Esta librería puede ser usada tanto para implementar herramientas de diagramado o para simplemente publicar diagramas online. Página principal de la herramienta: [http://www.jointjs.com](http://www.jointjs.com/).

Las principales características de esta librería son:

* Permite conectar objetos vectoriales con varios tipos de líneas y flechas.
* Interacción con objetos y conectores.
* Personalizar eventos y manejadores para mouse, teclado, entre otros.
* Elementos pre construidos para diagramas comúnmente usados (Entidad relación, UML, Org Chart, entre otros).
* Serialización (formato JSON, exportación a SVG en navegadores que lo permiten).
* Soporte para la mayoría de navegadores en versiones antiguas.

## MARCO CONCEPTUAL

### Herramientas CASE

Se puede definir a las Herramientas CASE (Computer Aided Software Engineering) como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un Software [2].

La realización de un nuevo software requiere que las tareas sean organizadas y completadas en forma correcta y eficiente. Las Herramientas CASE fueron desarrolladas para automatizar esos procesos y facilitar las tareas de coordinación de los eventos que necesitan ser mejorados en el ciclo de desarrollo de software.

La mejor razón para la creación de estas herramientas fue el incremento en la velocidad de desarrollo de los sistemas. Por esto, las compañías pudieron desarrollar sistemas sin encarar el problema de tener cambios en las necesidades del negocio, antes de finalizar el proceso de desarrollo.

Las herramientas CASE también permiten a los analistas tener más tiempo para el análisis y diseño y minimizar el tiempo para codificar y probar. Con un CASE integrado, las organizaciones pueden desarrollar rápidamente sistemas de mejor calidad para soportar procesos críticos del negocio y asistir en el desarrollo y promoción intensiva de la información de productos y servicios.

Estas herramientas pueden proveer muchos beneficios en todas las etapas del proceso de desarrollo de software, algunos de estos beneficios son:

* Verificar el uso de todos los elementos en el sistema diseñado.
* Automatizar el dibujo de diagramas.
* Ayudar en la documentación del sistema.
* Ayudar en la creación de relaciones en la Base de Datos.
* Generar estructuras de código.

La introducción de las herramientas CASE para ayudar en este proceso ha permitido que los diagramas puedan ser fácilmente creados y modificados, mejorando la calidad de los diseños de software. Los diccionarios de datos, un documento muy usado que mantiene los detalles de cada tipo de dato y los procesos dentro de un sistema, son el resultado directo de la llegada del diseño de flujo de datos y análisis estructural, hecho posible a través de las mejoras en las Herramientas CASE.

Pronto se reemplazaron los paquetes gráficos por paquetes especializados que habilitan la edición, actualización e impresión en múltiples versiones de diseño. Eventualmente, las herramientas gráficas integradas con diccionarios de base de datos para producir poderosos diseños y desarrollar herramientas, podrían sostener ciclos completos de diseño de documentos. Como un paso final, la verificación de errores y generadores de casos de pruebas fueron incluidos para validar el diseño del software. Todos estos procesos pueden saberse integrados en una simple herramienta CASE que soporta todo el ciclo de desarrollo.

La primera herramienta comercial se remonta a 1982, aunque algunos especialistas indican que algunos ejemplos de herramientas para diagramación ya existían.

No fue sino hasta 1985 en que las herramientas CASE se volvieron realmente importantes en el proceso de desarrollo de software. Los proveedores prometieron a la industria que muchas actividades serían beneficiadas por la ayuda de las CASE. Estos beneficios consistían, por ejemplo, en el aumento en la productividad. El objetivo en 1985 para muchos vendedores era producir software más rápidamente. Las herramientas del CASE serían una familia de métodos favorablemente estructurados para planeamiento, análisis y diseño. Esto llevaría a la generación automática de código para desarrollo de software vía una especificación formalmente diseñada.

Las herramientas CASE, en función de las fases del ciclo de vida abarcadas, se pueden agrupar de la forma siguiente:

1. Herramientas integradas, I-CASE (Integrated CASE, CASE integrado): abarcan todas las fases del ciclo de vida del desarrollo de sistemas. Son llamadas también CASE workbench.
2. Herramientas de alto nivel, U-CASE (Upper CASE - CASE superior) o Front-End, orientadas a la automatización y soporte de las actividades desarrolladas durante las primeras fases del desarrollo: análisis y diseño.
3. Herramientas de bajo nivel, L-CASE (Lower CASE - CASE inferior) o Back-End, dirigidas a las últimas fases del desarrollo: construcción e implantación.

Juegos de herramientas o Tools-Case, son el tipo más simple de herramientas CASE. Automatizan una fase dentro del ciclo de vida. Dentro de este grupo se encontrarían las herramientas de reingeniería, orientadas a la fase de mantenimiento.

### UML

El UML (Lenguaje Unificado de Modelado) es una de las herramientas más emocionantes en el mundo actual de desarrollo de sistemas. Esto se debe a que permite a los creadores de sistemas generar diseños que capturen sus ideas en una forma convencional y fácil de comprender para comunicarlas a otras personas [3].

La comunicación de la idea es de suma importancia. Antes del advenimiento del UML, el desarrollo de sistemas era, con frecuencia, una propuesta al azar. Los analistas de sistemas intentaban evaluar los requerimientos de sus clientes, generar un análisis de requerimientos en algún tipo de notación que ellos mismos comprendieran (aunque el cliente no lo comprendiera), dar tal análisis a uno o más programadores y esperar que el producto final cumpliese con lo que el cliente deseaba.

Dado que el desarrollo de sistemas es una actividad humana, hay muchas posibilidades de cometer errores en cualquier etapa del proceso, por ejemplo, el analista pudo haber malentendido al cliente, es decir, probablemente produjo un documento que el cliente no pudo comprender. Tal vez ese documento tampoco fue comprendido por los programadores quienes, por ende, pudieron generar un programa difícil de utilizar y no generar una solución al problema original del cliente.

Hoy en día, es necesario contar con un plan bien organizado. Un cliente tiene que comprender que es lo que hará un equipo de desarrolladores; además tiene que ser capaz de señalar cambios si no se han captado claramente sus necesidades (o si cambia de opinión durante el proceso). A su vez, el desarrollo es un esfuerzo orientado a equipos, por lo que cada uno de sus miembros tiene que saber qué lugar toma su trabajo en la solución final (así como saber cuál es la solución general).

La clave está en organizar el proceso de diseño de tal forma que los analistas, clientes, desarrolladores y otras personas involucradas en el desarrollo del sistema lo comprendan y convengan con él. UML proporciona tal organización.

El UML es la creación de Grady Booch, James Rumbaugh e Ivar Jacobson. Estos caballeros, apodados recientemente “Los tres amigos”, trabajaban en empresas distintas durante la década de los años ochenta y principios de los noventa y cada uno diseño su propia metodología para el análisis y diseño orientado a objetos. Sus metodologías predominaron sobre la de sus competidores. A mediados de los años noventa empezaron a intercambiar ideas entre si y decidieron desarrollar su trabajo en conjunto.

Los anteproyectos de UML empezaron a circular en la industria del software y las reacciones resultantes trajeron consigo considerables modificaciones. Conforme diversos corporativos vieron que el UML era útil a sus propósitos, se conformó un consorcio de UML. Entre los miembros se encuentran DEC, Hewlett-Packard, Intellicorp, Microsoft, Oracle, Texas Instruments y Rational. En 1997 el consorcio produjo la versión 1.0 del UML.

UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que el UML es un lenguaje, cuenta con reglas para combinar tales elementos entre los cuales encontramos: diagrama de clases, diagrama de objetos, diagrama de casos de uso, diagrama de estados, diagrama de secuencias, diagrama de actividades, diagrama de colaboraciones, diagrama de componentes y diagrama de distribución.

El estándar ofrece una variedad de diagramas, cada uno representa una vista del sistema en construcción, con diferente nivel de abstracción. El diagrama 1 muestra la jerarquía o relación que existen en los diferentes diagramas del estándar UML versión 2.0.



Figura . Jerarquía de los Diagramas de UML 2.0 [4]

### Diagramas de Clase

Los diagramas de clases muestran un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Los diagramas de clases se utilizan para modelar la vista de diseño estática de un sistema. Principalmente, esto incluye modelar el vocabulario del sistema, modelar las colaboraciones o modelar esquemas. Los diagramas de clases también son la base para un par de diagramas relacionados, los diagramas de componentes y los diagramas de despliegue. Los diagramas de clases son importantes no sólo para visualizar, especificar y documentar modelos estructurales, sino que también para construir sistemas ejecutables aplicando ingeniería directa e inversa.

Un diagrama de clases es un diagrama que muestra un conjunto de clases, interfaces, colaboraciones y sus relaciones. Al igual que otros diagramas los diagramas de clases pueden contener notas y restricciones. También pueden contener paquetes o subsistemas, los cuales se usan para agrupar los elementos de un modelo en partes más grandes. A veces se colocarán instancias en los diagramas de clases, especialmente cuando se quiera mostrar el tipo (posiblemente dinámico) de una instancia. El capítulo 5 está dedicado a explicar completamente tanto conceptos básicos como avanzados relacionados a los diagramas de clase.

### Meta-Programación

Meta-programación es el proceso o la práctica por la cual escribimos programas que generan, manipulan o utilizan otros programas. Por ejemplo, un compilador se puede pensar como un programa que genera otro programa, un formateador de código es un programa que manipula otro programa, o una herramienta como Javadoc utiliza nuestro programa para generar su documentación.

En general la Meta-programación se utiliza más fuertemente en el desarrollo de Frameworks, simplemente porque un Framework debe resolver cierta problemática de una aplicación, pero no va a estar diseñado para ninguna aplicación en particular. Es decir, la idea de Framework es que se pueda aplicar y utilizar en diferentes dominios desconocidos para el creador del mismo, entonces estos Frameworks manipulan objetos sin necesidad de conocerlos de antemano. Un ejemplo básico de un Framework es un ORM[[9]](#footnote-9) como Hibernate, el cual se encarga de persistir en base de datos, instancias de nuestras clases sin siquiera conocerlas de antemano.

Así como todo programa construye un modelo para describir su dominio, el dominio de un meta-programa es otro programa denominado programa objeto o base, y este tendrá a su vez, un modelo que describe dicho programa al que llamamos meta-modelo.

**Reflexión Computacional**

Reflexión es un caso particular de meta-programación, en donde “meta-programamos” en el mismo lenguaje en que están escritos (o vamos a escribir) los programas. Es decir todo desde el mismo lenguaje. Inicialmente el lenguaje pionero en cuanto a reflexión fue LISP.

El ejemplo más visible de esto es el caso de SmallTalk, donde no existe una diferenciación entre IDE y nuestro programa. Ambos están hechos en SmallTalk y de hecho viven en un mismo ambiente. Ambos construidos con objetos y pueden interactuar entre sí.

**Tipos de Reflexión:**

Reflexión también abarca los siguientes ítems que vamos a mencionar:

* Introspección: Se refiere a la capacidad de un sistema, de analizarse a sí mismo. Algo así como la introspección humana, pero en términos de software. Para eso, el lenguaje debe proveer ciertas herramientas que le permitan al mismo programa ver o reflejar cada uno de sus componentes.
* Auto modificación: Es la capacidad de un programa de modificarse a sí mismo. Nuevamente esto requiere cierto soporte del lenguaje y las limitaciones van a depender de este soporte.
* Intercesión: Es la capacidad de modificar la semántica del modelo que estamos manipulando desde el mismo lenguaje.

### Cloud Computing

La nube (Cloud Computing) es el procesamiento masivo de datos y almacenamiento de información en servidores, ubicados en cualquier parte del mundo,  conectados a una conexión de internet y a los que se puede acceder desde cualquier dispositivo: computador, Smartphone, tabletas, etc.

En los años 70’s John McCarthy, famoso científico e informático ganador del Premio Turing, expresó: “algún día la computación podrá ser organizada como un servicio público”. El primer uso académico de la expresión “computación en nube”, fue durante una conferencia de RamnathChellappa (actual profesor de la EmoryUniversity) en 1997. Desde ese momento el famoso concepto de la nube ha venido trascendiendo y hoy en día estamos entiendo con más claridad a qué se referían estos visionarios de la tecnología.

EL NIST (National Institute of Standards and Technology) de EE.UU lo define como: “Cloud Computing es un modelo para permitir el acceso adecuado y bajo demanda a un conjunto de recursos de cómputo configurables (ejemplos: redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente provistos y puestos a disposición del cliente con un mínimo esfuerzo de gestión y de interacción con el proveedor del servicio”.

Sin embargo, para algunos el concepto de nube sigue siendo confuso. Lo cierto es que muchas personas  ya realizan actividades diarias en la nube y lo desconocen. Cada vez que alguien consulta  su correo electrónico,  hace una transacción bancaria por Internet,  ve un video de su artista favorito en YouTube, sube fotos de una fiesta en Flickr, escucha música en  Grooveshark o  envía documentos por Dropbox, está  viviendo en la nube.

Para los usuarios finales, el cómputo cloud significa que no existen los costos de adquisición de hardware, ni el manejo de las licencias de software o de actualizaciones, ni nuevos empleados o consultores que contratar, ni instalaciones que rentar, ni costos de capital de ninguna clase — ni costos ocultos. Sólo una tarifa medida por uso o una cuota fija de subscripción. Use sólo lo que quiera, pague sólo lo que usa.

**Modelos de servicios**

* Software como servicio: El software como servicio (en inglés software as a service, SaaS) se encuentra en la capa más alta y caracteriza una aplicación completa ofrecida como un servicio, en demanda, vía multi-tendencia que significa una sola instancia del software que corre en la infraestructura del proveedor y sirve a múltiples organizaciones de clientes. Un ejemplo es la plataforma MS Office como servicio SaaS con su denominación de Microsoft Office 365, que incluye versiones online de la mayoría de las aplicaciones de esta suite ofimática de Microsoft.
* Plataforma como servicio: La capa del medio, que es la plataforma como servicio (en inglés platform as a service, PaaS), es la encapsulación de una abstracción de un ambiente de desarrollo y el empaquetamiento de una serie de módulos o complementos que proporcionan, normalmente, una funcionalidad horizontal (persistencia de datos, autenticación, mensajería, etc.). De esta forma, un arquetipo de plataforma como servicio podría consistir en un entorno conteniendo una pila básica de sistemas, componentes o APIs pres configurados y listos para integrarse sobre una tecnología concreta de desarrollo (por ejemplo, un sistema Linux, un servidor web, y un ambiente de programación como Perl o Ruby). Un ejemplo de este tipo de nube es Google App Engine que sirven aplicaciones de Google y Microsoft.
* Infraestructura como servicio: La infraestructura como servicio (infrastructure as a service, IaaS) -también llamado en algunos casos hardware as a service, HaaS)6 se encuentra en la capa inferior y es un medio de entregar almacenamiento básico y capacidades de cómputo como servicios estandarizados en la red. Servidores, sistemas de almacenamiento, conexiones, enrutadores, y otros sistemas se concentran (por ejemplo a través de la tecnología de virtualización) para manejar tipos específicos de cargas de trabajo, desde procesamiento en lotes (batch) hasta aumento de servidor/almacenamiento durante las cargas pico.

**Tipos de nubes**

* Las nubes públicas se manejan por terceras partes, y los trabajos de muchos clientes diferentes pueden estar mezclados en los servidores, los sistemas de almacenamiento y otras infraestructuras de la nube. Los usuarios finales no conocen qué trabajos de otros clientes pueden estar corriendo en el mismo servidor, red, discos como los suyos propios.
* Las nubes privadas son una buena opción para las compañías que necesitan alta protección de datos y ediciones a nivel de servicio. Las nubes privadas están en una infraestructura en-demanda manejada por un solo cliente que controla qué aplicaciones debe correr y dónde. Son propietarios del servidor, red, y disco y pueden decidir qué usuarios están autorizados a utilizar la infraestructura.

Las nubes híbridas combinan los modelos de nubes públicas y privadas. Usted es propietario de unas partes y comparte otras, aunque de una manera controlada. Las nubes híbridas ofrecen la promesa del escalado aprovisionada externamente, en-demanda, pero añaden la complejidad de determinar cómo distribuir las aplicaciones a través de estos ambientes diferentes. Las empresas pueden sentir cierta atracción por la promesa de una nube híbrida, pero esta opción, al menos inicialmente, estará probablemente reservada a aplicaciones simples sin condicionantes, que no requieran de ninguna sincronización o necesiten bases de datos complejas.

## MARCO TEORICO

### Java

El significado de Java tal y como se le conoce en la actualidad es el de un lenguaje de programación y un entorno para ejecución de programas escritos en el lenguaje Java. AI contrario que los compiladores tradicionales, que convierten el código fuente en instrucciones a nivel de máquina, el compilador Java traduce el código fuente Java en instrucciones que son interpretadas por la Máquina Virtual Java (JVM, *Java Virtual Machine).* A diferencia de los lenguajes C y C++ en los que está inspirado, Java es un lenguaje interpretado. Aunque hoy en día Java es por excelencia el lenguaje de programación para Internet y la World Wide Web en particular, Java no comenzó como proyecto Internet y por esta circunstancia es idóneo para tareas de programación de propósito general y, de hecho, muchas de las herramientas Java están escritas en Java[[10]](#footnote-10).

La independencia de plataforma es una de las razones por las que Java es interesante para Internet, ya que muchas personas deben tener acceso con ordenadores distintos. Pero no se queda ahí, Java está desarrollándose incluso para distintos tipos de dispositivos además del ordenador como móviles, agendas y en general para cualquier cosa que se le ocurra a la industria.

**Java Virtual Machine (JVM)**

La JVM es una de las piezas fundamentales de la plataforma Java. Básicamente se sitúa en un nivel superior al Hardware del sistema sobre el que se pretende ejecutar la aplicación, y este actúa como un puente que entiende tanto el bytecode, como el sistema sobre el que se pretende ejecutar. Así, cuando se escribe una aplicación Java, se hace pensando que será ejecutada en una máquina virtual Java en concreto, siendo ésta la que en última instancia convierte de código bytecode a código nativo del dispositivo final.

La gran ventaja de la máquina virtual java es aportar portabilidad al lenguaje de manera que desde Sun Microsystems se han creado diferentes máquinas virtuales java para diferentes arquitecturas y así un programa .class escrito en un Windows puede ser interpretado en un entorno Linux. Tan solo es necesario disponer de dicha máquina virtual para dichos entornos.

**Componentes Java**

*J2EE* es un grupo de especificaciones diseñadas por Sun que permiten la creación de aplicaciones empresariales, esto sería: acceso a base de datos (JDBC), utilización de directorios distribuidos (JNDI), acceso a métodos remotos (RMI/CORBA), funciones de correo electrónico (Java Mail). Java EE también configura algunas especificaciones únicas para componentes Java EE, estas incluyen: Enterprise JavaBeans EJB, Servlets, Portlets (siguiendo la especificación de Portlets Java), Java Server Pages y varias tecnologías de servicios web. Ello permite al desarrollador crear aplicaciones empresariales portables entre plataformas y escalables, a la vez que integrable con tecnologías anteriores. Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de en tareas de mantenimiento de bajo nivel[[11]](#footnote-11).

*J2SE* es una colección de APIs del lenguaje de programación Java útiles para muchos programas de la Plataforma Java. La Plataforma Java 2 Enterprise Edition incluye todas las clases en el Java SE, además de algunas de las cuales son útiles para programas que se ejecutan en servidores sobre estaciones de trabajo.

*J2ME* es la versión de Java orientada a los dispositivos móviles. Debido a que los dispositivos móviles tienen una potencia de cálculo baja e interfaces de usuario pobres, es necesaria una versión específica de Java destinada a estos dispositivos, ya que el resto de versiones de Java, J2SE o J2EE, no encajan dentro de este esquema. J2ME es por tanto, una versión “reducida” de J2SE.

### Frameworks

En el desarrollo de software, un Framework es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

Muchas veces a los Frameworks se les tiende a dar el mismo nombre de Patrón de diseño, pensando que hacen referencia al mismo concepto, sin embargo esto no es cierto y de hecho tienen significados diferentes; un Framework representa código implementado, mientras que un patrón de diseño representa conocimiento y experiencia.

Con el crecimiento exponencial de Internet en los últimos años, las aplicaciones Web se han convertido en una parte básica y común dentro del desarrollo de software; estas han acaparado la atención no solo de las empresas que desean formar parte de este nuevo mundo, sino también de aquellas que se han dedicado a las herramientas de desarrollo de software.

Dentro de este tipo de aplicaciones, Java juega un papel muy importante actualmente ya que este es uno de los usos más comunes que se le da a este lenguaje de programación, además de que representa uno de los mejores medios para construir dichas aplicaciones.

Aunque no existe una clasificación estándar y formal que se aplique a los Frameworks para aplicaciones web, existen algunos aspectos que se pueden considerar para diferenciar unos de otros.

De esta manera y teniendo en cuenta el enfoque de los Frameworks, se pueden mencionar dos tipos para el lenguaje Java:

* **Frameworks de aplicación**: Los Frameworks de aplicación son aquellos que están basados en los Request HTTP, es decir hacen uso del API Servlet. Así mismo, estos Frameworks no se preocupan acerca de los detalles de cómo se renderiza la interfaz de usuario y no hacen distinción entre acciones que solo afectan a la interfaz de usuario y aquellas que requieren ser procesadas por código de la aplicación.
* **Frameworks de Interfaz de usuario**: Estos Frameworks están basados en componentes, se enfocan en los detalles de la interfaz de usuario y no se interesan en cómo es implementado el resto de la aplicación. De igual forma, definen un API detallado para los componentes que formaran la interfaz de usuario con los objetivos de ligar a estos con la lógica de la aplicación adecuada, determinar qué acciones de los usuarios resultaran eventos de la interfaz y como serán manejados estos últimos, entre otras finalidades.

De entre todos los Frameworks para el desarrollo en Java, sobresalen los siguientes:

* **Struts**: Framework de aplicación open source, este fue el primer Framework en salir para el desarrollo web en Java.
* **Java Server Faces (JSF)**: Framework de interfaz de usuario, es el estándar de Java actualmente para el desarrollo web.
* **Spring**: Framework de aplicación muy popular últimamente.

### Servidor de Aplicaciones

En informática, se denomina servidor de aplicaciones a un servidor en una red de computadores que ejecuta ciertas aplicaciones. Usualmente se trata de un dispositivo de software que proporciona servicios de aplicación a las computadoras cliente. Un servidor de aplicaciones generalmente gestiona la mayor parte (o la totalidad) de las funciones de lógica de negocio y de acceso a los datos de la aplicación. Los principales beneficios de la aplicación de la tecnología de servidores de aplicación son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones.

También conocidos como Middleware, el cual es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o sistemas operativos. Éste simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones que son necesarias en los sistemas distribuidos. De esta forma se provee una solución que mejora la calidad de servicio, seguridad, envío de mensajes, directorio de servicio, etc.

Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red). El middleware abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas. Dependiendo del problema a resolver y de las funciones necesarias, serán útiles diferentes tipos de servicios de middleware. Por lo general el middleware del lado cliente está implementado por el Sistema Operativo, el cual posee las bibliotecas que ejecutan todas las funcionalidades para la comunicación a través de la red.

En Java estos son llamados contenedores JEE, los cuales proveen un entorno de ejecución de componentes Java. También brindan servicios de seguridad, transacciones, administración de ciclo de vida, caching, persistencia y comunicación en la red. Existen dos tipos de contenedores JEE:

* **Contenedores Web**: Almacenan componentes de presentación (JSPs, Servlets). Un ejemplo claro de este tipo de contenedor es Apache Tomcat, el cual permite el procesamiento de este tipo de componentes.
* **Contenedores EJB**: Adicional a los componentes permitidos por los contenedores web, los contenedores EJB permiten la ejecución de componentes EJB (Enterprise Java Beans). Dentro de este tipo de software se encuentran herramientas como Glassfish, JBoss AS y WebLogic.

### Extreme Programming (XP)

Metodología ágil basada en cuatro principios: simplicidad, comunicación, retroalimentación y valor. Además, orientada por pruebas y refactorización, se diseña e implementan las pruebas antes de programar la funcionalidad, el programador crea sus propias Pruebas de Unidad[[12]](#footnote-12).

Este método es típicamente atribuido a Kent Beck, Ron Jeffries y Ward Cinningham. El objetivo de XP son grupos pequeños y medianos de construcción de software en donde los requisitos aún son muy ambiguos, cambian rápidamente o son de alto riesgo. XP busca la satisfacción del cliente tratando de mantener durante todo el tiempo su confianza en el producto. Además, sugiere que el lugar de trabajo sea una sala amplia, si es posible sin divisiones (en el centro los programadores, en la periferia los equipos individuales). Una ventaja del espacio abierto es el incremento en la comunicación y el proporcionar una agenda dinámica en el entorno de cada proyecto [5].

**Actividades de XP**

* **Codificar**: Es necesario codificar y plasmar nuestras ideas a través del código. En programación, el código expresa la interpretación del problema, así podemos utilizar el código para comunicar, para hacer comunes las ideas, y por tanto para aprender y mejorar.
* **Hacer pruebas**: Las características del software que no pueden ser demostradas mediante pruebas simplemente no existen. Las pruebas dan la oportunidad de saber si lo implementado es lo que en realidad se tenía en mente. Las pruebas nos indican que nuestro trabajo funciona, cuando no podemos pensar en ninguna prueba que pudiese originar un fallo en nuestro sistema, entonces habremos acabado por completo.
* **Escuchar**: Si vamos a hacer pruebas tenemos que preguntar si lo obtenido es lo deseado, y tenemos que preguntar a quién necesita la información. Tenemos que escuchar a nuestros clientes cuáles son los problemas de su negocio, debemos de tener una escucha activa explicando lo que es fácil y difícil de obtener, y la realimentación entre ambos nos ayudan a todos a entender los problemas.
* **Diseñar**: El diseño crea una estructura que organiza la lógica del sistema, un buen diseño permite que el sistema crezca con cambios en un solo lugar. Los diseños deben de ser sencillos, si alguna parte del sistema es de desarrollo complejo, lo apropiado es dividirla en varias. Si hay fallos en el diseño o malos diseños, estos deben de ser corregidos cuanto antes.

**Ciclo de Vida de XP**

El ciclo de vida de XP se enfatiza en el carácter interactivo e incremental del desarrollo, en donde una iteración de desarrollo es un período de tiempo en el que se realiza un conjunto de funcionalidades determinadas que en el caso de XP corresponden a un conjunto de historias de usuarios.

Las iteraciones son relativamente cortas ya que se piensa que entre más rápido se le entreguen desarrollos al cliente, más retroalimentación se va a obtener y esto va a representar una mejor calidad del producto a largo plazo. Existe una fase de análisis inicial orientada a programar las iteraciones de desarrollo y cada iteración incluye diseño, codificación y pruebas, fases superpuestas de tal manera que no se separen en el tiempo.

**Fase de Exploración**: En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

**Fase de Planeación**: Se priorizan las historias de usuario y se acuerda el alcance del Release[[13]](#footnote-13). Los programadores estiman cuánto esfuerzo requiere cada historia y a partir de allí se define el cronograma. La duración del cronograma del primer Release no excede normalmente dos meses. La fase de planeamiento toma un par de días. Se deben incluir varias iteraciones para lograr un Release. El cronograma fijado en la etapa de planeamiento se realiza a un número de iteraciones, cada una toma de una a cuatro semanas en ejecución. La primera iteración crea un sistema con la arquitectura del sistema completo. Esto es alcanzado seleccionando las historias que harán cumplir la construcción de la estructura para el sistema completo. El cliente decide las historias que se seleccionarán para cada iteración. Las pruebas funcionales creadas por el cliente se ejecutan al final de cada iteración. Al final de la última iteración el sistema está listo para producción.

**Fase de Producción**: Requiere prueba y comprobación extra del funcionamiento del sistema antes de que éste se pueda liberar al cliente. En esta fase, los nuevos cambios pueden todavía ser encontrados y debe tomarse la decisión de si se incluyen o no en el Release actual. Durante esta fase, las iteraciones pueden ser aceleradas de una a tres semanas. Las ideas y las sugerencias pospuestas se documentan para una puesta en práctica posterior por ejemplo en la fase de mantenimiento. Después de que se realice el primer Release productivo para uso del cliente, el proyecto de XP debe mantener el funcionamiento del sistema mientras que realiza nuevas iteraciones.

**Fase de Mantenimiento**: Requiere de un mayor esfuerzo para satisfacer también las tareas del cliente. Así, la velocidad del desarrollo puede desacelerar después de que el sistema esté en la producción. La fase de mantenimiento puede requerir la incorporación de nueva gente y cambiar la estructura del equipo.

**Fase de Muerte**: Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.

### SVG

Los Gráficos Vectoriales Redimensionables (del inglés Scalable Vector Graphics) o SVG son una especificación para describir gráficos vectoriales bidimensionales, tanto estáticos como animados (estos últimos con ayuda de SMIL), en formato XML.

SVG se convirtió en una recomendación del W3C en septiembre de 2001, por lo que ya ha sido incluido de forma nativa en el navegador web del W3C Amaya. Las versiones 1.5 y posteriores de Mozilla Firefox soportan gráficos hechos con SVG, así como el navegador Opera que desde su versión 8 ha implementado SVG 1.1 Tiny en su núcleo. Navegadores como Google Chrome, Safari e Internet Explorer 9 también son capaces de mostrar imágenes en formato SVG sin necesidad de complementos externos. Otros navegadores web, como versiones anteriores a la 9 de Internet Explorer, necesitan un conector o Plug-in, para lo que se puede utilizar.

El SVG permite tres tipos de objetos gráficos:

* Elementos geométricos vectoriales (por ejemplo caminos consistentes en rectas y curvas, y áreas limitadas por ellos)
* Imágenes de mapa de bits /digitales
* Texto

Los objetos gráficos pueden ser agrupados, transformados y compuestos en objetos previamente renderizados, y pueden recibir un estilo común. El texto puede estar en cualquier espacio de nombres XML admitido por la aplicación, lo que mejora la posibilidad de búsqueda y la accesibilidad de los gráficos SVG. El juego de características incluye las transformaciones anidadas, los clipping paths, las máscaras alfa, los filtros de efectos, las plantillas de objetos y la extensibilidad.

El dibujado de los SVG puede ser dinámico e interactivo. El Document Object Model (DOM) para SVG, que incluye el DOM XML completo, permite animaciones de gráficos vectoriales sencillos y eficientes mediante ECMAScript o SMIL. Un juego amplio de manejadores de eventos, como "onMouseOver" y "onClick", pueden ser asignados a cualquier objeto SVG. Debido a su compatibilidad y relación con otras normas Web, características como el scripting pueden ser aplicadas a elementos SVG y a otros elementos XML desde distintos espacios de nombre XML simultáneamente dentro de la misma página web. Un ejemplo extremo de esto es un juego completo de Tetris realizado como un objeto SVG.

Si el espacio de almacenamiento es un problema, las imágenes SVG pueden salvarse comprimidas con gzip, en cuyo caso pasan a ser imágenes SVGZ. Debido a la verbosidad del XML, este tiende a comprimirse muy bien, y estos ficheros pueden ser mucho más pequeños. Aun así, a menudo el fichero vectorizado original (SVG) es más pequeño que la versión de mapa de bits. A pesar de ser un lenguaje vectorial, SVG permite crear imágenes complejas.

### HTML, JavaScript y CSS

Originalmente, las páginas HTML sólo incluían información sobre sus contenidos de texto e imágenes. Con el desarrollo del estándar HTML, las páginas empezaron a incluir también información sobre el aspecto de sus contenidos: tipos de letra, colores y márgenes. La posterior aparición de tecnologías como JavaScript, provocaron que las páginas HTML también incluyeran el código de las aplicaciones (llamadas scripts) que se utilizan para crear páginas web dinámicas.

Incluir en una misma página HTML los contenidos, el diseño y la programación complica en exceso su mantenimiento. Normalmente, los contenidos y el diseño de la página web son responsabilidad de diferentes personas, por lo que es conveniente separarlos. CSS es el mecanismo que permite separar los contenidos definidos mediante XHTML y el aspecto que deben presentar esos contenidos.

Una ventaja añadida de la separación de los contenidos y su presentación es que los documentos XHTML creados son más flexibles, ya que se adaptan mejor a las diferentes plataformas: pantallas de ordenador, pantallas de dispositivos móviles, impresoras y dispositivos utilizados por personas discapacitadas.

De esta forma, utilizando exclusivamente XHTML se crean páginas web "feas" pero correctas. Aplicando CSS, se pueden crear páginas "bonitas" a partir de las páginas XHTML correctas.

El código JavaScript se encierra entre etiquetas <script> y se incluye en cualquier parte del documento. Aunque es correcto incluir cualquier bloque de código en cualquier zona de la página, se recomienda definir el código JavaScript dentro de la cabecera del documento (dentro de la etiqueta <head>). Las instrucciones JavaScript también se pueden incluir en un archivo externo de tipo JavaScript que los documentos XHTML enlazan mediante la etiqueta <script>. Se pueden crear todos los archivos JavaScript que sean necesarios y cada documento XHTML puede enlazar tantos archivos JavaScript como necesite.

## MARCO DE LEGAL

### Acuerdos de uso

El software, resultado de la ejecución del proyecto, se regirá bajo los marcos legales fijados por la licencia pública Creative Common (LPCC) que dicta las siguientes condiciones de uso, encontradas más detalladamente en el documento Atribución No Comercial Compartir Igual 2.5 (Colombia) [6]:

* Se posee la libertad de compartir la obra, entendiendo compartir como la capacidad de copiar, distribuir, ejecutar y comunicar públicamente la obra.
* Además puede llevar a cabo obras derivadas de la original.
* Es obligatorio reconocer los créditos de la obra de manera especificada por el autor o el licenciante, pero no de manera que sugiera que tiene el apoyo de este último, o que apoya el uso que hacen a su obra.
* No puede ser utilizada para uso comercial.
* Si se altera o transforma, o genera un derivado a partir de esta obra, solo puede ser distribuida bajo una licencia idéntica a la presente.

### Contrato de licencia de código binario, SUN MICROSYSTEMS

Licencia de uso de software concedidos por Sun Microsystems, para la utilización de Java SE, siempre y cuando sean aceptados los términos de licencia, expuestos en el documento SUN MICROSYSTEMS, INC. CONTRATO DE LICENCIA DE CÓDIGO BINARIO [7].

“SUN MICROSYSTEMS, INC. (EN ADELANTE DENOMINADO “SUN”) LE CONCEDE LA LICENCIA DEL SOFTWARE DEFINIDO A CONTINUACIÓN ÚNICAMENTE CON LA CONDICIÓN DE QUE USTED ACEPTE TODOS LOS TÉRMINOS ESTIPULADOS EN EL PRESENTE CONTRATO DE LICENCIA DE CÓDIGO BINARIO Y TÉRMINOS DE LICENCIA ADICIONALES (EN CONJUNTO DENOMINADOS “CONTRATO”). POR FAVOR, LEA EL CONTRATO DETENIDAMENTE. EL USO DEL SOFTWARE SIGNIFICA QUE HA LEÍDO LAS CONDICIONES Y QUE LAS ACEPTA. SI ACEPTA ESTAS CONDICIONES EN REPRESENTACIÓN DE UNA COMPAÑÍA U OTRA ENTIDAD LEGAL, SIGNIFICA QUE ESTÁ EN POSESIÓN DE LA AUTORIDAD QUE LE PERMITE VINCULAR LA ENTIDAD LEGAL A ESTAS CONDICIONES. SI NO ESTÁ EN POSESIÓN DE DICHA AUTORIDAD O SI NO DESEA QUEDAR VINCULADO A ESTAS CONDICIONES, NO DEBE UTILIZAR EL SOFTWARE EN ESTA INSTALACIÓN NI EN NINGÚN OTRO SOPORTE EN EL QUE SE UBIQUE EL SOFTWARE”.

### GNU General Public Licence

La Licencia Pública General de GNU (GNU GPL, por sus siglas en inglés) es una licencia libre y gratuita con derecho de copia para software y otros tipos de obras.

Las licencias para la mayoría del software y otras obras de índole práctica están diseñadas para privarle de la libertad para distribuir y modificar las obras. Por el contrario, la Licencia Pública General de GNU garantiza la libre distribución y modificación de todas las versiones de un programa, a fin de asegurarle dicha libertad a todos los usuarios. En la Fundación para el Software Libre utilizamos la Licencia Pública General de GNU para la mayoría de nuestro software; también se aplica a cualquier otra obra publicada de esta manera por sus autores. Usted también puede aplicarla a sus programas.

Cuando hablamos de software libre, nos referimos a la libertad, no al precio. Nuestras Licencias Públicas Generales están diseñadas para garantizarle a usted la libertad de distribuir copias de software libre (y cobrar por ellas, si así lo desea), obtener el código fuente, o tener la posibilidad de obtenerlo, modificar el software o utilizar partes del mismo en nuevos programas libres, y saber que puede hacer estas cosas [8].

### Propiedad Intelectual

De acuerdo con lo estimulado en el artículo 156 del acuerdo 065 del 26 de 1996, correspondiente al estatuto estudiante de la Universidad Francisco de Paula Santander, el cual dicta de la siguiente manera: “Los trabajos de grado son propiedad intelectual de la Universidad y su uso estará sujeto a las normas que para tal fin estén vigentes”[[14]](#footnote-14).

# DISEÑO METODOLÓGICO

El diseño metodológico del proyecto está enmarcado en la investigación aplicada y se fundamentará en la ingeniería de software, de ella se tomarán los conceptos, metodologías y técnicas propias para la construcción de la aplicación.

## TIPO DE INVESTIGACIÓN

Según la problemática planteada, el proyecto pretende dar solución a una necesidad vista dentro del programa de Ingeniería de Sistemas de la UFPS, de tal manera que se englobará el proyecto como una INVESTIGACIÓN APLICADA en términos del desarrollo de software que permitan solventar necesidades existentes dentro de la organización.

## FUENTES DE INFORMACIÓN

### Fuentes de información primaria

Teniendo en cuenta que la problemática afecta a la comunidad perteneciente al programa de Ingeniería de Sistemas, se consideran las siguientes fuentes de información primaria:

* Estudiantes que están cursando las asignaturas, “Programación Orientada a Objetos” y “Análisis y diseño de Sistemas”, las cuales se enfocan principalmente en el modelado de aplicaciones. Estos son cursos de 2º y 6º semestre respectivamente según el pensum 115 [9]. No obstante, a manera de observación indirecta se tomaran otros cursos que puedan usar el producto en construcción.
* Docentes del programa Ingeniería de Sistemas, los cuales son fuente de información de vital importancia, ya que ellos son los que acompañan el proceso de aprendizaje de los estudiantes.

### Fuentes de información secundaria

Estas fuentes hacen referencia a estudios, técnicas, teorías, documentos y toda clase de información existente sobre el tema en particular de modo que se establecen las siguientes:

* Herramientas CASE existentes, comerciales o de código abierto, que permitan el diseño de diagramas de clase. Se tomará en cuenta las herramientas utilizadas actualmente por los estudiantes, de modo que se pueda determinar que funcionalidad o que servicios ofrecen.
* Documento de Especificación del estándar UML para Diagramas de Clase.
* Documentación de proyectos de software online que permitan diseñar diagramas de clase.

Libros enfocados en temas como: Cloud computing, Ingeniería de Software, Lenguaje Unificado de Modelado, Procesos de desarrollo de software, desarrollo de aplicaciones Web, desarrollo de aplicaciones con J2EE, AJAX.

## RECOLECCIÓN Y ANALISIS DE INFORMACIÓN

El desarrollo del proyecto requiere obtener información general sobre la construcción de diagramas de clase, esto implica conocer a fondo conceptos tales como: Clases, Interfaces, Relaciones, Navegabilidad, entre otros; además de todos los elementos de los cuales dispone UML para el diseño de este tipo de diagramas. Para esto se realizará una investigación profunda sobre el tema, consultado bibliografía existente y diferentes autores.

Adicionalmente se necesita determinar las funcionalidades básicas que poseen las herramientas CASE utilizadas actualmente por los estudiantes de la UFPS, de manera que se pueda conocer qué procedimientos, servicios y características poseen con respecto al diseño de diagramas de clase; esto se hace con el objetivo de determinar los requerimientos de funcionalidad mínimos que debe poseer el software en construir y los servicios que este debe ofrecer. De cierta manera, mediante observación indirecta y a manera de tradición se conocen las principales herramientas CASE utilizadas por estudiantes y docentes de Ingeniería de Sistemas de la UFPS (Enterprise Architect, ArgoUML, StarUML), y considerando que el programa académico dispone de algunas licencias de uso para este tipo de herramientas (como es el caso de Enterprise Architect), se procederá a realizar un análisis concreto de estas herramientas específicamente. De igual manera y como se mencionó anteriormente, este análisis servirá como punto de partida del proyecto y además ayudará a tener un idea clara de la aplicación que se desea llegar a construir.

El procedimiento utilizado para la recolección de información en este caso será de manera literaria, mediante consultas e investigaciones de material bibliográfico, logrando una apropiación de conceptos y técnicas importantes para la construcción del proyecto.

Por último es necesario aclarar que este proyecto no pretende dar respuestas a incógnitas o hipótesis que se tengan sobre algún fenómeno académico o social dentro de la Universidad, sino que se encargara de suplir una necesidad detectada dentro del establecimiento.

Para el análisis de la información se procederá a clasificar los datos según su naturaleza, determinando los aspectos importantes para cada fase del desarrollo y la información necesaria para construir los componentes en cada una de ellas.

El análisis de la información consistirá en evaluar los datos obtenidos de las fuentes mencionadas, documentación de las herramientas CASE existentes, manuales de usuario, análisis del funcionamiento de las aplicaciones utilizadas actualmente, literatura especifica del caso, y en general, fuentes que sean cercanas al tema. Se realizará un proceso de clasificación y agrupamiento de los datos, de manera que se pueda mediante ellos, apoyar la siguiente fase del desarrollo y que sean las directrices de todo el proyecto en general, de modo que mediante estos se puedan tomar decisiones importantes para el desarrollo.

# ANALISIS DE LAS HERRAMIENTAS CASE

Con el objetivo de obtener información importante para la realización del proyecto y con el ánimo de iniciar el proceso de levantamiento de requerimientos se ha realizado un pequeño análisis de algunas de las herramientas utilizadas dentro del plan de estudios de Ingeniería de Sistemas para realizar modelado UML, de modo que se pueda establecer cuáles son las principales características y servicios que estas ofrecen a los usuarios en cuanto al diseño de *Diagramas de Clase* y que puedan ser incluidos en las funcionalidades del software en construcción.

De modo aleatorio se han seleccionado dos de las herramientas más conocidas y que a manera de observación se ha determinado que son ampliamente utilizadas por estudiantes y docentes del plan de estudios, adicionalmente se ha seleccionado una herramienta similar a la que está en construcción, de modo que podamos conocer qué características ofrece a los usuarios y cuales son imprescindibles en una herramienta de modelamiento UML. Las herramientas seleccionadas son: *Enterprise Architect* que es un software comercial conocido a nivel mundial por su gran calidad, *StarUML* que a su vez es open source pero ampliamente utilizado, y por ultimo ArgoUML que es una herramienta open source de modelado construida en Java puro.

Al momento de realizar este análisis se tomó la última versión de las aplicaciones que se instalan de manera nativa, Enterprise Architect en su versión 9.0.9 con un periodo de prueba de 30 días y StarUML en su versión 5.0. En el caso de ArgoUML la versión usada fue 0.3.4.

A continuación se describirá cada uno de ellos, resaltando en detalle cada una de sus características:

## ENTERPRISE ARCHITECT

Enterprise Architect de Sparx Systems es una herramienta CASE (Computer Aided Software Engineering) para el diseño y construcción de sistemas de software, para el modelado de procesos de negocios, y para objetivos de modelado más generalizados.

EA es una herramienta de análisis y diseño UML comprensivo, que cubre el desarrollo de software desde la obtención de los requisitos, diseño del modelo, pruebas, cambio de control y mantenimiento para la implementación, con completa trazabilidad. EA combina el poder de la última especificación UML 2.4.1 con alta performance, una interfaz intuitiva, para brindar un modelado avanzado a todo el equipo de desarrollo. Es una herramienta visual con multi-usuario con un gran establecimiento de características ayudando a los analistas, testers, administradores de proyectos, personal del control de calidad y desarrolladores alrededor del mundo a construir y documentar, software sostenible.

La información consignada a continuación fue tomada de la documentación explicita de la herramienta [10].

### Historia

Sparx Systems es una empresa australiana dedicada a la creación de herramientas para el modelado UML, fundada en 1996 por Geoffrey Sparx y situada en Creswick una pequeña población de Victoria, Australia. La primera versión de Enterprise Architect fue liberada en el año 2000 y fue originalmente diseñada como una herramienta de modelado UML para la versión 1.1 del estándar, de allí en adelante el producto ha evolucionado para soportar otras especificaciones OMG[[15]](#footnote-15) UML como 1.3, 2.0, 2.1, 2.3 y 2.4.1.

Durante el tiempo de vida de la aplicación se han liberado alrededor de 20 versiones, desde la versión 1.0 hasta la más reciente 9.3 liberada en Agosto de 2012, cada una con mejoras y características importantes según los cambios que ha sufrido el estándar UML y la adopción de algunas tecnologías.

Actualmente la cantidad de estándares para el diseño y modelado de software y sistemas de negocios que son soportados por Enterprise Architect es considerable, teniendo en cuenta la adopción de UML 2.4.1 como principal herramienta para modelar aplicaciones e incluyendo algunos otros BPMN, WSDL, BPEL y XSD, ha permitido que la herramienta se imponga como uno de los líderes del mercado y que su aceptación por parte de la industria del software haya sido grandiosa.

Enterprise Architect se encuentra disponible en diferentes ediciones, según las necesidades del usuario o de las organizaciones que usan la aplicación, de modo que se pueden adquirir diferentes tipos de licencias para diferentes usos. Actualmente existen dos ediciones y dentro de las cuales se pueden comprar varias versiones del producto:

* Ediciones base
* Corporativa
* Profesional
* Escritorio
* Ediciones Suite
* Ultimate
* Ingeniería de Sistemas
* Ingeniería de Software y negocios

Los precios del producto dependen de la versión adquirida y de la cantidad de licencias que se obtengan de cada uno, siendo la versión de Escritorio la más simple hasta la versión Ultimate que a su vez posee la capacidad de trabajo en múltiples dominios de manera remota y un soporte completo de ingeniería de software y de negocios.

### Características Generales

Enterprise Architect posee un gran abanico de características, todas encaminadas al desarrollo y mantenimiento de proyectos de software, aunque también brinda soporte para otro tipo de proyectos.

La información aquí consignada fue tomada de la documentación explicita de Enterprise Architect, disponible en [10], además se usó parte de la información del “Contenido de Ayuda” que viene incluido en la versión de prueba del software.

| **Característica** | **Descripción** |
| --- | --- |
| Plataforma | Diseñado para ser usado bajo sistemas operativos Windows, sin embargo puede ser instalado y utilizado mediante emuladores como CrossOver en sistemas Linux o MAC. CrossOver es una herramienta propietaria basada en una tecnología open source llamada Wine, el cual es una API Win32 re-desarrollada para sistemas operativos basados en Unix como Mac OS X o Linux. |
| Categoría de Software | Software propietario distribuido bajo licencias comerciales, el costo de estas depende de la versión del software y la cantidad de licencias adquiridas. |
| Versión de UML | UML 2.4.1 Liberada en agosto de 2011. |
| Diagramas UML | Soporte para todos los tipos de diagramas UML tanto estructurales como comportamentales. Algunos diagramas adicionales no UML: Análisis, Requerimientos, Bases de Datos, Mantenimiento, Interfaces de Usuario, Interacción y modelos del negocio. |
| Tecnologías MDG | Soporta tecnologías como ICONIX, ArcGIS, BPMN, SMOF, SoaML, SysML. |
| Fases del desarrollo | Modelamiento del Negocio, Ingeniería de Requerimientos, Análisis, Diseño, Implementación, Pruebas, Mantenimiento, Gestión de la Calidad. |
| Estándar XMI | Soporte completo del estándar en su versión 2.1, permite exportar e importar proyectos en archivos .XMI de modo que puedan ser llevados a otras herramientas de modelado. |
| Proyectos y equipos de trabajo | Permite generar repositorios de servidor DBMS , de modo que se pueda trabajar de manera concurrente en proyectos de gran envergadura y con gran cantidad de usuarios. Adicionalmente permite integración con repositorios de control de versiones como Subversion, CVS y SCC. |
| Importación / Exportación | Exportación: Existen varias formas de exportar proyectos, diagramas o cualquier otro tipo de artefacto creado dentro del software, archivos XML, XMI, Código fuente, imágenes planas de los diagramas en formato PNG, JPG, GIF, entre otros. Importación: Desde XML, XMI, archivos CSV, desde código fuente en varios lenguajes, desde archivos binarios como JARs o ensamblados .NET. |
| Ingeniería Directa e Inversa | Enterprise Architect soporta la generación directa e ingeniería reversa de código fuente para muchos lenguajes populares incluyendo C++, C#, Java, Delphi, VB.Net, Visual Basic, ActionScript, Python y PHP. Con un editor de código fuente “resaltador de sintaxis” integrado. Enterprise Architect le permite rápidamente navegar y explorar su código fuente del modelo en el mismo ambiente. Las plantillas de generación de código le permiten personalizar el código fuente generado de acuerdo a las especificaciones de su compañía. |
| Documentación generada | Soporte completo de WYSIWYG manejador de plantillas para generar archivos RTF. |
| Soporte de Versiones | Soporte para servidores Subversion SVN. |

Tabla . Características principales de Enterprise Architect

### Funciones para Diagramas de Clase

El diagrama de Clases captura la estructura lógica del sistema - las clases y cosas que constituyen el modelo -. Es un modelo estático, describiendo lo que existe y qué atributos y comportamiento tiene, más que cómo se hace algo. Los diagramas de Clases son los más útiles para ilustrar las relaciones entre las clases e interfaces. Las generalizaciones, las agregaciones y las asociaciones son todas valiosas para reflejar la herencia, la composición o el uso y las conexiones respectivamente.

El diagrama de abajo ilustra las relaciones de agregación entre clases. La agregación con la punta de flecha en color claro indica que la clase Cuenta es usada por LibroDeDirecciones, pero que no está contenida necesariamente. La agregación con la punta de flecha en color oscuro indica la posesión o la contención de las clases destino (en el extremo del rombo) por las clases origen.



Figura . Ejemplo de diagrama de clases con Enterprise Architect

**Caja de herramientas**

La Caja de herramientas de UML de Enterprise Architect se usa para crear elementos y conectores en un diagrama. Dentro de la Caja de herramientas, los elementos relativos de UML y conectores están organizados en páginas, cada página conteniendo los elementos o conectores para un tipo de diagrama en particular. Los diagramas incluyen los diagramas estándares de UML, los diagramas extendidos de Enterprise Architect, y cualquier Tecnología MDG o Perfil UML que usted haya agregado a Enterprise Architect.

Los componentes en la caja de herramientas se pueden arrastrar y soltar (Drag & Drop) al lienzo principal del diseñador. En cuanto a diagramas de clase la caja de herramientas muestra el listado de componentes específico para este tipo de diagrama. La siguiente imagen muestra la caja de herramientas para diagramas de clase.



Figura . Caja de herramientas para diagramas de clase de Enterprise Architect

**Explorador de proyecto**

La ventana del Explorador del proyecto permite navegar a través del espacio de proyectos de Enterprise Architect. Muestra paquetes, diagramas, elementos y propiedades de los elementos. Los proyectos pueden contener diferentes tipos de modelos y a su vez estos contienen diagramas, los diagramas son agrupaciones de elementos los cuales contienen propiedades.

Puede arrastrar y soltar elementos entre carpetas, o incluso soltar elementos desde el Explorador del proyecto directamente dentro del diagrama actual.

Si hace clic con el botón derecho del mouse sobre un ítem en el Explorador del Proyecto, puede llevar a cabo acciones adicionales, tales como agregar nuevos paquetes, crear diagramas, renombrar ítems, crear documentación y otros reportes y eliminar elementos del modelo.



Figura . Explorador del proyecto de Enterprise Architect

**Panel de propiedades**

La ventana Propiedades provee una forma conveniente de ver (y en algunos casos editar) las propiedades comunes de los elementos. Cuando se selecciona un elemento, la pestaña de Propiedades mostrará el nombre, estereotipo, versión, autor, fechas, y otra información pertinente del elemento.



Figura . Panel de propiedades de Enterprise Architect

**Trabajar con elementos**

Los Modelos UML se construyen desde los elementos, cada uno de los cuales tiene su propio significado, reglas y notación. Los elementos se pueden usar en diferentes estados del proceso de diseño para distintos propósitos. Los elementos básicos para diagramas de clase UML 2.1 se describen a continuación:



Figura . Elementos de diagramas de clase de Enterprise Architect

La anterior imagen contiene los diferentes elementos que Enterprise Architect permite para el diseño de diagramas de clase. Esta contiene los elementos: Paquete, Clase, Interfaz, Objeto, Tabla y Asociación.

Haciendo clic con el botón derecho del mouse sobre un elemento del diagrama se abre el menú contextual del mismo. Si selecciona dos o más elementos, se mostrará un menú contextual de selección múltiple.

El menú contextual del elemento está dividido en un número secciones y submenús:

* Propiedades
* Agregar
* Buscar
* Transforme - permite que Transforme el elemento seleccionado de un dominio a otro.
* Elementos embebidos
* Características
* Generar DDL - Genera DDL para una tabla, procedimiento o Clase
* Ingeniería de código
* Apariencia
* Ayuda UML - abre el tema de ayuda de Enterprise Architect para el tipo de elemento UML.
* Eliminar - puede eliminar el elemento desde la opción menú.



Figura . Menú contextual de una clase en Enterprise Architect

De este menú contextual podemos resaltar las opciones de Atributos, Operaciones, Generación de Código, Sincronizar con código y Ver código, estas opciones permiten al diagramador trabajar fácilmente y de manera síncrona entre el diseño visual de la clase y con el código fuente que esta representa.

Los atributos son características de una clase u otro elemento que representan las propiedades o elementos de información interna de ese elemento. Para una Clase como Usuario, nombreDeUsuario y direcciónDeUsuario pueden ser atributos. Los atributos tienen varias características importantes como el tipo, alcance (visibilidad), estática, derivada y notas.



Figura . Representación de una clase en Enterprise Architect

Enterprise Architect provee un mecanismo para agregar, editar o eliminar los atributos a una Clase, Interfaz o Enumeración. Mediante el dialogo de edición de elementos (en este caso una Clase) se puede acceder a esta opción.



Figura . Ventana de creación de atributos para una clase en Enterprise Architect

Otro aspecto importante a la hora de trabajar con elementos de diagramas de clase, son las operaciones o métodos. Estos hacen parte de Clases e Interfaces. Las operaciones son características de una clase u otro elemento que representan el comportamiento o los servicios que soportan un elemento. Para una clase Cliente, actualizarNombreDelCliente y obtenerDomicilioDelCliente pueden ser Operaciones. Las Operaciones tienen varias características importantes, tales como tipo, alcance (visibilidad), estática, abstracta y notas.



Figura . Ventana de creación de operaciones para clases en Enterprise Architect

**Trabajar con conectores**

Las conexiones UML, junto con los elementos, forman la base de los modelos UML. Las conexiones relacionan elementos entre sí, para marcar algún tipo de relación lógica o funcional entre ellos. Cada conector tiene su propio propósito, significado y notación y se utilizan en tipos específicos de diagramas UML.

Enterprise Architect proporciona los siguientes tipos de conectores para diagramas de clase.

|  |  |
| --- | --- |
| **Conector de diagrama de clase** | **Descripción** |
|  | Relación básica entre dos elementos. |
|  | Herencia entre dos clases. |
|  | Relación de composición. |
|  | Relación de agregación. |
|  | Clase asociación entre dos clases más. |
|  | Relación de unión entre 2 interfaces. |
|  | Implementación de una interfaz por una clase. |
|  | Un elemento se anida dentro de otro. Usado para representar clases internas de otras. |
|  | Incluye paquetes importados y anidados. |
|  | Importa paquetes de uno a otro. |

Tabla . Relaciones para diagramas de clase en Enterprise Architect

**Ingeniería de código**

Ingeniería de Código es un proceso que incluye generación automática de código, ingeniería inversa de código fuente y sincronización entre el código y el modelo.

*Generación de Código*

Generación de código en también conocida como Ingeniería de Código Directa. Enterprise Architect le permite generar código fuente de los elementos de los modelos UML, creando un código fuente equivalente de la clase o interface para futura elaboración y compilación. En particular puede generar código fuente en C, C++, C#, Delphi, Java, PHP, Python, ActionScript, Visual Basic and VB.NET. El código fuente generado incluye definiciones de clases, variables y funciones para cada atributo y método en la clase UML. Puede usar el Visor de código fuente para cualquier código fuente que usted esté abriendo.



Figura . Ventana de generación de código en Enterprise Architect

*Ingeniería Inversa*

Ingeniería Inversa es la importación de código fuente existente en elementos del modelo, mapeando las estructuras de código fuente en sus representaciones UML. Esto le permite examinar código antiguo y la funcionalidad de las librerías de código para reusar, o para actualizar el modelo UML con el código. Puede realizar ingeniería inversa en los mismos lenguajes mientras realiza generación de código con Enterprise Architect. Enterprise Architect también permite realizar ingeniería inversa en algunos tipos de archivos binarios: Java .JAR y .NET PE.

*Sincronización*

Sincronización es cuando los cambios en el modelo son exportados a la fuente y los cambios de la fuente son importados en el modelo. Esto le permite mantener su modelo y su código actualizados mientras el proyecto avanza.

*Ingeniería Completa*

Ingeniería completa ocurre como una combinación de generación de código inversa y directa y debe incluir sincronización entre el código fuente en todos los proyectos de ingeniería de código. Para obtener el mayor provecho de esta ingeniería en Enterprise Architect, usted debe estar familiarizado con las convenciones de modelado usadas cuando genera e ingeniería inversa de los lenguajes que usted usa.

**Plantillas de código fuente**

El Code Template Framework (CTF) le permite adaptar la manera en que Enterprise Architect genera código fuente y también le permite generar en lenguajes que Enterprise Architect no soporta específicamente ayudándole a definir las plantillas de generación de código para ese lenguaje (esto es discutido en "Enterprise Architect Software Developers' Kit" (SDK).

Las plantillas de código de EA especifican la transformación desde los elementos UML a las varias partes de un lenguaje de programación dado. Las plantillas están escritas como texto plano con una sintaxis que comparte algunos aspectos de lenguajes de marcas y de scripting. Un ejemplo simple de una plantilla usada por EA es la "plantilla de clase". Se usa para generar código fuente desde una clase UML:

%ClassNotes%   
%ClassDeclaration%   
%ClassBody%

La plantilla de arriba simplemente se refiere a otras tres plantillas, que se llaman ClassNotes, ClassDeclaration y ClassBody. Los signos delimitadores porcentaje (%) indican una "macro." Las Plantillas de Código consisten de varios tipos de macros, cada una de ellas resultantes en una sustitución en la salida que se genera. Para un lenguaje tal como C++, el resultado de procesar la plantilla de arriba podría ser:

/\*\*   
 \* Este es una nota de clase de ejemplo que se generó usando las plantillas de código  
 \* @author Sparx Systems  
 \*/   
class ClassA : public ClassB

{ .. }

**Generación de imágenes planas**

El usuario puede generar una imagen plana del diagrama y guardarla en disco, la imagen puede ser generada en los siguientes formatos:

* Windows bitmap (256 color bitmap)
* GIF image
* Windows Enhanced Metafile (standard metafile)
* Windows Placeable Metafile (older style metafile)
* PNG format
* JPG
* TGA

**Distribución automática de elementos en diagrama**

Enterprise Architect provee la facilidad de distribuir los elementos del diagrama de manera automática basada en una estructura de árbol. Algunos diagramas que debido a su estructura compleja deben ser distribuidos de manera manual por el usuario.

Esta opción está disponible para diagramas estructurales y diagramas adicionales, pero no para diagramas comportamentales de UML. Sin embargo, esta opción está disponible para diagramas de secuencia generados automáticamente por Enterprise Architect.

Si la distribución automática hecha por la herramienta no es del agrado del usuario, este tiene la posibilidad de revertir los cambios con la opción Ctrl+Z. Existen varios “Opciones de distribución o disposición” y cada uno de ellos permite organizar el diagrama en una forma específica en base a algunas configuraciones, una vez definido la manera de organizar los elementos del diagrama, solamente es necesario utilizar la opción de distribución y esta se ejecuta automáticamente, Enterprise Architect provee una configuración por defecto. A continuación se mencionan las diferentes configuraciones de disposición para los diagramas:



Figura . Ventana de opciones de disposición de Enterprise Architect

*Panel Opciones de remoción de ciclos* - Estas configuraciones determinan el tipo de eliminación de ciclos a usarse durante la organización.

* Voraz - Seleccione para usar el algoritmo de optimización de ciclos Voraz.
* Recorrido en profundidad - Seleccione para usar el algoritmo de optimización de ciclos de Recorrido en profundidad.

*Panel Opciones de disposición* - Estas configuraciones determinan el tipo de capas que se usará durante la organización.

* Mayor camino al sumidero - Seleccione para usar el algoritmo de Opciones de disposición de Mayor camino al sumidero.
* Mayor camino a la fuente - Seleccione para usar el algoritmo de Mayor camino a la fuente.
* Longitud de vínculos óptima - Seleccione para usar el algoritmo de Longitud de vínculos óptima.

*Panel Opciones de inicialización* - Estas configuraciones determinan el tipo de inicialización de índices y columnas a usarse durante la organización.

* Sencillo - Seleccione para usar el algoritmo de Opciones de inicialización, sencillo.
* DFS hacia afuera - Seleccione para usar el algoritmo de DFS hacia afuera.
* hacia dentro - Seleccione para usar el algoritmo de DFS hacia adentro.

*Panel Opciones de Reducción de Cruces*.

* Iteraciones - Ingrese el número de iteraciones a usarse durante la eliminación de bucles.
* Agresivo - Esta opción permite especificar si se usa o no un paso de reducción de cruces agresiva (consumo de tiempo de lectura). Si se marca, se usará la reducción de cruces agresiva. Si no se marca, no se usará la reducción de cruces agresiva.

*Panel Opciones de Disposición*.

* Espaciado de capas - Ingrese el número por defecto de unidades lógicas entre capas.
* Espaciado de columna - Ingrese el número por defecto de unidades lógicas entre columnas.
* Arriba, abajo, izquierda, derecha - Seleccionar la dirección al cual deberían dirigirse los vínculos.
* Configurar como proyecto predeterminado.

Seleccione esta casilla de verificación para aplicar las configuraciones de disposición del diagrama a todos los diagramas en el proyecto. Si marca esta casilla y presiona el botón Aceptar para un diagrama diferente, las nuevas configuraciones sustituirán las configuraciones guardadas anteriormente.

**Importación o Exportación a XMI**

Un paquete puede ser importado desde un archivo XMI (basado en XML). Esto permite que los elementos de los modelos de Enterprise Architect puedan ser movidos entre distintos modelos, para el desarrollo distribuido, manual control de versiones y otros beneficios.

Los siguientes formatos pueden ser importados:

* UML 1.3 (XMI 1.0)
* UML 1.3 (XMI 1.1)
* UML 1.4 (XMI 1.2)
* UML 2.0 (XMI 2.1)
* UML 2.1 (XMI 2.1)
* MOF 1.3 (XMI 1.1)
* MOF 1.4 (XMI 1.2)

Enterprise Architect también puede importar los archivos \*.emx and \*.uml2 generados por herramientas como por ejemplo Rational Software Architect (RSA) and Rational Software Modeler (RSM).

Para importar un paquete desde XMI se necesita hacer lo siguiente:

1. En el Explorador del proyecto, seleccione el paquete en el cual desea importar el archivo.
2. Puede:

* Hacer clic en el botón derecho y seleccionar la opción Importar/Exportar | Importar paquete desde XMI o
* Seleccione la opción Proyecto | Importar/Exportar | Importar paquete desde XMI.

Se abre la ventana Importar paquete desde XMI.



Figura . Ventana de importación de archivos XMI en Enterprise Architect

Se puede exportar un Paquete a un archivo XMI (basado en XML). Esto permite que los elementos del Modelo de EA se muevan entre modelos, para desarrollo distribuido, control de versiones y otros beneficios. También permite una exportación limitada de elementos de modelo de EA a Rational Rose y a otras herramientas que implementen los estándares UML 2.1 XMI 2.1 o UML 1.3 XMI 1.2/XMI 1.1 / XMI 1.0.

Para exportar un paquete a XML, proceda con los siguientes pasos:

1. En el Explorador del proyecto, seleccione el paquete que quiere exportar.
2. Puede:

* Hacer con el botón derecho y seleccionar el menú Importar/Exportar | Exportar el paquete a XMI, o
* Seleccionar la opción Proyecto | Importar/Exportar | Exportar el paquete a XMI.

Se abre la ventana Exportar paquete a XMI.



Figura . Ventana de exportación de archivos XMI en Enterprise Architect

## STARUML

StarUML™ es una plataforma de modelado de software que soporta UML (Unified Modeling Language). Se basa en la versión 1.4 de UML versión y proporciona 11 tipos diferentes de diagrama, y acepta la notación UML 2.0. Apoya activamente el enfoque de MDA (Model Driven Architecture), apoyando el concepto de perfil UML. StarUML destaca en la personalización del entorno del usuario y tiene una alta extensibilidad en su funcionalidad.

### Historia

En un principio StarUML fue formalmente conocido como “Plastic” o “Agora Plastic”. En 1996 se conoce la primera versión de “Plastic” (v0.9), se trataba de una herramienta muy simple para dibujar módulos de software y sus dependencias. En 1997 la versión 1.0 de Plastic fue liberada, se caracterizó por ser una herramienta gratuita con soporte para OMT.

La versión 1.1 de Plastic fue liberada en 1998, con la principal característica que incluía soporte para diagramas de clase UML. En 1999 se funda la empresa Plastic Software, Inc, junto con la liberación de la versión 2.0 de su herramienta Plastic. Esta versión de la herramienta cubría parcialmente el estándar UML, generación de código en Java e ingeniería inversa.

En 2001 se libera la versión 3.0 de Plastic en donde se soporta completamente el estándar UML en su versión 1.3. Sin embargo en 2003 la herramienta Plastic es completamente rediseñada y reescrita, logrando un soporte completo de UML 1.4 mediante una arquitectura abierta.

En 2005 Agora Plastic es liberado, incluyendo características de internacionalización y muchas otras para extender la plataforma. En ese mismo año recibe la certificación de ‘Good Software’ de parte del Ministerio de Información y comunicación de Korea. A finales de 2005 la herramienta fue renombrada a StarUML 5.0 y fue liberada, se convirtió en un proyecto open source con soporte completo de UML 2.0 y se implementó una notación de tecnología extensible, de tal manera que se pueden desarrollar complementos en varios lenguajes y añadirlos a la aplicación.

### Características Generales

La siguiente tabla muestra las características generales de esta herramienta de modelado. Esta información fue tomada explícitamente de la guía de usuario de StarUML [11].

| **Característica** | **Descripción** |
| --- | --- |
| Estándar UML | StarUML se adhiere estrictamente a la especificación estándar UML especificado por el OMG para el modelado de software. Teniendo en cuenta el hecho de que los resultados de la información de diseño pueden llegar a 10 años o más en el futuro, la dependencia específica del proveedor sintaxis UML irregular y la semántica puede ser muy arriesgado. StarUML maximiza la misma para ordenar estándar UML 1.4 y significado, y acepta la notación UML 2.0, sobre la base de meta-modela robusto. |
| Formato de modelo Open Source | A diferencia de muchos productos existentes que gestionan sus propios modelos de formato heredado ineficiente, StarUML gestiona todos los archivos en el formato XML estándar. Códigos escritos en estructuras fáciles de leer y sus formatos se pueden modificar convenientemente usando un analizador XML. Dado que XML es un estándar mundial, este es sin duda una gran ventaja. |
| Soporte real de MDA | StarUML verdaderamente soporta UML Profile. Esto maximiza la extensibilidad de UML, haciendo el modelado de aplicaciones posibles, incluso en áreas como las finanzas, la defensa, el comercio electrónico, los seguros y la aeronáutica. Verdaderamente modelos independientes de plataforma (PIM) pueden ser creados, y modelo específicos de la plataforma (PSM) y códigos ejecutables pueden ser generados de forma automática de cualquier modo. |
| Aplicación de metodologías y plataformas | StarUML manipula el concepto de enfoque, creando de ambientes que se adaptan a cualquier metodología o proceso. No sólo los modelos de Frameworks de aplicación para plataformas como. NET y J2EE, sino también las estructuras básicas de los modelos de software (por ejemplo, 4+1 View Model, etc.) se puede definir fácilmente. |
| Extensibilidad | Todas las funciones de las herramientas de StarUML están automatizadas acorde a Microsoft COM. Cualquier lenguaje que soporte COM (Visual Basic Script, JavaScript, VB, Delphi, C++, C#, VB.NET, Python, etc.) puede ser usado para manejar StarUML y desarrollar elementos integrados Add-In. |
| Función de verificación de modelos de software | Los usuarios pueden cometer muchos errores durante el modelamiento de software. Estos errores pueden ser muy costosos si permanecen hasta la etapa final de codificación. Con el objetivo de prevenir este problema, StarUML automáticamente verifica los modelos de software desarrollados por el usuario, facilitando descubrir de manera temprana errores, permitiendo software más completo y con menos faltas. |
| Add-Ins útiles. | StarUML incluye algunos Add-Ins útiles con varias funcionalidades. Estos generan código fuente en varios lenguajes de programación y convierten código fuente en modelos, importan archivos de Rational Rose, intercambian información de modelos con otras herramientas usando XMI, y soportan diseño de patrones. Estos Add-Ins proporcionan reusabilidad adicional, productividad, flexibilidad e interoperabilidad para la información de modelado. |

Tabla . Características principales de StarUML

### Funciones para Diagramas de Clase

StarUML permite muchos elementos para el diseño de diagramas de clase, entre ellos se encuentran: Clases, Interfaces, Enumeraciones, Paquetes, Subsistemas, Signal, Exception, Port, Asociación, Asociación Directa, Agregación, Composición, Generalización, Realización, Clase Asociación, entre otros.

La siguiente imagen representa la paleta de componentes disponibles en StarUML para el diseño de diagramas de clase.



Figura . Caja de herramientas para diagramas de clase en StarUML

**Distribución de la pantalla**

La siguiente muestra la distribución de los paneles en la pantalla principal de la herramienta StarUML. Como se puede observar esta sigue la distribución común de la mayoría de las herramientas CASE, los componentes o elementos pueden ser añadidos desde la paleta hasta el lienzo donde se está editando los diagramas, se puede hacer edición de varios diagramas al tiempo y estos son agregados como pestañas del lienzo principal.

****

Figura . Distribución de los paneles principales en StarUML

**Herramientas de Edición**

StarUML dispone de herramientas básicas y útiles para la edición de los diagramas de clase, y más específicamente para los elementos que hacen parte del diagrama. Las siguientes funcionalidades permites trabajar con cualquier tipo de elemento del diagrama.

* Drag & Drop: Permite arrastrar los elementos desde la paleta hacia el lienzo.
* Zoom: Permite acercar o alejar la vista del lienzo principal de la aplicación.
* Copiar: Permite copiar elementos del diagrama y almacenarlos de manera temporal en el portapapeles.
* Cortar: Permite quitar los elementos del diagrama y almacenarlos de manera temporal en el portapapeles.
* Pegar: Permite pegar los elementos almacenados en el portapapeles directamente al diagrama en edición.

**Atributos y Operaciones**

Los atributos y operaciones (métodos) de las clases, interfaces o enumerados, se pueden añadir a los elementos mediante el menú contextual que emerge al dar click derecho sobre alguno de ellos, adicional a esto, en menú dispone de la mayoría de acciones que se pueden realizar con dicho elemento.

La siguiente imagen muestra el menú contextual de una clase añadida a un diagrama.



Figura . Menú contextual de una clase en StarUML

Como se puede ver, aquí se puede abrir la ventana para manipular los atributos y operaciones, adicionalmente se puede generar el código fuente o realizar ingeniería inversa en 3 diferentes lenguajes de programación: C++, C# y Java. También permite aplicar patrones básicos de diseño al elemento.

**Formato de Textos**

La herramienta permite definir formatos de textos para los elementos, estos se aplican a los nombres de las clases, o nombres de los atributos y métodos. Adicionalmente también permite establecer el tipo de fuente usada, los colores de las líneas y algunas otras configuraciones. Este menú contextual también permite realizar distribución automática de los elementos en el diagrama, de tal manera que puedan ser ordenados según algunos criterios.



Figura . Menú contextual para formato de elementos en StarUML

**Generación de código**

Para la generación de código StarUML dispone de un asistente (Wizard) para el proceso, el cual dispone de varios pasos en los cuales se configura el paquete de elementos a generar, los elementos específicos en caso de generar solamente algunos, la ruta donde se localizaran los archivos y por ultimo unas opciones generales de formato para los archivos generados. Los archivos de código generado por defecto pueden ser lenguaje C++, C# o Java, sin embargo es posible añadir otros lenguajes instalando los Add-Ins necesarios para este propósito.

La siguiente imagen muestra el último paso del asistente de generación de código, después de seleccionar los elementos a generar y la ubicación de los archivos resultantes.



Figura . Asistente para generación de código Java en StarUML

**Ingeniería Inversa**

La ingeniería inversa es el proceso de generar un modelo a partir del código fuente, para esta actividad la herramienta también dispone de un asistente, el cual guía al usuario de tal manera que puede realizar el proceso de manera fácil e intuitiva.

Las siguientes dos imágenes muestran el primer y último paso del asistente, en los cuales se seleccionan los archivos de código fuente y se configura las opciones de generación del modelo respectivamente.



Figura . Primer paso del asistente de ingeniería inversa en StarUML



Figura . Ultimo paso del asistente de ingeniería inversa en StarUML

**Exportación e Importación**

La herramienta dispone de varios mecanismos de importación y exportación. En el caso de importación permite realizar ingeniería inversa de otras herramientas como Rational Rose o desde archivos XMI que contienen la información del diagrama o modelo. Para exportación se permite únicamente generar archivos XMI.



Figura . Ventana de importación de archivos XMI en StarUML



Figura . Ventana de exportación de archivos XMI en StarUML

## ARGOUML

ArgoUML es un entorno de diseño de dominio-orientado que proporciona apoyo cognitivo para el diseño orientado a objetos. ArgoUML proporciona algunas de las mismas características de automatización de una herramienta CASE comercial, pero se centra en las características que apoyan las necesidades cognitivas de los diseñadores.

Estas necesidades se describen por tres teorías cognitivas:

* Reflexión en acción.
* Diseño oportuno.
* Comprensión y Resolución de problemas.

ArgoUML está basado directamente en la especificación 1.4 de UML. El modelo Core del repositorio es una implementación de la Java Metadata Interface (JMI) la cual directamente soporta MOF y usa versión legible por máquina de la especificación 1.4 de UML provista por el OMG.

La versión actual de ArgoUML (0.34) implementa todos los tipos de diagramas del estándar UML 1.4. Este es escrito en Java y corre sobre cualquier computador que provea la versión 5 o más reciente de la plataforma Java. Este usa el formato de archivo abierto XMI para almacenamiento interno.

ArgoUML está disponible de manera gratuita y puede ser usado en ambientes comerciales. El código fuente está disponible para revisión y personalización que el usuario requiera, así como para realizar mejoras [12].

### Funciones para Diagramas de Clase

**Distribución de la pantalla**

ArgoUML tiene una manera especial de distribución de la pantalla principal, en donde la paleta de componentes se reduce a una barra de herramientas la cual posee cada elemento como un icono diferente, al clicar sobre el icono se selecciona para luego ser añadido al lienzo. Al igual que otras herramientas posee un árbol de proyecto y un panel de propiedades un poco más extenso, el cual se compone de varias pestañas y en cada una se realizan configuraciones diferentes, tal como añadir atributos y operaciones o renombrar algunas propiedades, realizar vinculaciones con otros elementos mediante conectores y muchas otras funcionalidades. La siguiente imagen muestra la distribución mencionada.



Figura . Distribución de los paneles principales de ArgoUML

**Generar Imagen del Diagrama**

La herramienta permite guardar una imagen plana del diagrama que está siendo editado, aunque únicamente en formato JPG esta opción es de mucha utilidad para realizar documentaciones o presentaciones. La siguiente imagen muestra una captura de la aplicación al generar una imagen de un diagrama de ejemplo.



Figura . Ventana de generación de imágenes en ArgoUML

**Exportación e Importación**

La exportación e importación se puede realizar en archivos de formato XMI, adicionalmente se puede realizar ingeniería inversa desde código fuente. La importación y exportación se realizar de manera muy básica, solamente seleccionando el nombre y la ubicación del archivo del cual o al cual se desea cargar la información.

La siguiente imagen muestra la ventana de exportación a XMI, como se puede observar es una opción básica que permite guardar el archivo en una ruta especifica de la maquina local.



Figura . Ventana de exportación XMI de ArgoUML

# DIAGRAMAS DE CLASES

Los diagramas de clase hacen parte de la vista estática o estructural de una aplicación según el estándar UML, estos representan los conceptos del universo del discurso (dominio del problema a resolver) en el cual se enfoca la aplicación, y la relación que existe entre estos conceptos. Dichos conceptos son modelados como clases, cada una de las cuales describe un conjunto de objetos discretos que almacenan información y se comunican para implementar un comportamiento. La información que almacenan es modelada como atributos; el comportamiento que realizan es modelado como operaciones.

Varias clases pueden compartir una estructura común usando generalización. Una clase hija añade nuevas estructuras y comportamientos a las estructuras y comportamientos que obtiene por herencia de la clase padre. Los objetos también tienen conexión durante la ejecución con otros objetos individuales. Tales relaciones “Objeto a Objeto” son modeladas como asociaciones entre clases. Algunas relaciones entre elementos se agrupan como relaciones de dependencia, incluyendo las relaciones para modelar desplazamientos en los niveles de abstracción, enlace de parámetros del modelo, concesión de permisos, y uso de un elemento por parte de otro. Hay muchos tipos de elementos en los diagramas UML, tales como interfaces, tipos de datos, clases y señales. En conjunto, se les llama clasificadores, y se comportan de forma muy parecida a las clases, con ciertas restricciones en cada tipo de clasificador [3].

Se dice que la vista de los diagramas de clase es estática porque no describe el comportamiento del sistema dependiendo del tiempo, el cual si es descrito por otras vistas. Como se mencionó anteriormente, los componentes principales de esta vista son las clases y sus relaciones: *asociación*, *generalización*, y varios tipos de dependencia, tales como *realización* y *uso*.

Las clases son el centro alrededor del cual se organiza la vista de clases; otros elementos pertenecen o se unen a las clases. Las clases de dibujan como rectángulos. Las listas de atributos y de operaciones se muestran en compartimientos separados. Los compartimientos pueden ser suprimidos cuando no es necesario el detalle completo. Una clase puede aparecer en varios diagramas. Sus atributos y operaciones se suprimen, a menudo en todos menos en un diagrama.

Las relaciones entre las clases se dibujan como las líneas que conectan rectángulos de clases. Las diversas clases de relaciones se diferencian por la textura de las líneas y por los adornos en las líneas o sus extremos.

Las clases se pueden describir con varios niveles de precisión y concreción. Al empezar el diseño, el modelo captura los aspectos más lógicos del problema. En las fases posteriores, el modelo también capta decisiones de diseño y detalles de la implementación.

Los elementos clave en la vista estática son los *clasificadores* y sus *relaciones*. Un clasificador es un elemento de modelo que describe cosas. Hay varias clases de clasificadores, como las Clases, Interfaces, y Tipos de Datos. Los aspectos de comportamiento son materializados por otros clasificadores, como los casos de uso y las señales. Los propósitos de implementación están detrás de varias clases de clasificadores, tales como subsistemas, componentes y nodos.

## CLASIFICADORES

Una clasificador es un concepto discreto en el modelo, que tiene identidad, estado, comportamiento, y relaciones. Los tipos de clasificadores incluyen la Clase, la Interfaz, y los Tipos de Datos. Otras clases de clasificadores son materializaciones de conceptos de comportamiento, de cosas del entorno, o de estructuras de implementación. Estos clasificadores incluyen casos de uso, actor, componente, nodo, y subsistema. El siguiente diagrama de clase fue tomado de la especificación del estándar UML 2.4.1, y describe la jerarquía de clases del meta modelo UML iniciando en la clase raíz *Element* [13].

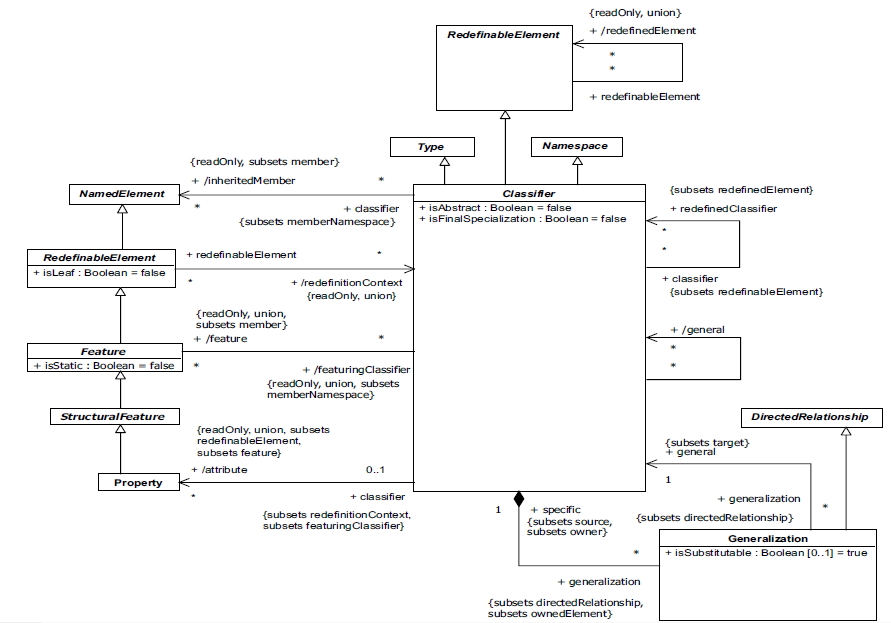


Figura . Diagrama de clasificadores del meta modelo UML [13]

Todas las clases del meta modelo UML son hijas de la case *Element*, esta clase abstracta engloba cualquier tipo de figura o elemento que hace parte de cualquier diagrama UML. En el diagrama anterior las clases *RedefinableElement* y *NamedElement* son hijas directas de la case *Element*.

### Clase

Una clase representa un concepto discreto dentro de la aplicación que se está modelando; una cosa física (tal como un aeroplano), una cosa de negocios (tal como un pedido), una cosa lógica (tal como un horario de difusión), una cosa de una aplicación (tal como un botón de cancelar), una cosa del computador (tal como una tabla hash), o una caso del comportamiento (tal como una tarea). Una clase es el descriptor de un conjunto de objetos con una estructura, comportamiento, y relaciones similares. Todos los atributos y operaciones están unidos a clases o a otros clasificadores. Las clases son los focos alrededor de los cuales se organizan los sistemas orientados a objetos.

Las clases son un tipo de Clasificador, esto significa que dentro del metamodelo UML, estas extienden directamente de la clase *Classifier*, heredando sus atributos y operaciones.



Figura . Notación UML para una clase

Como se puede ver tiene 3 compartimientos, en el primero se coloca el nombre de la case, en el segundo los atributos y en el tercero las operaciones o métodos. La representación en código Java de esta clase simple se puede ver a continuación:

/\*\*

\* Documentación en formato JavaDoc de la case NombreClase.

\*

\* **@author** Gabriel Leonardo Diaz, 27.09.2013.

\*/

**package** examples;

**public** **class** NombreClase {

}

El siguiente diagrama ilustra una parte de las clases del meta modelo UML donde se definen los atributos y relaciones de una clase. Tomado directamente del documento de especificación del estándar UML.

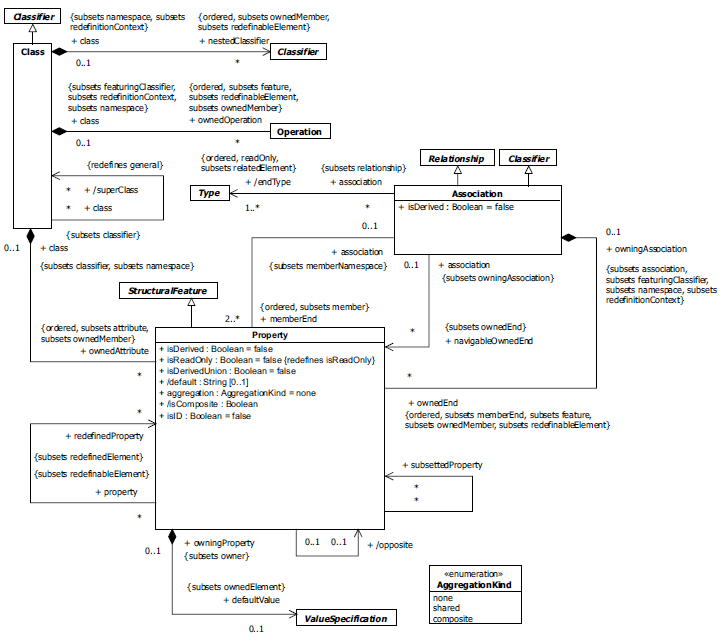


Figura . Paquete de clases del meta modelo UML [13]

Como se puede ver la clase *Class* extiende directamente de *Classifier*, lo cual significa que hereda todos los atributos y comportamientos de dicha clase y adicionalmente añade algunos otros componentes.

### Atributos

Los atributosdefinen la estructura de una clase y de sus correspondientes objetos. El atributo define el valor de un dato para todos los objetos pertenecientes a una clase. Ejemplo: *Nombre*, *edad*, *peso*, son atributos de la clase *Persona*. *Color*, *precio*, *modelo*, son atributos de la clase *Automóvil*. Los atributos corresponden a sustantivos y sus valores pueden ser sustantivos o adjetivos.

Se debe definir un valor para cada atributo de una clase, los valores pueden ser iguales o distintos en los diferentes objetos. No se puede dar un valor en un objeto si no existe un atributo correspondiente en la clase. Ejemplo: el valor del atributo *edad* puede ser "24" para los objetos Juan Pérez y María López, y "15" para Ramón Martínez.

Dentro de una clase, los nombres de los atributos deben ser únicos (aunque puede aparecer el mismo nombre de atributo en diferentes clases). Ejemplo: Las clases *Persona* y *Compañía* pueden tener ambas un atributo *dirección*, en cambio no pueden existir dos atributos llamados *dirección* dentro de la clase *persona*.

Los atributos no tienen ninguna identidad, al contrario de los objetos. Ejemplo: Los atributos *nombre* y *edad* de la clase *persona* tienen valores simples. El valor para *nombre* puede ser "Juan" o "María", mientras que el valor para *edad* puede ser "17" o "25". (Nótese que pudieran existir dos objetos distintos con exactamente el mismo *nombre* y *edad*, donde estos identificarían dos personas distintas).

Los atributos comienzan siempre con una letra minúscula, aunque las siguientes palabras en el caso de nombres compuestos, pueden comenzar con mayúsculas.

Como con los nombres de clases, no debe haber espacios dentro del nombre y en especial no debe haber nombres repetidos. Por ejemplo, consideremos agregar atributos a la clase mencionada anteriormente.



Figura . Notación UML para una clase con atributos

La representación en código de una clase con atributos seria la siguiente:

/\*\*

\* Documentación en formato JavaDoc de la case NombreClase.

\*

\* **@author** Gabriel Leonardo Diaz, 27.09.2013.

\*/

**package** examples;

**public** **class** NombreClase {

**private** **int** atributoTipoEntero;

**private** String atributoTipoCadena;

**private** **float** atributoTipoFlotante;

}

El orden de los atributos no tiene ninguna importancia dentro de la clase. Nótese que los tipos de los atributos no necesariamente tienen que ser tipos primitivos, como es el caso de String.

### Operaciones

Las *operaciones* son funciones o transformaciones que se aplican a todos los objetos de una clase particular. La operación puede ser una acción ejecutada por el objeto o sobre el objeto. Ejemplo: *Arrojar*, *atrapar*, *inflar*, y *patear*, son operaciones para la clase *pelota*. *Abrir*, *cerrar*, *ocultar*, y *dibujar*, son operaciones para la clase *ventana*.

Las operaciones deben ser únicas dentro de una misma clase, aunque no necesariamente para diferentes clases. Las operaciones pueden tener *argumentos*, o sea, una lista de *parámetros*, cada uno con un tipo, y pueden también devolver resultados, cada uno con un tipo. Las operaciones se incorporan en la tercera sección de la clase.



Figura . Notación UML de una clase con atributos y operaciones

La representación en código de esta clase seria como:

/\*\*

\* Documentación en formato JavaDoc de la case NombreClase.

\*

\* **@author** Gabriel Leonardo Diaz, 27.09.2013.

\*/

**package** examples;

**public** **class** NombreClase {

// Atributos

**private** **int** atributoTipoEntero;

**private** String atributoTipoCadena;

**private** **float** atributoTipoFlotante;

// Constructor

**public** NombreClase () {

}

// Operaciones o métodos

**public** **void** metodo () {

}

**public** **void** metodoConParametro (**int** parametro) {

}

**public** String metodoConRetorno () {

**return** **null**;

}

}

Aquí vemos los elementos básicos de una clase, con atributos y métodos. Este ejemplo tiene un tipo especial de método llamado Constructor, este método se caracteriza porque no tiene tipo de retorno.

Aunque conceptualmente se habla de operaciones, en los lenguajes de programación es más preciso hablar de métodos. La relación entre estos dos términos es que múltiples métodos pueden corresponder a una misma operación. La lista de métodos anterior está compuesta por el tipo de valor de retorno, *tipoRetorno*, el nombre del método, *nombreMétodo*, los parámetros que recibe el método, *listaParámetros*, y finalmente el cuerpo del método, *nombreCuerpo*. (Nuevamente, los “...” son únicamente para resaltar que es una lista de métodos.) Nótese que los nombres de los métodos comienzan siempre con una letra minúscula, aunque las siguientes palabras en el caso de nombres compuestos, pueden comenzar con mayúsculas. Como con los nombres de clases y atributos, no debe haber espacios dentro del nombre.

### Interfaz

Una interfaz es la descripción del comportamiento de objetos sin dar su implementación o estado; una interfaz contiene operaciones pero no atributos, y no tiene asociaciones salientes que muestren la visibilidad desde la propia interfaz.

Una interfaz puede restringir la manera en la cual sus operaciones son invocadas y puede imponer pre y post condiciones a sus operaciones. Cada interfaz a menudo especifica solo una parte limitada del comportamiento de una clase.

Las interfaces pueden tener incluso relaciones de Generalización. Una Interfaz hija incluye todo el contenido de su ancestro pero puede añadir contenido adicional (al igual que las Clases).

Una interfaz puede tener la siguiente estructura:

* Atributos: Valores de estado que deben ser mantenidas por el clasificador que implementa dicha interfaz, no necesariamente como valores de atributos. A menudo en los lenguajes de programación, las Interfaces no pueden tener atributos variables, solo constantes (por ejemplo en Java). Además de esto, es común que el clasificador implementador de la interfaz es siempre una clase.
* Operaciones: Servicios que objetos inespecíficos o anónimos pueden invocar de un objeto implementación de la interfaz. Una interfaz puede no tener métodos para implementar en sus operaciones.
* Recepciones: Señales que objetos anónimos pueden enviar al objeto implementador.
* Restricciones: Precondiciones, Pos condiciones, u otros tipos de restricciones en la invocación a los servicios del objeto implementador.

Una interface puede tener asociaciones con otras interfaces. Esto significa que una asociación de conformidad debe existir entre instancias de los clasificadores que “Realizan” las interfaces (es decir, clasificadores que implementan la interfaz).

Si un clasificador implementa una interfaz, entonces debe declarar o heredar todas las operaciones en la interfaz. Este puede contener operaciones adicionales, ver el concepto *Realización* en la siguiente sección.



Figura . Notación UML para una interfaz

La representación en código es la siguiente:

**package** examples;

/\*\*

\* Documentacion en formato JavaDoc de la interfaz NombreInterfaz

\*

\* **@author** Gabriel Leonardo Diaz, 27.09.2013.

\*/

**public** **interface** NombreInterfaz {

**public** **static** **final** **int** ATRIBUTO\_CONSTANTE = 5;

**public** **void** metodoDeInterfaz();

}

### Enumeración

Un tipo de dato cuyas instancias forman parte de una lista de valores literales con nombre. Usualmente, tanto el nombre de la enumeración y sus valores literales son declarados. Una enumeración es un tipo de dato definible por el usuario. Este tiene un nombre y una lista ordenada de valores literales de enumeración, cada uno es un valor en el rango del tipo de dato.

Los literales de la enumeración pueden ser comparados para comprobar igualdad o para la posición relativa en la lista de literales. Estos literales son atomicos y no tienen subestructura.

Una enumeración se representa de la siguiente manera en gráficos UML:



Figura . Notación UML para una enumeración

La respectiva implementación en código es la siguiente:

/\*\*

\* Documentacion en formato JavaDoc de la enumeracion NombreEnumeracion.

\*

\* **@author** Gabriel Leonardo Diaz, 27.09.2013.

\*/

**public** **enum** NombreEnumeracion {

LITERAL\_UNO,

LITERAL\_DOS,

LITERAL\_TRES

}

### Paquete

Un mecanismo de propósito general para organizar elementos en grupos, estableciendo una jerarquía de elementos, y proporcionando nombres únicos para referenciar elementos.

Un paquete es un grupo de elementos de modelos y diagramas. Cada elemento de modelo que no es parte de otro elemento debe ser declarado en exactamente un *Namespace*; el *Namespace* que contiene la declaración de un elemento es llamado el propietario del elemento. Un paquete es un *Namespace* de propósito general que puede ser propietario de cualquier tipo de elemento de modelos que no está restringido a un tipo de elemento particular.

Un paquete tiene la siguiente notación en UML:



Figura . Notación UML para un paquete

En los lenguajes de programación los paquetes son representados mediante una instrucción específica, la cual es añadida en el encabezado de los elementos (Clases, Interfaces, Enumerados) y determinan el paquete en los cuales todos están ubicados. En los ejemplos mostrados anteriormente el paquete siempre era *examples.*

## RELACIONES

Las relaciones entre clasificadores son: Asociación, Generalización, Flujo y varias clases de dependencia, que incluyen la Realización y el Uso.

### Asociación

Una asociación es una relación estructural que especifica que los objetos de una clase están conectados con los objetos de otra. Cuando una asociación es una conexión semántica entre los objetos de dos clases, se llama asociación binaria. Aunque no es lo común, se pueden tener asociaciones que conecten más de dos clases; éstas se llaman asociaciones n-arias. También es posible que, dado un objeto de una clase, se pueda enlazar con otros objetos de la misma clase.

Normalmente una asociación es binaria y bidireccional (se puede navegar en ambas direcciones). Se dibuja como una línea sólida entre dos clases. También se puede representar una asociación binaria y unidireccional, añadiendo una flecha al final de la línea. Esta flecha indica que la asociación sólo se puede utilizar en la dirección de la flecha (este concepto se conoce como Navegabilidad).

Cada asociación puede presentar algunos elementos adicionales que dan detalle a la relación, como son:

* **Nombre**: Una asociación puede tener un nombre, que se usa para describir la naturaleza de la relación. Así no hay ambigüedad sobre su significado. Para indicar la dirección en que se debe leer el nombre se emplea un triángulo.
* **Rol**: Cuando una clase participa en una asociación, juega un rol específico en dicha relación. Se puede designar de forma explícita mediante un nombre a los finales de la línea, el cual describe la semántica de la asociación en el sentido indicado.
* **Multiplicidad**: La multiplicidad describe la cardinalidad de la relación, es decir, cuántos objetos están conectados en una instancia de una asociación (enlace). Cuando se establece una multiplicidad al final de la línea de una asociación, indica que para cada objeto de la clase en el lado opuesto existen varios objetos en el otro extremo. El rango puede ser tres (3), cero-a-uno (0..1), cero-a-muchos (0..\* ó \*), uno-a-muchos (1..\*), etc.
* **Restricciones**: Los elementos anteriores son suficientes para detallar una relación de asociación. Pero si queremos especificar un mayor significado, UML define cinco restricciones que se pueden aplicar. Por ejemplo, si queremos que los objetos de una clase al final de una asociación (con una multiplicidad mayor que uno) estén en un orden explícito, debemos escribir {ordenado} al final de la misma y cerca de la clase.



Figura . Notación UML para una Asociación simple

La representación en código de este tipo de relación es la siguiente.

**package** examples;

/\*\*

\* Documentacion en formato JavaDoc de la clase ClaseA

\*

\* **@author** Gabriel Leonardo Diaz, 02.10.2013.

\*/

**public** **class** ClaseA {

**private** **int** atributoEnA;

**private** ClaseC objClaseC; // Objeto de tipo ClaseC

**public** **void** metodoEnA () {

// Vacio

}

}

**package** examples;

/\*\*

\* Documentacion en formato JavaDoc de la clase ClaseC

\*

\* **@author** Gabriel Leonardo Diaz, 02.10.2013.

\*/

**public** **class** ClaseC {

**private** **boolean** atributoEnC;

**private** ClaseA objClaseA; // Objeto de tipo ClaseA

**public** **void** metodoEnC () {

// Vacio

}

}

### Generalización

La herencia es uno de los conceptos fundamentales de la programación orientada a objetos, en la que una clase «recoge» todos los atributos y operaciones de la clase de la que es heredera, y puede alterar/modificar algunos de ellos, así como añadir más atributos y operaciones propias. En UML, una asociación de generalización entre dos clases, coloca a estas en una jerarquía que representa el concepto de herencia de una clase derivada de la clase base. En UML, las generalizaciones se representan por medio de una línea que conecta las dos clases, con una flecha en el lado de la clase base.



Figura . Notación UML para una Generalización

En la imagen anterior se muestra un caso básico de generalización, en donde la clase ClaseB extiende de ClaseA, por lo tanto hereda los atributos y métodos de la misma. A continuación se muestra la representación en código de este diagrama.

**package** examples;

/\*\*

\* Documentacion en Javadoc para ClaseA.

\*

\* **@author** Gabriel Leonardo Diaz, 27.09.2013.

\*/

**public** **class** ClaseA {

**private** **int** atributoEnA;

**public** **void** metodoEnA() {

}

}

**package** examples;

/\*\*

\* Documentación en JavaDoc para ClaseB.

\*

\* **@author** Gabriel Leonardo Diaz, 27.09.2013.

\*/

**public** **class** ClaseB **extends** ClaseA {

**private** **int** atributoEnB;

}

La generalización es utilizada por clases y casos de uso y por otros elementos de modelado como los paquetes. La generalización normal es una relación de herencia entre clases, entre una general y otra específica. La clase específica, llamada subclase, hereda todo de la clase general, llamada superclase. Los atributos, las operaciones y todas las asociaciones son heredados.

Los atributos y las operaciones con visibilidad pública en la superclase serán también públicos en la subclase. Los atributos y las operaciones que tengan visibilidad privada también serán heredados, pero no serán accesibles dentro de la subclase. Para proteger los atributos y las operaciones de accesos desde fuera de la superclase y de la subclase, los podemos declarar con visibilidad protegida. No se puede acceder a ellos desde otras clases, pero están disponibles para la superclase y para cualquiera de sus subclases. La relación de generalización se representa por una línea continua desde la clase más específica (subclase) hasta la clase más general (superclase), con un triángulo sin rellenar al final de la línea en la superclase. Una clase abstracta es aquella que no tiene objetos. Sólo se utiliza para heredar a partir de ella, es decir, en una clase abstracta se describen los atributos y las operaciones comunes para otras clases. Se especifica de forma explícita poniendo {abstracta} dentro del compartimento del nombre de la clase y debajo de su nombre. Normalmente una clase abstracta tiene operaciones abstractas. Una operación abstracta es aquella que no tiene método de implementación en la superclase donde está definida. Se especifica de forma explícita escribiendo {abstracta} después del nombre de la operación en la superclase. Lo contrario de una clase abstracta es una clase concreta, lo que significa que es posible crear objetos a partir de la clase y que tiene implementaciones para todas las operaciones.

### Realización

Representa una relación entre una *especificación* y su *implementación*; indica la herencia de comportamiento sin la herencia de estructura. Una especificación describe el comportamiento o estructura de algo sin determinar como el comportamiento será implementado. Una implementación provee los detalles de cómo implementar el comportamiento en una manera algorítmica efectiva. La relación entre un elemento que especifica el comportamiento y uno que provee la implementación es llamada *Realización*. Elementos tales como Interfaces y Casos de Uso, son tipos de especificaciones, los cuales son pensados especificar el comportamiento, y ellos no contienen información sobre la implementación.

El elemento implementador debe soportar todos los comportamientos incluidos en el elemento especificador. Por ejemplo, una clase debe soportar todas las operaciones de las interfaces que el implementa. La clase puede implementar operaciones adicionales, y la implementación de las operaciones pueden hacer cosas adicionales, una vez se provee la implementación de las operaciones, las interfaces no son violadas.

La relación de Realización es mostrada como una línea punteada con una flecha triangular cerrada en el final, adyacente al elemento que provee la especificación y en la cola el elemento que provee la implementación.



Figura . Notación UML para una Realización

Como se puede ver en la figura anterior, existe una interfaz que define una operación que debe ser implementada, la clase que se relaciona con la interfaz mediante la Realización es la encargada de proveer dicha implementación.

La representación en código de este tipo de relación se muestra a continuación.

**package** examples;

/\*\*

\* Documentacion en formato JavaDoc de la interfaz NombreInterfaz

\*

\* **@author** Gabriel Leonardo Diaz, 02.10.2013.

\*/

**public** **interface** NombreInterfaz {

**public** **static** **final** **int** *ATRIBUTO\_CONSTANTE* = 5;

**public** **void** metodoDeInterfaz ();

}

**package** examples;

/\*\*

\* Documentacion en formato JavaDoc de la clase ClaseImplementacion

\*

\* **@author** Gabriel Leonardo Diaz, 02.10.2013.

\*/

**public** **class** ClaseImplementacion **implements** NombreInterfaz {

@Override

**public** **void** metodoDeInterfaz() {

// Implementacion

}

}

La implementación del método definido por la interfaz es provista por la clase que se relaciona con esta mediante una Realización.

### Agregación

La agregación es un tipo de asociación que indica que una clase es parte de otra clase (composición débil). Los componentes pueden ser compartidos por varios compuestos (de la misma asociación de agregación o de varias asociaciones de agregación distintas). La destrucción del compuesto no conlleva la destrucción de los componentes. Habitualmente se da con mayor frecuencia que la composición. La agregación se representa en UML mediante un diamante de color blanco colocado en el extremo en el que está la clase que representa el “todo”. A continuación se mencionan los tipos de agregación que existen.

**Agregación Normal**: Una agregación normal se denota dibujando una línea con un rombo sin rellenar al final de la misma del lado del todo (del lado de la clase que contiene a la otra clase). También puede tener un nombre (con dirección) y ser navegable, así como añadir roles en el lado de la parte (lado de la clase contenida).

La siguiente figura muestra un ejemplo de este tipo de relación:



Figura . Notación UML para una Agregación Normal

La representación en código de este tipo de relación seria la siguiente.

**package** examples;

/\*\*

\* Documentacion en formato JavaDoc de la clase ClaseParte

\*

\* **@author** Gabriel Leonardo Diaz, 02.10.2013.

\*/

**public** **class** ClaseParte {

**private** **int** atributoId;

**private** **String** nombre;

}

**package** examples;

/\*\*

\* Documentacion en formato JavaDoc de la clase ClaseTodo

\*

\* **@author** Gabriel Leonardo Diaz, 02.10.2013.

\*/

**public** **class** ClaseTodo {

**private** Collection<ClaseParte> partes = **new** ArrayList<ClaseParte>();

**public** **void** agregarParte(ClaseParte parte) {

**this**.partes.add(parte);

}

**public** **void** borrarParte (ClaseParte parte) {

**this**.partes.remove(parte);

}

}

Cabe aclarar que aunque en la figura anterior el todo tiene una multiplicidad de 1, la representación en código mostrada anteriormente no cumple con esta restricción, ya que esto requiere un mecanismo manual de control para garantizar que una de las partes pertenezca únicamente a un todo. Esta característica se puede apreciar mucho mejor con el siguiente tipo de agregación.

**Agregación Compartida**: Una agregación compartida es aquella en la que las partes pueden ser partes en varios todos. Se representa de la misma forma que una agregación normal. Para que una agregación se considere compartida es necesario que la multiplicidad en el lado del todo sea superior a uno. Una agregación compartida es un caso especial de una agregación normal.

****

Figura . Notación UML para una Agregación Compartida

La representación en código es prácticamente la misma que la mencionada en el anterior tipo, y no es necesario añadir ningún control para verificar que las partes pertenezcan únicamente a un solo todo, debido a que estas pueden pertenecer a varios de ellos por definición.

### Composición

Una composición se denota dibujando una línea con un rombo relleno al final de la misma del lado del todo (del lado de la clase que contiene a la otra clase). La multiplicidad en el lado del todo puede ser uno (1) o cero-a-uno (0..1), pero la multiplicidad en el lado de la parte puede ser cualquier intervalo. Se trata de una relación de pertenencia muy fuerte.



Figura . Notación UML para una Composición

La representación en código del diagrama seria la siguiente:

**package** examples;

/\*\*

\* Documentacion en formato JavaDoc de la clase ClaseParte

\*

\* **@author** Gabriel Leonardo Diaz, 02.10.2013.

\*/

**public** **class** ClaseParte {

**private** **int** atributo;

**private** String nombre;

**public** ClaseParte() {

// Constructor vacio

}

**public** ClaseParte (**int** atributoId, String nombre) {

**this**.atributo = atributoId;

**this**.nombre = nombre;

}

**public** **int** getAtributo() {

**return** atributo;

}

**public** String getNombre() {

**return** nombre;

}

}

**package** examples;

**import** java.util.ArrayList;

**import** java.util.Collection;

/\*\*

\* Documentacion en formato JavaDoc de la clase ClaseTodo

\*

\* **@author** Gabriel Leonardo Diaz, 02.10.2013.

\*/

**public** **class** ClaseTodo {

**private** Collection<ClaseParte> partes;

**public** ClaseTodo() {

**this**.partes = **new** ArrayList<ClaseParte>();

}

**public** **void** agregarParte(**int** atributoId, String nombre) {

**this**.partes.add(**new** ClaseParte(atributoId, nombre));

}

**public** **void** borrarParte (**int** atributoId) {

ClaseParte parte = **null**;

// Busqueda comparando el campo atributoId

**this**.partes.remove(parte);

}

**public** ClaseParte buscarParte (**int** atributoId) {

ClaseParte parte = **null**;

// Busqueda comparando el campo atributoId

**return** parte;

}

}

Cabe aclarar que esta representación incluye muchos otros elementos y conceptos que no se pueden abstraer del diagrama dibujado, tal como el tipo de colección a utilizar, la instanciación de la colección, el agregado de las instancias de las partes al todo, y algunas otras.

La mayoría de las herramientas CASE no pueden generar por si solos una composición, es necesario agregar cada elemento tal como el atributo que almacena las partes, especificar el tipo de colección a utilizar y agregar los métodos para mantenimiento de las partes en el todo. Esto puede ser debido a la gran ambigüedad que existe entre lo que se modela y la manera de convertir dicho modelo a código, inclusive debido a la manera en la cual cada lenguaje de programación maneja las colecciones y sus operaciones.

### Clase Asociación

Cuando una asociación tiene propiedades propias se representa como una clase unida a la línea de la asociación por medio de una línea a trazos. Tanto la línea como el rectángulo de clase representan el mismo elemento conceptual: la asociación. Por tanto ambos tienen el mismo nombre, el de la asociación. Cuando la clase asociación sólo tiene atributos el nombre suele ponerse sobre la línea. Por el contrario, cuando la clase asociación tiene alguna operación o asociación propia, entonces se pone el nombre en la clase asociación y se puede quitar de la línea.



Figura . Notación UML para una Clase Asociación

Este concepto de Clase Asociación es utilizado al momento de realizar el modelo de análisis del software en construcción, una vez el modelo de análisis es llevado a un modelo de diseño, la clase asociación cambia su notación y se convierte en dos asociaciones independientes que conectan las clases extremo con la clase intermedia.

La representación en código es muy similar a la ya mencionada en la sección 5.2.1.

### Dependencia

Una dependencia es una conexión semántica entre dos elementos de modelado, uno independiente y otro dependiente. Un cambio en el elemento independiente afectará al dependiente. Un elemento de modelado puede ser una clase, un paquete, un caso de uso, etc. Los casos más típicos de dependencia son: una clase tiene por parámetro en una operación un objeto de otra clase; una clase accede a un objeto global de otra clase; o una clase llama a una operación cuyo ámbito es la clase en otra clase. La relación de dependencia se representa por una línea discontinua con una flecha en el lado del elemento independiente. Si se utiliza una etiqueta, entonces se trata de un estereotipo que identifica el tipo de dependencia.



Figura . Notación UML para una Dependencia

La figura 42 muestra el caso de dependencia mencionado en el párrafo anterior, en donde la ClaseDependiente recibe como parámetro un objeto del tipo ClaseIndependiente. La dependencia se genera debido a que algún cambio en la clase independiente (por ejemplo, remover un atributo utilizado por la clase dependiente) obliga a realizar cambios en la otra clase.

## CONCEPTOS AVANZADOS

### Multiplicidad

La multiplicidad es el número de instancias que tiene una clase en relación con otra clase pueden ser de varias maneras: con un número fijo, indica que no puede ser más de lo que se indica, con un intervalo de valores 2…5, otra forma es la de un número que indica el comienzo y al final un asterisco, este indica que empieza con el número que se indica y el asterisco que pueden ser muchos o infinito, solo el asterisco indica que puede tomar cualquier valor (cero o más) , también se puede usar la combinación de cualquier manera siempre y cuando se encuentren separados de comas.

|  |  |
| --- | --- |
| **Notación** | **Lectura** |
| 1 | Exactamente Uno |
| \* | Muchos |
| 0…1 | Cero a uno |
| 0…\* | Cero a muchos |
| 1…\* | Uno a Muchos (al menos uno) |
| M…N | De M hasta N (enteros naturales) |

Tabla . Tipos de multiplicidad de una relación UML

### Navegabilidad

La navegabilidad indica en qué orden fluyen los datos en el diagrama se indica por medio de una flecha que se pone al final de una line de asociación apuntando hacia la clase que recibe los datos en la figura representa la navegabilidad que existe entre la clase pedido y la de cliente, esta representa una asociación unidireccional también hay otra que es la bidireccional, esta indica que existe navegabilidad en ambas direcciones.

### Visibilidad

En el estándar UML existen 4 tipos de clasificaciones de visibilidad para atributos y métodos: *Public*, *Protected*, *Package* y *Private*.

* **Public (+)**: Indica que el atributo o método será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.
* **Private (-)**: Indica que el atributo o método sólo será accesible desde dentro de la clase (sólo sus métodos lo pueden usar).
* **Protected (#)**: Indica que el atributo o método no será accesible desde fuera de la clase, pero si podrá ser usado por métodos de la clase además de las subclases que se deriven (herencia).
* **Package (~)**: Indica que el atributo o método solo puede ser usado por la clase misma o por clases que se encuentren en el mismo paquete.

### Roles

Para indicar el papel que juega una clase en una asociación se puede especificar un nombre de rol. El rol provee un nombre para identificar cada lado de una Asociación (también llamado “final de asociación”), además de ser usado para navegar de un objeto a otro usando la Asociación.

Todos los roles en una asociación deben ser diferentes. En una auto Asociación (una asociación que involucra la misma clase más de una vez), el rol es necesario para remover la ambigüedad de cada lado de la asociación entre la misma clase.

# CLASIFICACIÓN DE LA INFORMACIÓN

## MATRIZ DE CARACTERISTICAS

A continuación se mencionan las características que poseen las herramientas para el diseño de diagramas de clase. Adicionalmente se seleccionan aquellas que serán susceptibles de implementación en el proyecto.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Característica** | | **EA** | **StarUML** | **ArgoUML** | **Resultado** |
| Edición | Portapapeles (Copiar, Cortar, Pegar) | X | X | X | Seleccionado |
| Arrastrar y Soltar (D&D) | X | X | X | Seleccionado |
| Formato de Textos | X | X |  |  |
| Ordenamiento (Traer al frente, enviar atrás) | X | X | X | Seleccionado |
| Renombrado inmediato | X | X | X | Seleccionado |
| Alinear elementos | X | X | X | Seleccionado |
| Zoom | X | X | X | Seleccionado |
| Vista Miniatura | X |  |  |  |
| Atajos de teclado | X | X | X | Seleccionado |
| Deshacer-Rehacer | X | X |  | Seleccionado |
| Generación de Código Fuente | | X | X | X | Seleccionado |
| Ingeniería Inversa | | X | X | X |  |
| Importación/Exportación XMI | | X | X | X | Seleccionado |
| Generar Imágenes a partir del diagrama | | X | X | X | Seleccionado |
| Sincronización con código fuente | | X |  |  |  |
| Edición de código fuente | | X |  |  |  |
| Control de Versiones | | X |  |  |  |
| Generación de Documentación a partir del diagrama | | X | X |  |  |
| Distribución de elementos automáticamente | | X | X |  |  |
| Plantillas de código para Lenguajes adicionales. | | X | X |  |  |

Tabla . Matriz de características para diagramas de clase

## CARACTERISTICAS PROPIAS

Si bien es cierto que nuestra aplicación posee características compartidas y comunes por la mayoría de herramientas de modelado para diagramas de clase, también es cierto que provee algunas características que son propias y fueron consideradas para aportar un valor agregado a la herramienta. A continuación se listan estas características.

* Compartir diagramas entre los usuarios de la aplicación
* Diseñar de manera concurrente un diagrama específico al cual acceden dos o más usuarios diferentes.

Estas dos características representan las funcionalidades propias de la aplicación, las cuales otorgan a la herramienta un grado de singularidad.

## LISTA DE FUNCIONALIDADES

En base al análisis hecho anteriormente con respecto a las características de cada una de las herramientas case estudiadas, se abstrajo la siguiente lista de funcionalidades que serán susceptibles de implementación en el proyecto, estas servirán como insumo principal en la fase de exploración para determinar las historias de usuario del proyecto.

* Registrar usuarios
* Crear diagramas para los usuarios
* Compartir diagramas entre diferentes usuarios
* Diseñar o dibujar de manera concurrente un diagrama
* Generar código fuente a partir de un diagrama
* Generar imágenes planas a partir de un diagrama
* Exportar el diagrama en formato XMI
* Importar un diagrama en formato XMI.

Las características mencionadas anteriormente representan de manera global el alcance del proyecto y los servicios que la aplicación ofrecerá a los usuarios una vez se encuentren en producción.

Adicionalmente existen características un poco más específicas que se deben considerar al momento de dibujar diagramas en el diseñador. Estas características proporcionan una mejor experiencia de usuario y permiten diseñar de manera fácil diagramas de manera conjunta.

* Herramienta Zoom
* Portapapeles (Copiar, Cortar y Pegar elementos del diagrama)
* Deshacer y Rehacer (Undo-Redo)
* Alinear elementos

# METODOLOGÍA DE DESARROLLO

La metodología de desarrollo seleccionada para el proyecto fue *eXtreme Programming* (XP), debido a su naturaleza y al tamaño del equipo de trabajo se puede adaptar fácilmente al desarrollo del proyecto, considerando que brinda flexibilidad y la posibilidad de adaptación a los cambios que pueden surgir en el transcurso del proyecto.

## EXPLORACIÓN

### Identificación de Roles del Proyecto

El desarrollo del proyecto será llevado a cabo por un grupo pequeño de personas, sin embargo cada una tiene una labor específica e importante para la correcta ejecución del mismo. A continuación se describen los roles del proyecto y su participación en el proyecto:

* **Desarrollador**: Gabriel Leonardo Díaz Cárdenas, estudiante de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander.
* **Cliente**: En un proceso de desarrollo ágil es de vital importancia para el desarrollo del proyecto, sin embargo en nuestro caso no tenemos un ente o persona específica a la cual se le entregará la aplicación, en su lugar tenemos un grupo general de personas a las cuales irá dirigido el proyecto:

“Estudiantes del Plan de Estudios de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander, en especial, estudiantes que cursan materias como Programación Orientada a Objetos y Análisis y Diseño de sistemas. Adicionalmente los Docentes que hacen parte de estas cursos del pensum académico”.

* **Director de Proyecto**: Marco Antonio Adarme Jaimes, Msc. en Computación. Director general del proyecto quien estará controlando su ejecución y orientando el proceso de implementación, además quien estará verificando que se cumplan los objetivos planteados al inicio del proyecto.
* **Asegurador de la Calidad**: Las labores de verificación y pruebas del funcionamiento de la aplicación estarán a cargo del Msc. Margo Antonio Adarme Jaimes, a quien se le entregaran las liberaciones de cada iteración (mencionadas más adelante) y quien dará el visto de aprobación.
* **Consultor**: Este rol es asumido por personas externas, expertos en algunos de los temas que vamos a tratar durante la ejecución del proyecto. A continuación se mencionan las personas que ejercen este rol en el proyecto:
  + **Gaudenz Alder**: Cofundador de la empresa JGraph Ltd, creadora de la librería JavaScript mxGraph utilizada para el funcionamiento de este proyecto. Gaudenz inició JGraph como su proyecto de tesis para el instituto tecnológico de Suiza, Zurich en el 2000. Ha liderado el desarrollo del proyecto para la empresa en 13 años continuos de experiencia en el diseño de la librería.
  + **PrimeFaces Community**: Este agente externo está compuesto por un grupo de desarrolladores alrededor del mundo que se encargan de resolver problemas relacionados con el uso del FrameWork JSF PrimeFaces, y permiten a usuarios novatos aprender fácilmente el uso de la herramienta.

### Identificación de Actores del Sistema

En UML se define un *actor* como aquella persona, proceso o cosa que interactúa con un sistema, subsistema o clase [3]. Para nuestro proceso de desarrollo es de vital importancia determinar cuáles son los actores de nuestro sistema, de modo que se pueda tener una idea clara de las necesidades conjuntas y particulares de cada uno de ellos. A continuación se mencionan cada uno de los actores y su descripción.

|  |  |  |
| --- | --- | --- |
| Visitante.bmp | **VISITANTE** | |
| Representa cualquier persona que llega a la aplicación en búsqueda de información, ya sea de manera accidental o voluntaria. Este actor no tiene una representación real dentro de la aplicación y solamente es manejado de manera conceptual, en efecto cualquier persona inicia siendo un visitante de la aplicación. | |
| **Clase** | --- |
| **Súper Clase** |  |
| **Interacción con la aplicación** | | |
| * Búsqueda de Información o curiosidad. * Registrarse en el sistema. | | |

Tabla . Especificación del actor *Visitante*

|  |  |  |
| --- | --- | --- |
| Visitante.bmp | **USUARIO** | |
| Representa una persona que inicia sesión dentro de la aplicación y que le es permitido usar el diseñador de diagramas. Este es un actor abstracto y necesita ser redefinido en una representación concreta. Dentro de la aplicación Usuario es una interfaz implementada por dos clases. | |
| **Clase** | *User* |
| **Súper Clase** |  |
| **Interacción con la aplicación** | | |
| * Inicio de Sesión * Cerrar Sesión * Diseñar diagrama * Generar Código * Generar Imagen * Exportar XMI | | |

Tabla . Especificación del actor *Usuario*

|  |  |  |
| --- | --- | --- |
| Visitante.bmp | **INVITADO** | |
| Un invitado en la aplicación es la representación de un usuario que utiliza la opción de *Ver Demostración* e inicia sesión dentro de la aplicación sin registro previo. Este actor posee una representación real dentro de la aplicación, sin embargo sus privilegios son limitados y no tiene acceso a algunas funcionalidades. | |
| **Clase** | *Guest* |
| **Súper Clase** | *User* |
| **Interacción con la aplicación** | | |
| * Todas las interacciones heredadas * Ver demostración: Esta opción puede ser usada por cualquier visitante, sin embargo en el momento de ingresar a la aplicación dicho visitante se convierte en un usuario invitado. | | |

Tabla . Especificación del actor *Invitado*

|  |  |  |
| --- | --- | --- |
| Visitante.bmp | **DIAGRAMADOR** | |
| Representa una persona registrada en la aplicación mediante el formulario designado para tal labor. A esta persona se le asigna una cuenta usuario mediante su dirección de correo electrónico, y por medio de la cual ingresa a la aplicación de manera controlada para hacer uso de todos los servicios que están a su disposición. | |
| **Clase** | *Diagrammer* |
| **Súper Clase** | *User* |
| **Interacción con la aplicación** | | |
| * Todas las interacciones heredadas * Ver tablero de control * Gestionar diagramas (Crear, editar, borrar, copiar) * Compartir diagramas * Importar XMI * Gestionar Perfil * Cambiar Contraseña * Desactivar cuenta * Reactivar cuenta * Recuperar contraseña * Edición concurrente de diagramas | | |

Tabla . Especificación del actor *Diagramador*

Dentro de la aplicación no se dispone de un usuario administrador, ya que no existen tareas o configuraciones especiales que se tengan que hacer de manera controlada. Cada usuario es capaz de interactuar con la aplicación de la misma manera y todos ellos disponen de los mismos privilegios. Esto evita la necesidad de tener a una persona realizando configuraciones en la aplicación y la libera de este tipo de dependencia. Ambos *Invitado* y *Diagramador* son *Usuarios* de la aplicación debido a que pueden iniciar sesión dentro de ella e ingresar al diseñador. Los actores pueden ser representados mediante el siguiente diagrama.



Diagrama . Diagrama de actores de la aplicación CLASS Modeler

### Historias de Usuario

A continuación se describen las historias de usuario recolectadas para la implementación del proyecto. El siguiente formato [12] dispone de la información correspondiente a cada historia de usuario, incluyendo la prioridad, la estimación de tiempo y la iteración a la cual fue asignada, estos conceptos son explicados en la sección 7.2.

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H1 | **Nombre de historia:** Crear cuenta de diagramador | |
| **Usuario:** Visitante | | **Iteración Asignada:**  Iteración 1 |
| **Prioridad:** Media | | **Estimación:** 1 Semana |
| **Descripción:**  Un visitante puede crear una cuenta de diagramador para iniciar sesión en el sistema. Para ello, dispone de un formulario de registro donde se captura información básica del visitante, como los nombres, apellidos, correo electrónico, género y contraseña. | | |
| **Reglas u Observaciones:**   * La dirección de correo electrónico de cada diagramador debe ser única en el sistema, de modo que la aplicación debe controlar y verificar que se cumpla esta condición. El correo electrónico es usado como identificador de la cuenta. * La contraseña ingresada debe tener mínimo 5 caracteres de longitud y máximo 20. Puede incluir cualquier combinación de letras y números. * El correo electrónico es único para todo el sistema, por lo tanto un diagramador no puede registrarse con un correo electrónico ya usado por otra persona. * El género se usa para asignar una imagen por defecto al diagramador una vez crea su cuenta, esta imagen puede ser cambiada a gusto de la persona en cualquier momento. * Una vez creada la cuenta de diagramador, esta permanece INACTIVA hasta que se realice la activación respectiva mediante el correo de confirmación. Este correo se envía a la dirección ingresada al momento de crear la cuenta. * Al momento de crear la cuenta de diagramador el sistema debe generar un código de verificación encriptado, el cual se envía junto con el correo de activación y sirve para validar los datos al momento de ejecutar el proceso. * El código de activación debe ser único por cada solicitud. * El diagramador no puede iniciar sesión en el sistema hasta que active su cuenta. | | |

Tabla . Formato de historia de usuario H1

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H2 | **Nombre de historia:** Activar cuenta de diagramador | |
| **Usuario:** Diagramador | | **Iteración Asignada:** Iteración 1 |
| **Prioridad:** Media | | **Estimación:** 1 Semana |
| **Descripción:**  Este es un requerimiento para poder usar la aplicación una vez se creada la cuenta de diagramador. La activación de la cuenta se hace mediante un link que se envía a la dirección de correo que el visitante ingresó al momento de su registro, este link contiene la información necesaria para validar los datos y un código único encriptado para brindar seguridad. | | |
| **Reglas u Observaciones:**   * El código de activación tiene un periodo de vigencia de 2 días, por lo tanto si la persona no realiza la activación dentro de ese tiempo el código debe caducar y se debe generar uno nuevo. * El código de activación es válido solamente una vez. Por lo tanto una vez utilizado ya se será válido nunca más. * Las cuentas de diagramador pueden tener 3 estados diferentes: INACTIVA el cual se obtiene cuando recién se crea la cuenta de diagramador, ACTIVADA que es el estado normal de una cuenta vigente y DESACTIVADA el cual indica que la cuenta esta temporalmente invalida y no puede ser usada, esto último hecho por voluntad del diagramador. * La cuenta podrá ser activada únicamente si esta existe y el código de verificación concuerda con el guardado en base de datos y además si el código de verificación no ha caducado. * Cuando el código esta vencido la aplicación debe generar uno nuevo y enviarlo automáticamente a la dirección de correo del diagramador, adicionalmente debe informarle que se ha generado un nuevo código. * Una vez realizada la activación de la cuenta, el usuario puede ingresar normalmente a la aplicación por medio de la opción iniciar sesión. | | |

Tabla . Formato de historia de usuario H2

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H3 | **Nombre de historia:** Desactivar cuenta de diagramador | |
| **Usuario:** Diagramador | | **Iteración asignada:** Iteración 9 |
| **Prioridad:** Baja | | **Estimación:** 1 Semana |
| **Descripción:**  Por medio de esta opción el diagramador podrá desactivar su cuenta temporalmente en caso de así quererlo. Cuando se desactiva una cuenta de diagramador, se mantiene toda la información que el usuario ya tenía, pero no le es permitido iniciar sesión en la aplicación, tampoco le podrán ser compartidos diagramas de otras personas y no recibirá notificaciones de lo que suceda con los diagramas que actualmente tiene compartidos. | | |
| **Reglas u Observaciones:**   * Para desactivar la cuenta el diagramador debe iniciar sesión y utilizar la opción disponible para tal labor. * Antes de desactivar la cuenta se debe confirmar la acción para evitar operaciones accidentales. * Al momento de desactivar la cuenta se le envía un correo al diagramador con la información pertinente, aclarando que no recibirá más notificaciones de la aplicación. * Una vez desactivada la cuenta, el diagramador no podrá ingresar al sistema. | | |

Tabla . Formato de historia de usuario H3.

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H4 | **Nombre de historia:** Reactivar cuenta de diagramador | |
| **Usuario:** Diagramador | | **Iteración asignada:** Iteración 9 |
| **Prioridad:** Baja | | **Estimación:** 1 Semana |
| **Descripción:**  Esta es la operación inversa a la descrita en H3, el usuario reactiva su cuenta mediante un correo de confirmación en donde se le envía un código encriptado y la información necesaria auto contenida en un link, lo único que debe hacer el usuario es clicar el link y lo llevará automáticamente a la aplicación donde la cuenta debe estar ACTIVADA nuevamente. | | |
| **Reglas u Observaciones:**   * Este proceso se ejecuta cuando el diagramador intenta iniciar sesión (H6) en la aplicación pero esta detecta que ha sido desactivada, en este caso la aplicación le informa al usuario que esta ha sido desactivada y le pregunta que si desea volver a activarla. * En caso de que el diagramador desee volver a activar la cuenta, entonces se le envía un correo electrónico con un código de verificación encriptado, mediante el cual se valida la información necesaria. * La activación se realiza tal cual como se menciona en H2. | | |

Tabla . Formato de historia de usuario H4

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H5 | **Nombre de historia:** Reasignar contraseña | |
| **Usuario:** Diagramador | | **Iteración Asignada:** Iteración 1 |
| **Prioridad:** Baja | | **Estimación:** 1 Semana |
| **Descripción:**  Un diagramador puede reasignar una nueva contraseña a su cuenta en caso de haber olvidado la anterior. El proceso consiste en ingresar la dirección de correo y de manera controlada se le envía un email de confirmación con un link para hacer la reasignación. El link contiene la información necesaria para validar la cuenta y lleva al diagramador a un formulario donde puede asignar una nueva contraseña para su cuenta. | | |
| **Reglas u Observaciones:**   * Las reglas de validación del código de verificación deben ser las mismas que las ya mencionadas anteriormente. * Una vez validado el código de verificación se le debe mostrar al diagramador un formulario donde debe ingresar la nueva contraseña y la confirmación, las reglas para la contraseña son las mismas descritas en H1. * La nueva contraseña y la confirmación deben coincidir. * La contraseña anterior no puede ser recuperada y no es necesario validar que la nueva contraseña sea diferente a la anterior. * Una vez los datos sean validados se procede a cambiar la contraseña actual por la nueva. | | |

Tabla . Formato de historia de usuario H5

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | | |
| **ID:** H6 | **Nombre de historia:** Iniciar sesión | |
| **Usuario:** Diagramador o Invitado | | **Iteración Asignada:** Iteración 2 |
| **Prioridad:** Media | | **Estimación:** 2 Semanas |
| **Descripción:**  Funcionalidad mediante la cual un usuario ingresa a la aplicación. Esta operación puede ser llevada a cabo de dos maneras diferentes, un usuario registrado (diagramador) que ingresa a la aplicación con su dirección de correo o un usuario invitado que ingresa a la aplicación sin registro previo.  Para los usuarios registrados se dispone de un formulario de ingreso, donde se solicita la dirección de correo electrónico y la contraseña. | | |
| **Reglas u Observaciones:**   * Para poder iniciar sesión con una cuenta creada es necesario que esta cuenta tenga el estado ACTIVADA, por lo tanto el usuario debe activar su cuenta antes de ingresar (únicamente en el caso de los diagramadores). * Si la cuenta está INACTIVA o DESACTIVADA el usuario no podrá iniciar sesión. * Los usuarios tipo invitado iniciar sesión para entrar en modo demostración a la aplicación, el cual se realiza sin necesidad de registro previo. * Considerar temas de seguridad, para proteger el acceso no autorizado a páginas de la aplicación que requieren el inicio de sesión. | | |

Tabla . Formato de historia de usuario H6

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H7 | **Usuario:** Diagramador |
| **Nombre de historia:** Modificar perfil de usuario | |
| **Descripción:**  Esta opción permite cambiar la información de registro del usuario, tal como nombres, apellidos, fecha de nacimiento, sexo y avatar (imagen de identificación). No se puede cambiar la dirección de correo electrónico del usuario. | |
| **Reglas u Observaciones:**   * La dirección de correo electrónico no puede ser cambiada. | |

Tabla . Formato de historia de usuario H7

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H8 | **Usuario:** Diagramador |
| **Nombre de historia:** Cambiar contraseña | |
| **Descripción:**  Permite cambiar la contraseña del usuario, es necesario siempre ingresar la contraseña anterior. El formulario solicita una nueva contraseña y una confirmación para garantizar que el usuario puede recordarla fácilmente. Solo se podrá cambiar la contraseña si la información requerida es correcta, es decir, si la anterior contraseña coincide con la guardada en base de datos y la nueva contraseña y su confirmación son exactamente iguales. | |
| **Reglas u Observaciones:**   * Se debe solicitar la contraseña anterior para proceder a cambiarla. * El cambio se hace mediante un formulario que tiene 3 campos, la contraseña anterior, la nueva contraseña y una confirmación de la nueva contraseña. * Las reglas para la nueva contraseña deben ser las mismas ya mencionadas. | |

Tabla . Formato de historia de usuario H8

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H9 | **Usuario:** Diagramador |
| **Nombre de historia:** Crear diagrama | |
| **Descripción:**  Esta opción le permite a un usuario crear un diagrama para sí mismo. Los datos básicos de entrada son solamente el nombre y una descripción opcional del diagrama. No se realiza ninguna validación especial aparte de que exista un nombre. El usuario quien crea el diagrama es asignado como propietario del mismo y puede compartirlo con otros. Inicialmente la representación del diagrama en XMI es vacía, solamente contiene un package (paquete) principal que tiene el mismo nombre del diagrama y no tiene elementos hijos. | |
| **Reglas u Observaciones:**   * Se debe solicitar el nombre del diagrama y una descripción opcional. * Se debe disponer de una página en donde el diagramador pueda ver todos los diagramas que tiene creados. * El diagramador puede crear tantos diagramas como quiera. * El nombre del diagrama no debe ser necesariamente único, es decir, varios diagramas pueden tener el mismo nombre; el diagramador decide colocar el nombre que quiera. | |

Tabla . Formato de historia de usuario H9

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H10 | **Usuario:** Diagramador |
| **Nombre de historia:** Compartir diagrama | |
| **Descripción:**  Permite al usuario registrado compartir diagramas con otros usuarios del sistema. Esta opción es permitida únicamente para los diagramas de los cuales el usuario tenga privilegios.  En general existen 4 diferentes tipos de privilegios ordenados de menor a mayor rango: READ\_ONLY (solo lectura), EDIT (edición), SHARE (volver a compartir) y OWNER (Propietario), este último se otorga por defecto al usuario quien crea el diagrama. Básicamente un usuario solamente podrá compartir diagramas de su propia autoría o cuando se le ha sido asignado el privilegio de SHARE, en cuyo caso el usuario es un segundo propietario del diagrama. Como se mencionó anteriormente, un usuario con su cuenta INACTIVADA o DESACTIVADA no podrá recibir ningún diagrama compartido. | |
| **Reglas u Observaciones:** | |

Tabla . Formato de historia de usuario H10

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H11 | **Usuario:** Diagramador |
| **Nombre de historia:** Gestionar privilegios del diagrama | |
| **Descripción:**  Esta opción le permite al propietario del diagrama modificar o quitar los privilegios otorgados a otros usuarios (ya sea por el mismo o por otros), mediante el uso de la funcionalidad descrita en H10. Básicamente el usuario puede remover los privilegios otorgados sobre un diagrama, lo que significa que el usuario afectado no podrá seguir haciendo uso de este, o también puede modificar los privilegios con el objetivo de otorgar al otro usuario mayor o menor control sobre el diagrama. Esta operación solo puede ser ejecutada por el propietario del diagrama, los demás usuario solo pueden ver quienes acceden a dicho diagrama pero no pueden hacer ningún cambio. | |
| **Reglas u Observaciones:** | |

Tabla . Formato de historia de usuario H11

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H12 | **Usuario:** Diagramador |
| **Nombre de historia:** Editar diagrama | |
| **Descripción:**  Permite cambiar los datos de creación del diagrama (nombre y descripción). Para poder ejecutar esta opción es necesario que el usuario tenga los privilegios necesarios para ello.  El privilegio necesario para ejecutar esta operación debe ser mayor que READ\_ONLY, por lo tanto un usuario a quien solo se le ha permitido ver el diagrama no podrá hacer uso de esta opción. | |
| **Reglas u Observaciones:**   * Se puede cambiar el nombre y la descripción del diagrama. | |

Tabla . Formato de historia de usuario H12

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H13 | **Usuario:** Diagramador |
| **Nombre de historia:** Eliminar diagrama | |
| **Descripción:**  Permite eliminar un diagrama de manera controlada. | |
| **Reglas u Observaciones:**   * Se debe confirmar el borrado antes de proceder con la operación. * Los diagramas solo pueden ser borrados por el propietario. * Si un diagramador tiene un diagrama compartido y desea borrarlo, en realidad se le quita el privilegio otorgado y no se borra el diagrama original. * Si el propietario del diagrama decide borrarlo, entonces se le debe notificar que otros diagramadores pueden verse afectados y que serán informados mediante correo electrónico sobre la operación realizada. | |

Tabla . Formato de historia de usuario H13

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H14 | **Usuario:** Diagramador |
| **Nombre de historia:** Copiar diagrama | |
| **Descripción:**  Esta opción permite copiar un diagrama al cual se tiene acceso (compartido o propio). Los datos son copiados exactamente pero el propietario de la copia será quien realiza la opción. Los diagramas compartidos pueden ser copiados pero ninguno de los usuarios podrá ver el nuevo diagrama a menos que se comparta explícitamente.  Un usuario solo puede copiar un diagrama si tiene el privilegio para editarlo, por lo tanto un usuario a quien solamente se le ha otorgado privilegios de lectura no podrá hacer uso de esta opción. | |
| **Reglas u Observaciones:** | |

Tabla . Formato de historia de usuario H14

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H15 | **Usuario:** Diagramador |
| **Nombre de historia:** Diseñar diagrama | |
| **Descripción:**  Permite abrir el Diseñador y editar los elementos UML del diagrama de clase. Dentro de este diseñador se encuentran todas las funciones básicas para manejar los diagramas, agregar elementos, editarlos, copiarlos, agregar comentarios.  Esta historia de usuario engloba todas las opciones disponibles en el diseñador relacionadas con el pintado de diagramas, a continuación se mencionan algunas de ellas:   * Añadir, editar, borrar elementos del diagrama. Clases, Interfaces, Enumeraciones, Relaciones, Comentarios, etc. * Agregar variables y métodos a los elementos respectivos. * Usar las opciones de edición: Copiar, Pegar, Cortar, Deshacer, Rehacer. * Zoom. | |
| **Reglas u Observaciones:** | |

Tabla . Formato de historia de usuario H15

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H16 | **Usuario:** Diagramador |
| **Nombre de historia:** Diseñar diagrama concurrentemente | |
| **Descripción:**  Esta opción permite a varios usuarios hacer uso de un diagrama de manera concurrente, reflejando los cambios hechos por uno o por otro en tiempo real. Para el usuario el uso de esta funcionalidad es transparente ya que no necesita realizar ninguna operación explicita adicional a la descrita en la historia H15. Sin embargo en el diseñador se les informa a los usuarios sobre quienes están haciendo uso del diagrama al mismo tiempo. | |
| **Reglas u Observaciones:** | |

Tabla . Formato de historia de usuario H16

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H17 | **Usuario:** Invitado |
| **Nombre de historia:** Ver demostración del diseñador | |
| **Descripción:**  Un invitado de la aplicación podrá hacer uso de ella sin registro previo, con el objetivo de obtener información o verificar los beneficios que esta puede prestarle. En la página de bienvenida se encuentra un link de ingreso, este link redirige al usuario al *Diseñador* de diagramas y crea un diagrama vacío, a partir de allí el usuario podrá hacer modificaciones a su antojo. Sin embargo, algunas funcionalidades están restringidas, como el compartir diagramas, guardarlos, general código fuente o imágenes. Únicamente se podrán hacer exportaciones en formato XMI del diagrama construido. Esta opción está estrechamente relacionada con H15. | |
| **Reglas u Observaciones:** | |

Tabla . Formato de historia de usuario H17

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H18 | **Usuario:** Diagramador |
| **Nombre de historia:** Generar código fuente | |
| **Descripción:**  Esta opción le permite al usuario generar el código fuente del diagrama que se encuentra editando mediante el Diagramador. El código fuente generado es únicamente en lenguaje Java. Los archivos fuentes pueden ser descargados uno a uno, o descargar un empaquetado que contenga todos los archivos generados. | |
| **Reglas u Observaciones:** | |

Tabla . Formato de historia de usuario H18

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H19 | **Usuario:** Diagramador |
| **Nombre de historia:** Generar imagen del diagrama | |
| **Descripción:**  Permite obtener una representación plana del diagrama en una imagen con diferentes tipos de formatos (.gif, .png, .jpg). Esta opción es muy útil para realizar documentaciones o impresiones del diagrama completo. | |
| **Reglas u Observaciones:** | |

Tabla . Formato de historia de usuario H19

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H20 | **Usuario:** Diagramador |
| **Nombre de historia:** Importar XMI | |
| **Descripción:**  Permite importar un diagrama desde un archivo en disco en formato estándar XMI. La importación del archivo XMI reemplaza la representación actual del diagrama. | |
| **Reglas u Observaciones:** | |

Tabla . Formato de historia de usuario H20

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | |
| **ID:** H21 | **Usuario:** Diagramador |
| **Nombre de historia:** Exportar XMI | |
| **Descripción:**  Permite exportar un diagrama en formato del estándar XMI. La exportación se realiza del diagrama completo, incluyendo todos los elementos creados en él. La salida de este será un archivo XML con el formato específico del estándar. | |
| **Reglas u Observaciones:** | |

Tabla . Formato de historia de usuario H21

### Arquitectura de la Aplicación

La arquitectura de la aplicación es una arquitectura basada en capas, cada capa cumple una función específica y dispone de componentes especializados para su correcto funcionamiento. La arquitectura establecida para la aplicación se asemeja a la arquitectura de la plataforma J2EE y de hecho se basa en dicha arquitectura, ya que la mayoría de herramientas usadas para el desarrollo son hechas en lenguaje Java.

La siguiente imagen describe a grandes rasgos la arquitectura usada para la aplicación.



Figura . Arquitectura de la aplicación CLASS Modeler

La imagen anterior se divide en 3 grandes grupos: *Cliente*, *Servidor* y *Datos*; estos representan los niveles de la aplicación *CLASS Modeler*, cada nivel interactúa con los otros mediante interfaces definidas y cada uno tiene responsabilidades propias.

El nivel ***Cliente*** representa la interfaz mediante la cual un usuario interactúa con la aplicación. Básicamente estamos hablando de una Interfaz Gráfica de Usuario (GUI) a la cual un usuario de la aplicación accede y mediante la cual se comunica con los otros componentes. Este componente puede ser personificado por cualquier Navegador Web disponible en el mercado, siempre y cuando posea las características necesarias para ejecutar la aplicación.

El nivel ***Servidor*** representa el Backend de la aplicación, desde y hacia la cual fluyen los datos manejados por la misma. Las tecnologías usadas para el desarrollo de la aplicación requieren que se disponga de un servidor especializado conocido como Servidor de Aplicaciones o Middleware, este tipo de software tiene características especiales más que el solo recibir y procesar peticiones HTTP.

Por último, el nivel ***Datos*** representa el almacenamiento de los datos de la aplicación de manera persistente, este es personificado por un motor de bases de datos del mercado.

A su vez, cada componente se divide en diferentes capas, tal como se mencionó anteriormente, la arquitectura de la aplicación está basada en capas. A continuación se describen cada una de las capas.

* **Capa de Vista**: Esta capa se encuentra en el Cliente, básicamente se encarga de presentar los datos de la aplicación de una manera agradable al usuario, adicionalmente se encarga de tomar las acciones realizadas por el usuario y enviarlas al Servidor, en donde serán procesadas y se generará una respuesta en base a los datos.
* **Capa de Control y Presentación**: Esta capa se encuentra en el Servidor, se encarga de tomar los datos enviados desde la vista, procesarlos y validarlos, de tal manera que se envíen a la capa de negocio donde reposa la lógica necesaria para determinar cuál debe ser la respuesta esperada para determinada acción. Adicionalmente se encarga de tomar la respuesta de la capa de negocio, generar
* **Capa de Lógica de Negocio**: Contiene la lógica de operación de la aplicación, define los procesos y procedimientos realizados por la aplicación según las acciones del usuario.
* **Capa de Integración**: También conocida como la capa de acceso a datos, esta capa se encarga de la comunicación con el repositorio de datos de la aplicación.

En la figura vista anteriormente, adicional a la arquitectura de la aplicación, se mencionan algunas tecnologías empleadas en cada capa, en la siguiente sección se explica con más detalle cada una de estas tecnologías y la función que cumplen dentro de la aplicación.

### Definición de la Tecnología

#### JSF

El objetivo de la tecnología JavaServer Faces es desarrollar aplicaciones web de forma parecida a como se construyen aplicaciones locales con Java Swing, AWT (Abstract Window Toolkit), SWT (Standard Widget Toolkit) o cualquier otra API similar.

Tradicionalmente, las aplicaciones web se han codificado mediante páginas JSP (Java Server Pages) que recibían peticiones a través de formularios y construían como respuesta páginas HTML (Hiper Text Markup Language) mediante ejecución directa o indirecta -a través de bibliotecas de etiquetas- de código Java, lo que permitía, por ejemplo, acceder a bases de datos para obtener los resultados a mostrar amén de realizar operaciones marginales como insertar o modificar registros en tablas relacionales, actualizar un carrito de la compra, etc.

JavaServer Faces pretende facilitar la construcción de estas aplicaciones proporcionando un entorno de trabajo (Framework) vía web que gestiona las acciones producidas por el usuario en su página HTML y las traduce a eventos que son enviados al servidor con el objetivo de regenerar la página original y reflejar los cambios pertinentes provocados por dichas acciones. En definitivas cuentas, se trata de hacer aplicaciones Java en las que el cliente no es una ventana de la clase JFrame o similar, sino una página HTML [15].

La tecnología JavaServer Faces constituye un marco de trabajo (Framework) de interfaces de usuario del lado de servidor para aplicaciones web basadas en tecnología Java y en el patrón MVC (Modelo Vista Controlador).

Los principales componentes de la tecnología JavaServer Faces son una API y una Implementación de referencia para:

* Representar componentes de interfaz de usuario (UI-User Interface) y manejar su estado.
* Manejar eventos, validar en el lado del servidor y convertir datos.
* Definir la navegación entre páginas.
* Soportar internacionalización y accesibilidad, y proporcionar extensibilidad para todas estas características.
* Una librería de etiquetas JavaServer Pages (JSP) personalizadas para dibujar componentes UI dentro de una página JSP.

Este modelo de programación bien definido y la librería de etiquetas para componentes UI facilita de forma significativa la tarea de la construcción y mantenimiento de aplicaciones web con UIs en el lado servidor. Con un mínimo esfuerzo, es posible:

* Conectar eventos generados en el cliente a código de la aplicación en el lado servidor.
* Mapear componentes UI a una página de datos en el lado servidor.
* Construir una interfaz de usuario con componentes reutilizables y extensibles.

JSF comprende dos grandes grupos de etiquetas, el grupo JSF HTML (usan el prefijo H) y el grupo JSF Core (usan el prefijo F). Las etiquetas H representan los componentes básicos HTML, como lo son tablas, párrafos, imágenes, campos de texto, etc. Por ejemplo un campo de texto JSF se representa mediante la etiqueta <h:inputText /> y asi sucesivamente con todos los componentes. Por otro lado las etiquetas F representan por lo general elementos de control, tales como eventos, validadores, interacciones AJAX, etc. Es así como mezclando unas etiquetas con otras se puede desarrollar aplicaciones web robustas.

Otro aspecto importante del Framework JSF son los controladores. Estos controladores son conocidos como JSF Managed Beans, los cuales son un tipo especial de Bean y se caracterizan por tener un ciclo de vida específico dependiendo del ámbito otorgado por el desarrollador. Cada Bean Manejado se caracteriza por estar marcado con la anotación @ManagedBean. Los objetos de estas clases son creados y destruidos de manera automática por el contenedor cuando es necesario. Este proceso depende del ciclo de vida que se le otorgue a determinado controlador y el ciclo de vida determina la duración de un objeto en memoria, este ciclo de vida se conoce como *ámbito (Scope)* del bean. Cada bean de JSF se instancia cuando se solicita una página en la cual este es usado y permanece en memoria hasta que el ámbito lo determine. Los tipos de ámbitos enmarcados por las siguientes anotaciones:

* @Application: Los beans de tipo Application son instanciados una vez se necesiten y permanecen en memoria mientras el contenedor o servidor de aplicaciones siga encendido. Es decir, el objeto permanece y conserva su estado durante toda la ejecución de la aplicación, el contenedor no crea ningún otro objeto de este tipo y usa la misma instancia en cada interacción.
* @Session: Los beans de tipo Session son instanciados una vez se necesitan y permanecen en memoria hasta que la sesión del usuario se termine. Una vez se instancia el objeto este se usa en cada interacción y mantiene su estado siempre y cuando la sesión de usuario siga activa. Es común que los controladores de tipo sesión almacenen información específica del usuario que ingresa a la aplicación de manera autorizada. Una vez la sesión se termina el objeto es destruido y el contenedor vuelve a instanciar un objeto cuando se requiera.
* @View: Los beans de tipo View son instanciados una vez el usuario solicita una página y permanece en memoria mientras el usuario siga en la misma página. Una vez el usuario solicita otra página de la aplicación el bean es destruido, por lo tanto la información almacenada no puede ser accedida. Es común utilizar este tipo de beans como controladores de formularios debido a que se debe mantener el estado mientras el usuario se encuentra visualizando el formulario.
* @Request: Los beans de tipo Request son instanciados una vez se necesitan y permanecen en memoria hasta que se realiza una nueva petición. Esto incluye peticiones síncronas (GET o POST) o asíncronas (AJAX).

#### Facelets

Las interfaces de usuario son típicamente el aspecto más volátil durante el desarrollo de una aplicación web, y estas son a menudo un compuesto de código frágil que es difícil de cambiar, haciendo las interfaces de usuario costosas de implementar. Facelets fue originalmente desarrollado como una alternativa al manejo de vistas basado en JSP en JSF1.x. En JSF 2.0, Facelets reemplaza JSP como la tecnología de vista por defecto de JSF. Aparte de ser un mejor manejador de vistas, Facelets soporta un número de etiquetas para hacer plantillas y otros propósitos. El prefijo usado para las etiquetas de Faceles es UI, por lo tanto todas las etiquetas lucen como <ui:xxxx /> [15].

Las etiquetas de Facelets pueden ser agrupadas en las siguientes categorías:

* Incluir contenido de otras páginas XHTML (ui:include).
* Construir paginas desde plantillas (ui:composition, ui:decorate, ui:insert, ui:define, ui:param).
* Crear componentes a la medida sin escribir código Java (ui:component, ui:fragment).
* Utilidades adicionales (ui:debug, ui:remove, ui:repeat).

La mayoría de aplicaciones web siguen un patrón similar, en el cual todas las páginas tienen una distribución y estilo común. Por ejemplo, es típico para las páginas tener el mismo encabezado, pie de página y barras laterales.

Facelets permite encapsular esos aspectos comunes en una plantilla diseñada por el desarrollador, de tal manera que puede actualizar el look de la aplicación haciendo cambios únicamente en el témplate y no en todas las paginas individuales. Encapsulación es la piedra angular de la programación orientada a objetos y un principio bien conocido llamado DRY[[16]](#footnote-16).

#### PrimeFaces

PrimeFaces es una suite de componentes open source para Java Server Faces, conteniendo un conjunto de más de 70 componentes JSF para el desarrollo de aplicaciones RIA[[17]](#footnote-17) basadas en AJAX, adicionalmente el módulo TouchFaces comprende un kit para el desarrollo de aplicaciones web para dispositivos móviles [16].

Las principales características de PrimeFaces son:

* Conjundo de componentes “ricos” (HtmlEditor, Dialog, AutoComplete, Charts y muchos otros).
* Renderizado ligero y parcial de páginas construido en AJAX.
* Soporte nativo de AJAX Push/Comet.
* Compatible con otras librerías.
* JavaScript discreto.
* Extensa documentación.

Más detalles en la página principal de PrimeFaces: <http://www.primefaces.org/> y en la página de demostración: <http://www.primefaces.org/showcase/ui/home.jsf>, en esta página se encuentra una pre visualización de cada uno de los componentes pre construidos que provee la librería, además ofrece ejemplos junto con el código fuente.

#### EJB

Los Enterprise JavaBeans (EJB) son una de las APIs que forman parte del estándar de construcción de aplicaciones empresariales J2EE. Su especificación detalla como los servidores de aplicaciones proveen objetos del lado del servidor, que son precisamente, los EJB:

* Comunicación remota utilizando CORBA.
* Transacciones
* Control de concurrencia.
* Eventos utilizando JMS (Java Messaging Service).
* Servicios de nombres y de directorio.
* Seguridad
* Ubicación de componentes en un servidor de aplicaciones (JNDI).

Los EJB proporcionan un modelo de componentes distribuido estándar del lado del servidor. El objetivo de los EJB es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (como los mencionados anteriormente), para centrarse en el desarrollo de la lógica de negocio en sí. El hecho de estar basado en componentes permite que estos sean flexibles y sobre todo reutilizables[[18]](#footnote-18).

No hay que confundir los Enterprise JavaBeans con los JavaBeans. Los JavaBeans también son un modelo de componentes creado por Oracle - Sun Microsystems para la construcción de aplicaciones, pero no pueden utilizarse en entornos de objetos distribuidos al no soportar nativamente la invocación remota RMI.

Existen tres tipos de EJBs:

1. **EJB de Entidad (Entity Beans)**: Su objetivo es encapsular los objetos del lado del servidor que almacenan los datos. Los EJB de entidad presentan la característica fundamental de la persistencia. Sin embargo, en la documentación de Java para JEE 5.0, los Entity Beans desaparecen ya que son reemplazados por JPA (Java Persistence API). Dentro de esta categoría se encuentran dos tipos de persistencia: Persistencia Gestionada por el Contenedor (CMP), en donde el contenedor se encarga de almacenar y recuperar los datos del objeto de entidad mediante el mapeo o vinculación de las columnas de una tabla de la base de datos con los atributos del objeto; el otro tio se llama Persistencia Gestionada por el Bean (BMP) en donde el propio vean de entidad se encarga, mediante una base de datos u otro mecanismo, de almacenar y recuperar los datos a los que se refiere, por lo cual la responsabilidad de implementar los mecanismos de persistencia es del programador.
2. **EJB de Sesión (Session Beans)**: Gestionan el flujo de la información en el servidor. Generalente sirven a los clientes como una fachada de los servicios proporcionados por otros componentes disponibles en el servidor. Puede haber dos tipos de Session Beans: Stateful y Stateless, estos ya fueron mencionados anteriormente en la sección 8.2.4.
3. **EJB dirigidos por mensajes (Message-Driven EJBs)**: Son los únicos Beans con funcionamiento asíncrono. Usando el Java Messaging System (JMS), se suscriben a un tema (topic) o u una cola (queue) y se activan para recibir un mensaje dirigido a dicho tema o cola. No requieren de su instanciación por parte del cliente.

#### Java Persistence API

JPA es un Framework ligero basado en POJOs[[19]](#footnote-19) para persistencia en Java. Aunque el mapeo objeto-relacional es un componente primordial del API, también ofrece solución a retos arquitecturales de integrar persistencia en aplicaciones empresariales escalables [17].

JPA consiste en cuatro áreas:

* El API de persistencia de Java.
* El lenguaje de consultas (JPQL).
* El API Java Persistence Criteria
* Metadatos para mapeo Objeto/Relacional.

A continuación se explican algunos conceptos básicos de este Framework:

**Entity**

Una Entity es un objeto liviano del dominio de persistencia. Típicamente, una entity representa una tabla en una base de datos relacional, y cada instancia de una entity corresponde con una fila en esa tabla. El artefacto primario de programación de una entity es una clase entity, aunque estas también pueden usar clases auxiliares.

Una clase Entity debe estar marcada por la *Anotación[[20]](#footnote-20)* @Entity de tal manera que puede ser catalogada como tal.

**Campos y Propiedades persistentes en una clase Entity**

El estado persistente de una Entity puede ser accedido mediante las variables de instancia de la entity o las propiedades. Los campos o propiedades deben ser de algún tipo de dato del lenguaje Java: int, long, double, String, etc.

Si una Entity usa propiedades persistentes, la Entity debe seguir las convenciones para métodos de los componentes JavaBeans. Las propiedades al estilo JavaBeans usan métodos get y set que permiten el acceso a los valores de la entidad y proveen el mecanismo de encapsulación. Cada propiedad de una Entity debe tener sus métodos get y set.

#### MxGraph

La librería mxGraph contiene un software de cliente, escrito en JavaScript, y una serie de Backends para varios idiomas. El software de cliente es un componente gráfico con una envoltura de aplicación opcional que está integrado en una interfaz web existente. El cliente requiere un servidor web para entregar los archivos necesarios en el cliente o puede ejecutarse desde el sistema de archivos local, sin un servidor web. Los motores que se utilizan tal cual, o pueden estar incrustados en una aplicación de servidor existente en uno de los idiomas compatibles.



Figura . Arquitectura de la libreria mxGraph

Si existe un backend, a continuación, el cliente puede ser configurado para utilizar este backend en diversas formas, tales como:

* La creación de imágenes
* Almacenar y cargar diagramas
* Creación de una representación de objeto de un gráfico
* Compartir diagramas con otros clientes

Los escenarios anteriores pueden ser combinados de varias maneras, tales como el envío de una descripción XML de cada cambio en el backend como sucede, o guardado automático del diagrama para evitar la pérdida de datos en el cliente. El cliente también puede operar en el modo fuera de línea, en el que no requiere un motor o de un servidor web.

#### Glassfish

GlassFish es un servidor de aplicaciones de software libre desarrollado por Sun Microsystems, compañía adquirida por Oracle Corporation, que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación. Es gratuito, de código libre y se distribuye bajo un licenciamiento dual a través de la licencia CDDL y la GNU GPL. La versión comercial es denominada Oracle GlassFish Enterprise Server (antes Sun GlassFish Enterprise Server).

GlassFish está basado en el código fuente donado por Sun y Oracle Corporation; este último proporcionó el módulo de persistencia TopLink. GlassFish tiene como base al servidor Sun Java System Application Server de Oracle Corporation, un derivado de Apache Tomcat, y que usa un componente adicional llamado Grizzly que usa Java NIO para escalabilidad y velocidad.

#### MySQL

MySQL, el sistema manejador de bases de datos SQL open source más popular, es desarrollado, distribuido y soportado por la corporación Oracle. MySQL es un sistema manejador de bases de datos. Para agregar, acceder y procesar datos almacenados en un computador de bases de datos, se necesita un DBMS tal como MySQL Server. Una base de datos relacional almacena datos en tablas separadas en vez de meter todos los datos en una gran sala de almacenamiento. Esto añade velocidad y flexibilidad.

MySQL soporta el lenguaje estructurado de consultas SQL, este es el lenguaje estandarizado más común usado para acceder bases de datos y es definido por el estándar ANSI/ISO SQL.

Al ser una herramienta open source permite que cualquiera pueda usar y modificar el software. Cualquier persona puede descargar MySQL desde internet y usarlo sin pagar nada. En caso de así desearlo también se puede estudiar el código fuente y cambiarlo al gusto o necesidad de cada quien. MySQL usa licencias públicas GPL, para definir lo que el usuario puede o no hacer con el software en diferentes situaciones.

Este sistema DBMS fue el seleccionado para servir de almacenamiento persistente de la aplicación CLASS Modeler, debido a que sus características se acoplan completamente a las necesidades de la aplicación y adicionando que es un software liviano y muy potente para tal labor.

#### Eclipse

Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus.

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, cubriendo casi todas las áreas de Model Driven Engineering.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse fue liberado originalmente bajo la Common Public License, pero después fue re-licenciado bajo la Eclipse Public License. La Free Software Foundation ha dicho que ambas licencias son licencias de software libre, pero son incompatibles con Licencia pública general de GNU (GNU GPL).

Esta herramienta fue seleccionada como ambiente para el desarrollo de la aplicación CLASS Modeler, basado en sus características y servicios esta herramienta permite el desarrollo cómodo de aplicación de todo tipo. Otro concepto importante para la selección fue la familiaridad que tiene el Equipo de Desarrollo del proyecto con la herramienta.

## PLANIFICACIÓN

### Priorización de Historias de Usuario

Una vez determinadas las historias de usuario es necesario proceder a realizar una priorización y establecer cuáles deben ser implementadas en iteraciones tempranas y cuales se pueden postergar hasta el final del proyecto. Así como determinar la interdependencia que pueda existir entre estas y poder definir el orden de implementación.

La prioridad se asigna en conjunto con el cliente y se determina en base al impacto que la historia de usuario tiene sobre los objetivos del proyecto. De esta manera se establecen la siguiente clasificación de prioridades:

* **ALTA**: Significa que la historia de usuario tiene un impacto significativo en los objetivos del proyecto y debe implementarse para poder alcanzar los mismos.
* **MEDIA**: Significa que la historia de usuario tiene un impacto moderado sobre los objetivos del proyecto.
* **BAJA**: Significa que la historia de usuario es trivial y no impacta directamente los objetivos del proyecto, se puede describir como un *Nice to Have* (bueno tener) el cual puede aumentar la satisfacción del cliente.

Adicional a la prioridad asignada a cada historia de usuario, se ha considerado la interdependencia que existe entre ellas, el cual es un concepto importante al determinar el orden de implementación y la precedencia.

|  |  |  |
| --- | --- | --- |
| **HISTORIA DE USUARIO** | **PRIORIDAD** | **DEPENDENCIA** |
| H1. Crear cuenta de usuario | MEDIA |  |
| H2. Activar cuenta de usuario | MEDIA | H1 |
| H3. Desactivar cuenta de usuario | BAJA | H6 |
| H4. Recuperar cuenta de usuario | BAJA | H3 |
| H5. Reasignar contraseña | BAJA | H1, H2 |
| H6. Iniciar sesión | MEDIA | H1, H2 |
| H7. Modificar perfil de usuario | BAJA | H6 |
| H8. Cambiar contraseña | BAJA | H6 |
| H9. Crear diagrama | MEDIA | H6 |
| H10. Compartir diagrama | ALTA | H9 |
| H11. Gestionar privilegios de diagrama | ALTA | H9 |
| H12. Editar diagrama | MEDIA | H9 |
| H13. Eliminar diagrama | MEDIA | H9 |
| H14. Copiar diagrama | BAJA | H9 |
| H15. Diseñar diagrama | ALTA | H9 |
| H16. Diseñar diagrama concurrentemente | ALTA | H15 |
| H17. Ver demostración | BAJA | H15 |
| H18. Generar código fuente del diagrama | ALTA | H9 |
| H19. Generar imagen del diagrama | ALTA | H9 |
| H20. Importar XMI | BAJA | H9 |
| H22. Exportar XMI | BAJA | H9 |

Tabla . Priorización de historias de usuario

### Estimación de Historias de Usuario

La estimación del esfuerzo en base al tiempo que emplea el llevar a cabo cada historia de usuario. Esta tarea es realizada por el equipo de desarrollo considerando la complejidad que implica la implementación.

|  |  |
| --- | --- |
| **HISTORIA DE USUARIO** | **ESTIMACIÓN** |
| H1. Crear cuenta de usuario | 1 Semana |
| H2. Activar cuenta de usuario | 1 Semana |
| H3. Desactivar cuenta de usuario | 1 Semana |
| H4. Recuperar cuenta de usuario | 1 Semana |
| H5. Reasignar contraseña | 1 Semana |
| H6. Iniciar sesión | 2 Semanas |
| H7. Modificar perfil de usuario | 1 Semana |
| H8. Cambiar contraseña | 1 Semana |
| H9. Crear diagrama | 1 Semana |
| H10. Compartir diagrama | 2 Semanas |
| H11. Gestionar privilegios de diagrama | 2 Semanas |
| H12. Editar diagrama | 1 Semana |
| H13. Eliminar diagrama | 1 Semana |
| H14. Copiar diagrama | 1 Semana |
| H15. Diseñar diagrama | 3 Semanas |
| H16. Diseñar diagrama concurrentemente | 2 Semanas |
| H17. Ver demostración | 1 Semana |
| H18. Generar código fuente del diagrama | 2 Semanas |
| H19. Generar imagen del diagrama | 2 Semanas |
| H20. Importar XMI | 2 Semanas |
| H22. Exportar XMI | 2 Semanas |

Tabla . Estimación de tiempo para historias de usuario

### Definición de Iteraciones

Basado en la priorización y estimación de las historias de usuarios, se definen las iteraciones del proyecto, determinando la cantidad de iteraciones (entregables del proyecto) necesarias para la completa ejecución del mismo y el cumplimiento de los objetivos, definiendo las historias de usuario cubiertas por cada una de ellas.

Basado el tamaño del proyecto se determinó que las iteraciones deben ser máximo de 3 semanas de duración, por lo tanto se trataran establecen la iteraciones basado en la estimación de tiempo realizada por cada historia de usuario. Otro factor adicional para la definición de las iteraciones es la dependencia que existe entre las historias de usuario, considerando que para poder implementar algunas de ellas es necesario que otras funcionalidades ya estén disponibles en la aplicación.

La siguiente tabla muestra el plan de iteraciones y la distribución de las historias de usuario en cada una de ellas.

|  |  |
| --- | --- |
| **ITERACIÓN** | **HISTORIAS DE USUARIO** |
| Iteración 1 | H1, H2, H5 |
| Iteración 2 | H6, H9 |
| Iteración 3 | H15 |
| Iteración 4 | H10, H12 |
| Iteración 5 | H16, H17 |
| Iteración 6 | H18, H19 |
| Iteración 7 | H11, H13, H14 |
| Iteración 8 | H7, H8 |
| Iteración 9 | H3, H4 |
| Iteración 10 | H20, H21 |

Tabla . Plan de iteraciones

### Metáfora del Sistema

Aplicación web basada en ambiente Cloud que asemeja una herramienta CASE desktop para la creación, edición y manejo de Diagramas de Clase UML. Permite a los usuarios diseñar de manera fácil diagramas de clase UML a través de una interfaz web, así como compartir sus diagramas con otros usuarios y editar concurrentemente los diagramas compartidos. Se permite agregar componentes desde una paleta al lienzo principal del diseñador, y se dispone de los elementos básicos para el diseño de diagramas de clase. Al final de la edición la aplicación permite obtener el código fuente que representa el diagrama dibujado.

## IMPLEMENTACIÓN

En primera instancia se describen los patrones de desarrollo que se utilizaron en la codificación de la herramienta, a continuación se menciona cada iteración del proyecto describiendo las historias de usuario que estas abarcan. Para cada iteración se describe los aspectos importantes que permiten su completa ejecución y los artefactos desarrollados, tales como interfaces gráficas, clases del dominio, servicios, persistencia de datos, controles y pruebas.

### Patrones de Desarrollo

#### Modelo Vista Controlador

Uno de los Frameworks más conocidos y ampliamente usado para el diseño de GUI es el Modelo-Vista-Controlador (MVC). El Framework MVC fue propuesto originalmente en la década de los 80 como una aproximación al diseño de GUI que permitió múltiples presentaciones de un objeto y estilos independientes de interacción en cada una de las presentaciones. El marco MVC soporta la presentación de los datos de diferentes formas e interacciones independientes con cada una de estas presentaciones. Cuando los datos se modifican a través de una de las presentaciones, el resultado de las presentaciones son actualizadas [2].

Este es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el modulo encargado de gestionar los eventos y comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador.

* El Modelo: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación. Envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada.
* El controlador: Responde a eventos (usualmente acciones de usuario) e invoca peticiones al modelo cuando se hace alguna solicitud sobre la información. También puede enviar comandos a su vista asociada si se solicita un cambio en la forma en que se presenta de modelo. Se puede decir que el controlador hace de intermediario entre la vista y el modelo.
* La vista: Presenta el modelo (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario).

La siguiente imagen ilustra este patrón.



Figura . Patrón Modelo-Vista-Controlador

El Framework JSF de Java se basa específicamente en este patrón, siguiendo a cabalidad y forzando al desarrollador a programar en base a este concepto. De modo que las aplicaciones desarrolladas forzosamente aplican el patrón MVC.

#### Intercepting Filter

Cuando una petición ingresa a la aplicación Web, a menudo esta debe pasar varias pruebas de ingreso antes de llegar a la fase principal de procesamiento. Por ejemplo,

* ¿El cliente ha sido autenticado?
* ¿El cliente tiene una sesión valida?
* ¿La IP del cliente procede de una red verdadera o valida?
* ¿La ruta de la petición web viola alguna restricción?
* ¿Qué codificación usa el cliente para enviar los datos?
* ¿La aplicación soporta el tipo de navegador del cliente?

Algunos de estos ítems chequeados son pruebas, resultantes en una respuesta SI o NO que determina si el procesamiento continuará. Otros chequeos manipulan los datos de flujo de entrada en un formulario susceptible de ser procesado.

La solución clásica consiste en una serie de chequeos condicionales, con cualquier chequeo fallido la petición se aborta. Sentencias if/else anidadas son la estrategia estándar, pero esta solución conduce a código frágil y al estilo de programación de copiar-y-pegar, debido a que el flujo de filtrado y la acción de los filtros es compilada dentro de la aplicación [18].

La llave para resolver este problema en un manera flexible y discreta es tener un mecanismo simple para adicionar y remover componentes de procesamiento, en los cuales cada componente complete una acción específica de filtrado.

La solución es crear filtros conectables para procesar servicios comunes en una manera estándar sin requerir cambios en el código principal de procesamiento de la petición. Estos filtros intersectan las peticiones entrantes y las respuestas salientes, permitiendo pre y post procesamiento. Seriamos capaces de adicionar y remover estos filtros sin interferir los unos con los otros, sin requerir cambios en el código existente.

En java, este patrón se implementa bajo la clase Filter, la cual permite realizar el filtrado de las peticiones basada en la URL de la petición, si esta coincide con el patrón asignado al filtro entonces dicha petición es procesada por el filtro en cuestión.

Dentro de la aplicación CLASS Modeler este patrón se aplica para verificar o forzar el inicio de sesión por parte de los Diagramadores, y para brindar un grado de seguridad confiable, ya que existen páginas a las cuales solo se puede acceder si se ha autenticado dentro de la aplicación mediante el formulario de inicio de sesión. Por lo tanto si una persona solicita una página específica y no se ha autenticado, la aplicación le responde con un mensaje apropiado o lo redirección a la página principal. Más adelante en la sección de seguridad se explica este tema con mayor detalle.

#### Front Controller

El sistema requiere un punto de acceso centralizado para manipular la petición de la capa de presentación para apoyar la integración de los servicios del sistema, recuperación de contenido, gestión de la vista y navegación. Cuando el usuario accede a la vista directamente sin ir a través de un mecanismo centralizado, se pueden presentar dos problemas:

* Se requiere Cada fin de proporcionar a sus propios servicios del sistema, a menudo dando por resultado duplicar el código.
* Ver navegación se deja a los puntos de vista. Esto puede resultar en vista mezclados contenido y la vista de navegación.

Además, el control distribuido es más difícil de mantener, ya que los cambios se a menudo tienen que hacer en muchos lugares.

El controlador proporciona un punto de entrada centralizado que controla y gestiona el tratamiento de la petición Web. Por puntos y controles de decisiones centralizadas, el controlador también ayuda a reducir la cantidad de código Java, llamado scriptlets, incrustado en la JSP.

Centralizar el control en el controlador y la reducción de la lógica de negocio en la vista promueve la reutilización de código en todas las solicitudes. Es un enfoque preferible a la código alternativo-incrustación en múltiples puntos de vista, porque este enfoque puede conducir a un ambiente reutilización por copiar y pegar más propenso a errores.

Típicamente, un controlador coordina con un componente despachador. Los despachadores son responsables de control de la vista y la navegación. Por lo tanto, un despachador elige la próxima vista para el usuario y el control de vectores al recurso. Los despachadores pueden estar encapsulados dentro del controlador directamente o se puede extraer en una separada componente [18].

Este patrón es típicamente implementado en JSF, ya que existe un control central el cual es un Servlet (Java Server Faces Servlet), al cual llegan todas las peticiones de la aplicación y este se encarga de re direccionarlas al controlador especifico.

#### Session Facade

En un ambiente de aplicación J2EE multicapa, algunos problemas comunes pueden surgir:

* Fuerte acoplamiento, que produce dependencia directa entre objetos del cliente y de negocio.
* Mucha invocación de métodos entre el cliente y el servidor, produciendo problemas de rendimiento de red.
* Falta de una estrategia de acceso uniforme al cliente, exponiendo objetos de negocio que no serán usados.

Una aplicación J2EE multicapa tiene numerosos objetos del lado del Servidor que son implementados como Enterprise Beans. Adicionalmente, algunos otros objetos arbitrarios pueden proveer servicios, datos, o ambos. Estos objetos son conjuntamente llamados objetos de negocio, desde que estos encapsulan datos y lógica de negocio.

Las aplicaciones J2EE implementan objetos de negocio que proveen procesar servicios como Beans de sesión. Objetos de negocio genéricos que representan un objeto vista del almacenamiento persistente y son compartidos por múltiples usuarios son usualmente implementados como Beans de entidad.

Este patrón provee:

* Una interfaz simple para el cliente que esconde toda la complejidad de interacción entre componentes de negocio.
* Reduce el número de objetos de negocio que son expuestos al cliente a lo largo de la capa de servicios sobre la red.
* Esconde del cliente las interacciones subyacentes y las interdependencias entre componentes de negocio. Este provee una mejor manejabilidad, centralización e interacción (responsabilidad), mayor flexibilidad y mayor habilidad para hacer frente a los cambios.
* Provee una capa de servicios uniforme y genérica para separar la implementación del objeto de negocio de la abstracción del servicio de negocio.
* Evita exponer los objetos de negocio subyacentes directamente al cliente para mantener el acoplamiento entre dos capas al mínimo.

Este patrón tiene varias estrategias de implementación, a continuación se mencionan cada una de las estrategias de implementación:

* Stateless Session Facade: Un proceso de negocio que solo necesita una invocación a un método para completar el servicio es un proceso de negocio no conversacional. Este tipo de proceso son comúnmente implementados como un Session Bean Stateless, debido a que no se necesita almacenar información temporal.
* Stateful Session Facade: Un proceso de negocio que necesita múltiples invocaciones a métodos para completar el servicio es un proceso de negocio conversacional. El estado conversacional debe ser almacenado entre cada invocación del método del cliente. En este escenario, un Session Bean Stateful puede ser un enfoque más adecuado para implementar el patrón Session Facade.

La implementación de Java de este patrón se llama EJB (Enterprise Java Beans). En cuyo caso también implementa Stateless y Stateful Session Beans, sin embargo Java agrega un tipo adicional de objeto que contiene características similares pero provee una funcionalidad adicional integrando el patrón Singleton, el cual de hecho lleva el mismo nombre de Session Bean. En la siguiente sección se explica un poco más de este tema [17].

#### Singleton

El patrón de diseño Singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

El patrón Singleton se implementa creando en nuestra clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con atributos como protegido o privado).

La instrumentación del patrón puede ser delicada en programas con múltiples hilos de ejecución. Si dos hilos de ejecución intentan crear la instancia al mismo tiempo y esta no existe todavía, sólo uno de ellos debe lograr crear el objeto. La solución clásica para este problema es utilizar exclusión mutua en el método de creación de la clase que implementa el patrón.

Las situaciones más habituales de aplicación de este patrón son aquellas en las que dicha clase controla el acceso a un recurso físico único (como puede ser el ratón o un archivo abierto en modo exclusivo) o cuando cierto tipo de datos debe estar disponible para todos los demás objetos de la aplicación.

El patrón Singleton provee una única instancia global gracias a que:

* La propia clase es responsable de crear la única instancia.
* Permite el acceso global a dicha instancia mediante un método de clase.

Declara el constructor de clase como privado para que no sea instanciable directamente.

#### Data Access Object

Muchas aplicaciones J2EE del mundo real necesitan usar datos persistentes en algún punto. Para muchas aplicaciones, el almacenamiento persistente es implementado con diferentes mecanismos, y hay muchas diferencias marcadas en las APIs usadas para acceder estos diferentes mecanismos de almacenamiento persistente. Otras aplicaciones pueden necesitar acceder datos que residen en sistemas separados. Por ejemplo, los datos pueden residir en sistemas MainFrame, repositorios Lightweight Directory Access Protocol (LDAP) y algunos otros. Otro ejemplo es donde los datos son proporcionados por servicios a través de sistemas externos tales como sistemas de integración B2B, servicios de tarjetas de crédito y así sucesivamente.

Tipicamente, las aplicaciones usan componentes distribuidos y compartidos tales como Entity Beans para representar datos persistentes. Una aplicación es considerada para emplear un Bean manejado de persistencia (BMP) para sus Entity Beans cuando estos accedan explícitamente el almacenamiento persistente. Una aplicación con requerimientos simples puede prescindir del uso de Entity Bean y en vez de ellos usar Session Beans o Servlets para acceder directamente al almacenamiento persistente para obtener o modificar datos. O, la aplicación puede usar Entity Beans con persistencia auto manejada por el contenedor (Servidor de Aplicaciones usualmente), y entonces dejar al contenedor manejar la transaccionalidad y los detalles de persistencia [18].

En este proyecto este concepto es ampliamente utilizado, ya que la persistencia se realiza utilizando este patrón, la transaccionalidad es manejada automáticamente por el contenedor y se utilizan Entity Beans que representan los objetos de negocio.

El patrón DAO implementa el mecanismo de acceso requerido para trabajar con la fuente de datos (Data Source). La fuente de datos puede ser un almacenamiento persistente como un DBMS, un servicio externo como intercambios B2B, un repositorio como una base de datos LDAP o un servicio de negocio accedido via CORBA o Sockets de bajo nivel.

### Estructura del Código del Proyecto

Como se mencionó anteriormente el ambiente de desarrollo utilizado fue el IDE Eclipse, esta aplicación permite la configuración y desarrollo de todos los componentes que utiliza la aplicación, tanto el código fuente en Java como el desarrollo de los componentes Web del cliente, paginas HTML, rutinas de JavaScript y hojas de estilo en cascada CSS.

Dentro del IDE se creó un Proyecto de Aplicación Java Enterprise, el cual engloba y define el contexto de la aplicación en general, dentro de este proyecto existen algunos archivos de configuración que permiten definir diferentes aspectos de la aplicación, tal como el nombre, la descripción y los módulos que harán parte de ella. Una aplicación Enterprise se puede componer de varios módulos y estos módulos pueden ser de diferentes tipos. Por ejemplo, un módulo de la aplicación puede ser un Proyecto EJB que engloba componentes y servicios EJB, otro modulo puede ser una aplicación Web que publica una interfaz de usuario para acceder a los servicios de la aplicación y un último modulo puede ser una aplicación Desktop hecha en Swing o JavaFX la cual accede a los servicios prestados por el proyecto EJB.

La siguiente imagen describe la estructuración del código del proyecto y sus módulos.



Figura . Estructura y distribución del código de la aplicación CLASS Modeler

CLASS Modeler se compone de 3 módulos base, un módulo EJB, un módulo Web y un módulo de Tests; cada uno de ellos aloja los componentes creados para cada capa de la arquitectura del proyecto.

De esta manera, las capas de Vista, Presentación y Control se encuentran alojadas en el proyecto *CLASSModelerWeb*, y las capas de Lógica de Negocio e Integración están alojadas en el proyecto *CLASSModelerEJB*.

El proyecto *CLASSModelerTest* aloja las clases creadas como pruebas unitarias para los servicios desplegados por el proyecto EJB, estas clases fueron aisladas en un proyecto propio debido a que estas no hacen parte de los componentes necesarios para ejecutar de la aplicación y no deben ser incluidos en el código binario a desplegar en el servidor. Los archivos binarios resultado de compilar y empaquetar el proyecto se generan desde el proyecto principal *CLASSModeler****,*** estos van auto contenidos en un archivo .EAR que se crea mediante el IDE.

### Interfaz Gráfica de Usuario

Al tratarse de una aplicación web es común que la interfaz gráfica de usuario se diseñe en lenguaje HTML, sin embargo, y debido al uso de algunos Frameworks como JSF, Facelets y PrimeFaces la interfaz de usuario se realizó de una manera un tanto diferente.

Básicamente las páginas de la aplicación se crearon en base a plantillas, estas plantillas son diseñadas mediante la librería Facelets, la cual a su vez es parte del Framework JSF 2.0. Estas plantillas permiten que todas las páginas de la aplicación sigan un mismo patrón de diseño sin la necesidad de reescribir todas las secciones de la página cada vez que se crea una nueva, esto reduce la cantidad de código duplicado, posibilita la reutilización y ayuda a que el mantenimiento de la aplicación sea mucho más fácil.

Facelets permite la inserción de código HTML de una página a otra, definiendo secciones para tal propósito, esto significa que cada página contiene el código especifico correspondiente a su contenido y el contenido de las demás secciones simplemente se “heredan” de la plantilla base. Para la aplicación se crearon en total 2 plantillas: common.xhtml y portal.xhtml.



Figura . Plantillas principales para la interfaz gráfica de la aplicación

Estas dos plantillas son la base para todas las paginas HTML de la aplicación, de igual manera es posible que una plantilla se pueda crear en base a otra; en este caso la plantilla *portal.xhtml* está basada en la plantilla *common.xhtml* y a su vez sirve como plantilla de todas las demás páginas de la aplicación, a excepción de la página creada para el Diseñador, la cual está basada directamente desde *common.xhtml*.

A continuación veremos la distribución de algunas otras páginas y algunas imágenes tomadas de la aplicación real.



Figura . Jerarquía de las páginas basadas en plantillas

¿Cómo funciona el proceso?, como ya se mencionó anteriormente, las páginas web no son programadas en lenguaje HTML, en su lugar se usan las etiquetas especiales del Framework Facelets. Estas etiquetas son procesadas por el controlador JSF y son convertidas a su equivalente en código HTML nativo al momento de visualizar una página, esto se hace mediante un mecanismo de Renderizacion que permite transformar las etiquetas especiales e insertar el código proveniente de las plantillas base de la página que está siendo procesada.

Adicionalmente a este mecanismo basado en plantillas, la personalización y embellecimiento de las páginas está a cargo del lenguaje CSS, mediante hojas de estilo en cascada se personaliza las vistas, agregando tipos de fuente, tamaño y color de letras y entre otras.

Los elementos que normalmente se usan en formularios web tales como Cajas de Texto, Combos, Tablas y otros componentes son personalizado mediante el uso del Framework PrimeFaces, el cual como ya se mencionó anteriormente es una librería de componentes gráficos que permiten diseñar páginas web basadas en JSF de manera fácil.

### Iteración 1

Las historias de usuario abarcadas por esta iteración son H1, H2 y H5. Estas historias de usuario comprenden el inicio del proyecto y permiten, además de las características propias, la correcta implementación de las historias de usuario de iteraciones subsiguientes, por lo tanto el análisis realizado para esta iteración es de vital importancia y debe considerar aspectos que si bien no serán implementados aún deben engranarse con las funcionalidades posteriores. A continuación se describen las actividades a realizar, el diseño de los componentes necesarios para cumplir con cada historia de usuario y las pruebas realizadas para verificar cada funcionalidad.

#### Actividades a Realizar

Dentro del desarrollo de esta iteración es necesario realizar una serie de actividades que permiten la ejecución y el cumplimiento de los objetivos para la misma; todas estas actividades están enfocadas finalmente en la implementación de las historias de usuario para esta iteración, sin embargo y como se mencionó anteriormente también servirán de base para posteriores iteraciones.

* Investigar sobre patrones de diseño J2EE para el desarrollo de aplicaciones en Java.
* Investigar las APIs de Java para envío de correos electrónicos.
* Investigar las APIs de Java para encriptación de textos.
* Modelar la estructura de clases necesarias para implementar cada historia de usuario, considerando lógica del negocio y relaciones entre los componentes.
* Modelar los datos que deben ser persistentes y diseñar la estructura de base de datos.
* Diseñar la interfaz gráfica principal de la aplicación.
* Diseñar el formulario de creación de usuarios.
* Implementar un mecanismo para enviar correos electrónicos desde la aplicación.
* Implementar un mecanismo para encriptar textos desde la aplicación.
* Implementar el mecanismo de guardado para nuevos usuarios.
* Verificar los resultados mediante los casos de prueba.

#### Diagrama de Clases

En esta iteración tenemos algunos conceptos importantes que descartar, y en base a los cuales se ha construido el siguiente diagrama de clases. Algunos de estos conceptos son: Cuenta de Usuario, Diagramador, Invitado, Código de Seguridad, Email.

El siguiente diagrama de clase incluye las clases del dominio para esta iteración, básicamente las clases se distribuyen en 2 paquetes, *classmodeler.domain.user* y *classmodeler.domain.security*:



Diagrama . Diagrama de clases del dominio de la iteración 1

Como se muestra en el diagrama, un diagramador (*Diagrammer*) es un usuario de la aplicación, el cual puede acceder mediante registro previo y de manera autorizada; sin embargo, considerando los actores de la aplicación y la historia de usuario H17, es posible ingresar en modo invitado (*Guest*) y acceder a las funcionalidades del software. Es por ello que se realizó una abstracción del concepto usuario y se definió la interfaz *User*, de tal manera que esta encapsulara el comportamiento de cualquier usuario de la aplicación sin importar si fuese registrado o no. Esto será útil en la iteración 2.

Las clases *Diagrammer* y *SecurityCode* representan entidades del dominio que son mapeadas mediante anotaciones Java con las tablas de base de datos. Algunos tipos que representan grupos de conceptos finitos fueron modelados como enumeraciones, por ejemplo, el estado de la cuenta del diagramador (*EDiagrammerAccountStatus*) o el tipo de código de seguridad (*ESecurityCode*). Los números entre paréntesis representan los valores almacenados en base de datos para el literal del enumerado.

La enumeración EDiagrammerAccountStatus representa los diferentes estados de una cuenta de diagramador, los cuales incluyen:

* INACTIVATED (0): Este es el estado que adquiere la cuenta de diagramador una vez se es creada por medio del formulario.
* ACTIVATED (1): Este estado se otorga cuando el diagramador activa su cuenta mediante la historia de usuario H2.
* DEACTIVATED (2): Representa que la cuenta de diagramador ha sido desactivada y no se puede ingresar al sistema a través de ella.

La enumeración ESecurityCode define los tipos de código de seguridad que se generan para las acciones del diagramador:

* ACTIVATE\_ACCOUNT (0): El tipo de código de seguridad generado para activar la cuenta de diagramador.
* RESET\_PASSWORD (1): El tipo de código de seguridad generado para solicitar una reasignación de contraseña de la cuenta de diagramador.

Por convención a las enumeraciones se les colocó la letra E al inicio del nombre.

#### Modelo de Datos

El modelo de datos comprende el almacenamiento de la información persistente de esta iteración, básicamente se necesitan algunos artefactos que permiten realizar esta labor, a continuación se muestra el modelo de datos de esta iteración.

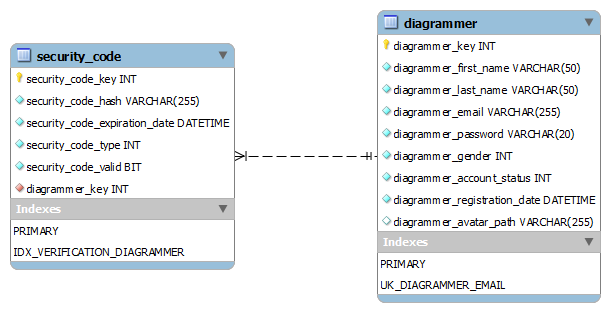


Diagrama . Modelo de datos para la iteración 1

La tabla *diagrammer* almacena la información básica y necesaria de una persona que se registra dentro de la aplicación, esta tabla se encuentra indexada por una llave primaria auto numérico. El email del diagramador está controlado mediante una llave única que restringe la posibilidad de ingresar dos veces la misma dirección para dos diagramadores distintos.

La tabla *security\_code* almacena la información del código de verificación, utilizado para confirmar la dirección de correo del diagramador, así como para reasignar la contraseña de la cuenta. Este incluye el código hash, la fecha de expiración y el tipo de verificación, esta tabla almacenará varios tipos de verificación para diferentes opciones de la aplicación.

La persistencia de los datos se realiza mediante el uso del Framework JPA, este permite interactuar con una abstracción de la base de datos representada por clases entidad (Entity Beans). Hablando específicamente de las tablas mencionadas anteriormente, estas son mapeadas y asociadas directamente con las entidades Diagrammer y SecurityCode (ver diagrama de clases anterior), de esta manera se pueden insertar, actualizar, eliminar u obtener registros de la base de datos como si se tratasen de objetos. Este mapeo se realiza mediante anotaciones agregadas al código de las clases, cada clase que contenga la anotación @Entity es considerada por el Framework y asociada con una tabla de la base de datos. Las filas o registros de la tabla son convertidos a objetos de la clase entidad, permitiendo manipular la base de datos bajo una simulación del paradigma orientado a objetos. De esta forma se encapsula completamente el acceso a los datos y permite al programador aislarse de aspectos como sentencias SQL y demás consideraciones de las bases de datos relacionales.

La conexión a la fuente de datos se realizó mediante la configuración de un recurso JDBC, el cual conecta a la base de datos relacional. El recurso JDBC es provisto por el servidor Glassfish, quien se encarga de mantener las conexiones entrantes a la base de datos, reutilizar las conexiones usadas y liberar el recurso una vez ya no se necesite, este mecanismo es llamado *Pool de Conexiones[[21]](#footnote-21)*. Dicho pool de conexiones se configura de manera manual mediante la consola de administración del servidor de aplicaciones (Ver documento *CLASS Modeler – Installation Guide.pdf*) y se identifica mediante un nombre (para nuestro caso se llama *jdbc/CLASSModeler*). Este nombre es el identificador mediante el cual el Framework JPA accede al recurso publicado por el servidor Glassfish y establece la conexiona a la base de datos. Este recurso JDBC puede ser accedido únicamente por las aplicaciones que se encuentren alojadas dentro del servidor, es decir, ninguna aplicación externa podrá acceder a los datos, lo cual significa cierto grado de seguridad.

Para establecer dicha conexión el Framework JPA dispone de un archivo de configuración, llamado *persistence.xml*, este archivo permite configurar la fuente de datos y algunas otras opciones para el funcionamiento del Framework. A continuación se muestra el contenido del archivo utilizado para la aplicación.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<persistence version=*"2.0"*

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\_2\_0.xsd"*>

<persistence-unit name=*"CLASSModelerPU"*>

<jta-data-source>jdbc/CLASSModelerDS</jta-data-source>

<shared-cache-mode>NONE</shared-cache-mode>

<properties>

<property name=*"eclipselink.logging.level"* value=*"FINE"*/>

</properties>

</persistence-unit>

</persistence>

Mediante el uso de este archivo, el Framework es capaz de buscar la fuente de datos a la cual se va a conectar, es por ello que el nombre del recurso provisto por Glassfish debe ser el mismo que el consignado en el archivo. Al momento de desplegar la aplicación, se inicializa y configura el Framework JPA, probando la conexión al recurso definido. Los datos básicos de conexión, tales como URL, nombre de la base de datos, usuario y contraseña se encuentran en el recurso JDBC y no en el archivo XML.

Algo importante para mencionar del archivo es el nombre de la unidad de persistencia (en este caso llamado *CLASSModelerPU*), este nombre se ubica en la etiqueta <persistence-unit name=*"CLASSModelerPU"*> y representa la identificación del recurso interno para la aplicación. Mediante este nombre el contenedor es capaz de realizar la inyección del objeto *EntityManager* en clases EJB o cualquier otra que lo requiera. Esto se explica un poco más a fondo en las siguientes secciones.

#### Interfaz grafica

Los elementos de interfaz gráfica son básicamente paginas XHTML, estas páginas siguen el mismo concepto descrito en la sección 7.3.3 y son implementadas en base a plantillas. A continuación se muestran algunas imágenes tomadas de las interfaces graficas desarrolladas para esta iteración.

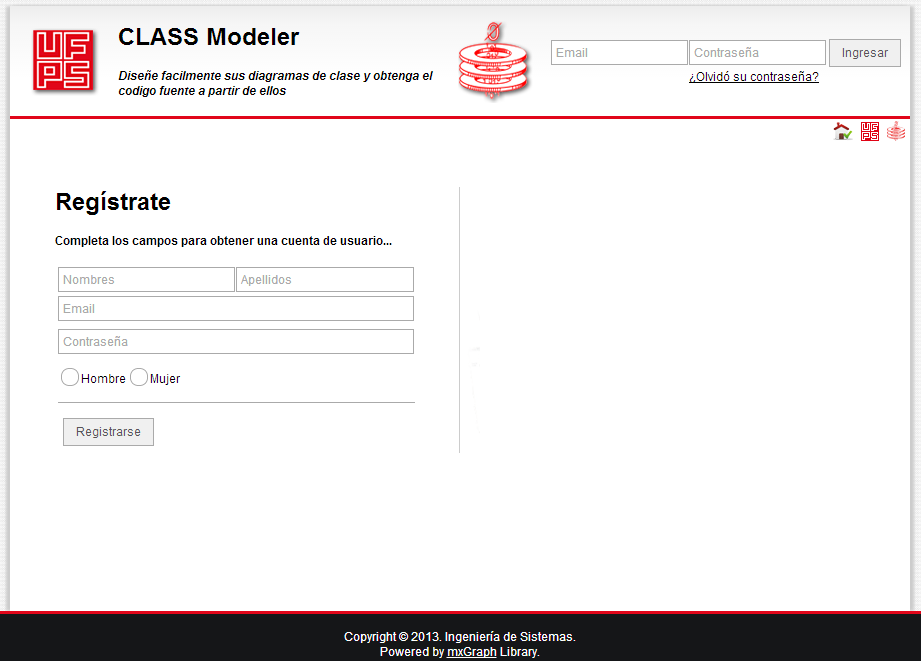


Figura . Interfaz gráfica principal de la aplicación CLASS Modeler

La página principal *index.html* se basa en la plantilla *portal.xhtml* (ver la Figura 48), y sobrescribe la parte central de dicha plantilla. Esta contiene 2 secciones, la sección izquierda *signUp.xhtml* contiene el formulario de registro para nuevos diagramadores, el cual fue diseñado para cumplir con la historia de usuario H1, la sección derecha aún se encuentra vacía y está destinada para una historia de usuario próxima.

La parte superior de la página contiene el encabezado de la aplicación, el cual es común para todas las demás páginas, y además contiene en la sección derecha, que contiene el formulario de inicio de sesión. De momento este formulario no realiza ninguna opción y será trabajado en una iteración posterior.

El formulario de creación de cuentas de diagramador *signUp.xhtml* se puede ver a continuación.

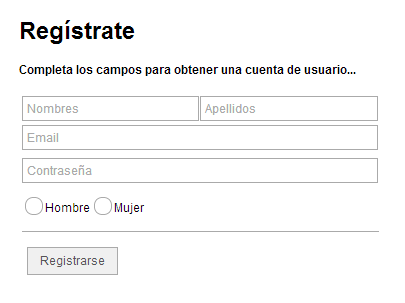


Figura . Interfaz gráfica para la creación de cuenta de diagramador

Por medio del formulario llamado *forgotPassword.xhtml* mostrado a continuación, el diagramador puede solicitar la reasignación de la contraseña para su cuenta, este formulario es mostrado cuando el diagramador pulsa sobre el link *“¿Olvidó su contraseña?”* que se sitúa en la parte superior de la página, debajo del formulario de inicio de sesión.

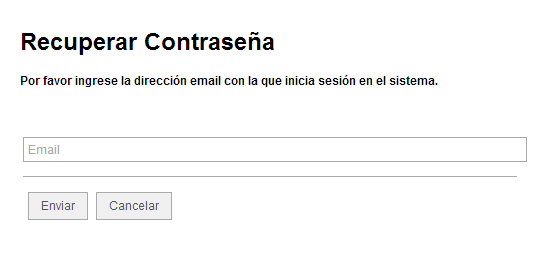


Figura . Interfaz gráfica para solicitud de reasignación de contraseña

Una vez solicitada la reasignación, el link de confirmación enviado al correo del diagramador conduce al formulario llamado *resetPassword.xhtml*, el cual le permite cambiar su contraseña. A continuación se muestra dicho formulario.

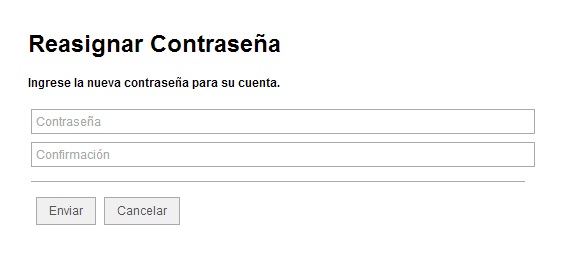


Figura . Interfaz gráfica para reasignar la contraseña

#### Controles

Los controladores desarrollados para esta iteración son los siguientes:

* SignUpControllerBean: JSF bean que controla la interacción del usuario con el formulario para crear una cuenta de diagramador. Este controlador tiene el ámbito @View, lo cual significa que mantiene su información mientras el usuario se encuentre visualizando dicho formulario.
* ActivateAccountControllerBean: JSF bean de ámbito @Request, el cual controla la activación de la cuenta de diagramador.
* ForgotPasswordControllerBean: JSF bean de ámbito @View que controla el formulario para solicitar la reasignación de la contraseña para la cuenta de diagramador.
* ResetPasswordControllerBean: JSF bean de ámbito @View que controla el formulario para reasignar la contraseña de la cuenta de diagramador.

#### Servicios

Para esta iteración se definieron varios servicios que componen la capa de lógica de negocio de las historias de usuario abarcadas (ver sección 7.1.4 Arquitectura de la Aplicación). Estos servicios proveen métodos básicos para cada funcionalidad, tales como insertar diagramador, activar la cuenta, enviar el email de activación de la cuenta.

Estos servicios están definidos por las interfaces:

* UserService
* SecurityService
* EmailService

El siguiente diagrama de clases muestra la relación de los servicios para esta iteración, esta relación se mantiene y reutiliza para varias de las otras funcionalidades e iteraciones.



Diagrama . Diagrama de clases de los servicios para la iteración 1

Las interfaces son marcadas con la anotación @Local, que indica que la implementación de la clase es un recurso en el mismo contenedor.

Los servicios son implementados como clases Enterprise Java Beans Stateless, es decir, clases sin estado, debido a que estos sus métodos solo contienen lógica de negocio y no necesitan guardar un valores temporales a lo largo de la ejecución de sus métodos. Algunos servicios descritos en las siguientes iteraciones necesitan mantener estado e inclusive necesitan compartir información a través de diferentes sesiones y peticiones de usuarios.

Cada clase implementación de la interfaz definida para un servicio, es marcada con la anotación @Stateless, esto le indica al contenedor (Glassfish) que esta clase representa un componente Enterprise que debe ser publicado y reutilizado como un recurso de otras clases.

El acceso a los datos desde los servicios es realizado a través del Java Persistence API (JPA), esta API provee los mecanismos para obtener, insertar, actualizar y borrar registros de la base de datos mediante una interfaz central llamada *EntityManager*. Esta interfaz dispone de métodos tanto para operaciones básicas (CRUD), como para operaciones complejas como ordenamientos y agrupaciones. El objeto EntityManager se agrega al servicio mediante *Inyección de Dependencias[[22]](#footnote-22)*, y es realizado de manera automática por el contenedor. Lo único que se necesita es un atributo en la clase del servicio en el cual se almacena el objeto. Dicho atributo es marcado con una anotación especial llamada @PersistenceContext. El nombre de la unidad de persistencia debe ser exactamente el mismo que se asignó en el archivo *persistence.xml*.



Diagrama . Diagrama de clases de implementacion para servicos de la iteración 1

Una vez mencionados los servicios y su estructura, pasamos a describir la interacción básica de éxito para cada historia de usuario abarcada en esta iteración, los 4 siguientes diagramas de secuencia muestran el flujo de mensajes y los componentes que intervienen en cada una de ellas. Cada diagrama describe únicamente el flujo principal de la historia de usuario, sin embargo en cada una existen flujos alternativos que conducen a resultados diferentes. En cada diagrama se muestran los servicios utilizados para llevar a cabo la historia de usuario, sin embargo estos son representados a nivel de interfaces y no de la implementación concreta en EJB, esto es debido a que se quiere resaltar el flujo del proceso global sin mencionar los detalles de implementación.



Diagrama . Diagrama de secuencia para creación de cuenta de diagramador H1

**Descripción del Proceso**:

1. Una persona en calidad de visitante accede a la página de bienvenida de la aplicación, y diligencia los datos de registro que se encuentran allí.
2. Pulsa sobre el botón “Registrarse”.
3. Internamente se envía una petición HTTP (Request) con los datos ingresados por el visitante, esta petición es asíncrona por lo tanto la página no se recarga.
4. Los datos son recibidos por el controlador Servlet de JSF (Faces Servlet), el cual se encarga de ejecutar todo el ciclo de vida de las peticiones JSF.
5. En primera instancia se ejecutan los validadores, los cuales determinan si los datos ingresados son correctos. Estos validadores comprueban si los campos marcados como obligatorios tienen valor, y si la cantidad de caracteres es la correcta.
6. Para este formulario se desarrollaron 2 validadores JSF personalizados (*SignUpEmailJSFValidator* y *PasswordJSFValidator*), estos validadores comprueban que la dirección email tenga el formato correcto y que la contraseña tenga la cantidad de caracteres mínima obligatoria, respectivamente.
7. En caso de que durante los validadores se encuentre algún problema, se notifica a la vista y se muestran mensajes de alerta.
8. Una vez ha finalizado la fase de validación, se procesan los datos y se envían desde el Servlet controlador de JSF al controlador específico del formulario, llamado *SignUpControllerBean*.
9. A esta clase se le pasan los datos ingresados por el usuario mediante los métodos set de cada atributo, estos atributos concuerdan con los del bean *Diagrammer*, el cual será inicializado con estos los valores.
10. Luego el Servlet de JSF invoca el método en el controlador *SignUpControllerBean.processAJAX(),* el cual se encarga mediante el uso del servicio (*UserService*) de crear la cuenta de diagramador. Este controlador tiene el ámbito (*View*) lo que significa que el objeto es instanciado automáticamente por el contendor una vez se renderiza la página y permanece vivo mientras el usuario se encuentre visualizando la misma página, el método mencionado anteriormente es enlazado con el evento clic del botón en el formulario. El servicio es creado mediante inyección de dependencias por el contenedor de EJBs. Este concepto permite la instanciación automática de los objetos cada vez que son requeridos, y son destruidos una vez el objeto ya no es necesario. Además que permiten realizar la inyección de la implementación adecuada para la interfaz definida.
11. La implementación del servicio (*UserServiceBean*) contiene toda la lógica necesaria para crear la cuenta de diagramador, realizando las validaciones necesarias, por ejemplo que la dirección email sea única. Luego de hacer estas validaciones, se persiste en base de datos el nuevo diagramador, se crea el código de verificación y se envía el correo de confirmación a la dirección ingresada.
12. Finalmente se le notifica al visitante que la cuenta ha sido creada y que debe revisar su dirección de correo para activar la cuenta.



Diagrama . Diagrama de secuencia para activación de cuenta de diagramador H2

**Descripción del proceso**:

1. En la historia de usuario anterior se envió el correo de activación de la cuenta a la dirección email ingresada por el diagramador.
2. El diagramador ingresa a su dirección de correo y pulsa sobre el link de activación, el cual contiene el código de activación y la dirección email para confirmar los datos.
3. Cuando se pulsa el link, se envía una petición HTTP al controlador de JSF, el cual pasa el flujo de la petición al controlador llamado *ActivateAccountControllerBean*, el cual toma los parámetros de la petición y valida los datos. En caso de no encontrar la cuenta de diagramador o de no encontrar una dirección valida se cancela el proceso y se le notifica a la persona con un mensaje adecuado.
4. La verificación de los datos es realizada por el servicio (*UserService*), más específicamente el método *UserService.activateDiagrammerAccount()*, el cual es invocado desde el controlador mencionado anteriormente.
5. Adicional a las verificaciones básicas se comprueba si el código de activación no ha expirado, en caso de haberlo hecho, se genera un nuevo código y se envía nuevamente el email de activación mencionado en la historia anterior.
6. El código de activación es único y está unido a la dirección email del diagramador, por lo tanto si el código no concuerda con el email, este se considera inválido y no se continúa el proceso.
7. Una vez se pasan todas las validaciones se procede a cambiar el estado de la cuenta a ACTIVATED y se actualiza el registro en base de datos.
8. Por último se notifica al diagramador que la cuenta ha sido activada correctamente, esta notificación se realiza re-direccionando al usuario a la página *activateAccount.xhtml* en donde se muestra un mensaje con el resultado del proceso.



Diagrama . Diagrama de secuencia para solicitud de nueva contraseña H3



Diagrama . Diagrama de secuencia para reasignación de contraseña H3

**Descripción del Proceso:**

El proceso de reasignación de contraseña de divide en 2 partes, inicialmente la solicitud de reasignación de contraseña y posterior el cambio de contraseña por parte del diagramador.

* Solicitud de reasignación de contraseña:

1. El diagramador inicia el proceso al hacer clic en el link (*Olvidó la contraseña?)* ubicado en la página principal de la aplicación.
2. Este link redirige a la página *forgotPassword.xhtml*, dentro de la cual se solicita únicamente la dirección email usada por el diagramador.
3. Una vez se llena el campo y al presionar clic sobre el botón de aceptar, se envía el formulario con los datos. Esta petición es recibida y procesada por el bean JSF *ForgotPasswordControllerBean*, el cual se encarga únicamente de verificar el formato del email ingresado y de pasar los datos al servicio *UserService.resetPassword()*.
4. Internamente el servicio implementado (*UserServiceBean*) valida la existencia de la cuenta de usuario y verifica que la cuenta se encuentre ACTIVADA. En caso de que esta verificación no sea válida, entonces se arroja una excepción personalizada y se aborta el proceso, notificando al usuario sobre el problema encontrado.
5. Una vez ejecutadas las validaciones, se genera el código de verificación y se envía un email a la dirección ingresada. Esto se realiza para tener constancia que el propietario de la cuenta es quien está intentando recuperar o cambiar la contraseña.

* Reasignación de contraseña:

1. Como se mencionó anteriormente, el correo de verificación que se envía al email del diagramador contiene la información necesaria para realizar la reasignación. El diagramador solo tiene que hacer clic en la URL y automáticamente se lanza esta segunda parte el proceso.
2. El link envía una petición que es procesada por el bean controlador *ResetPasswordControllerBean*, el cual válida la información (email y código de verificación) y redirige el proceso a la vista que contiene el formulario de reasignación (*resetPassword.xhtml*).
3. En la vista mencionada el diagramador debe ingresar la nueva contraseña y la confirmación de la misma. Al hacer clic sobre el botón confirmar, se envía la petición y se procesa por el controlador *ResetPasswordControllerBean* el cual valida que los dos valores sean idénticos y que correspondan con una dirección email válida.
4. Luego los datos se pasan al servicio (*UserServiceBean*) el cual se encarga de modificar los datos de la cuenta.

#### Pruebas de unidad

Las pruebas de unidad están enfocadas a verificar el correcto funcionamiento de componentes específicos de la aplicación, en este caso estas pruebas van dirigidas directamente a probar los servicios provistos por la aplicación. Por lo tanto para esta iteración se encargaran de probar los servicios para crear diagramadores, enviar correos electrónicos y generar códigos de verificación.

A continuación se listan en la siguiente tabla, las pruebas unitarias realizadas a los servicios que intervienen en esta iteración.

|  |  |
| --- | --- |
| **Componente** | **Pruebas de unidad** |
| UserServiceTest | testExistsUser (); |
| testInsertDiagrammer (); |
| testInsertDiagrammer\_failAlreadyExisting(); |
| testActivateDiagrammerAccount(); |
| testActivateDiagrammerAccount\_failNonExisting(); |
| testActivateDiagrammerAccount\_invalidAccountStatus(); |
| testActivateDiagrammerAccount\_invalidSecurityCode(); |
| testActivateDiagrammerAccount\_expiredSecurityCode(); |
| testRequestResetPassword(); |
| testResetPassword(); |
| testGetDiagrammerByEmail(); |
| SecurityServiceTest | testInsertSecurityCode(); |
| testGetSecurityCode(); |
| EmailServiceTest | testSendEmail(); |
| testSendAccountActivationEmail(); |
| testSendResetPasswordEmail(); |

Tabla . Listado de pruebas de unidad para la iteración 1

El proceso de pruebas unitarias se realizó mediante el uso de un Framework para pruebas automáticas en Java. TestNG permite la realización de pruebas automáticas desde las pruebas unitarias hasta las pruebas de integración[[23]](#footnote-23).

En TestNG como en algunos otros Frameworks para testing, basta con marcar las clases con la anotación @Test para que puedan ser reconocidas y ejecutadas por el Framework.

A continuación se muestra una fracción de código utilizado para una de las pruebas unitarias que se realizaron, este se encuentra en la clase *UserServiceTest*.

/\*\*

\* Unit test to verify the service method {@link UserService#insertDiagrammer(Diagrammer)}.

\*

\* **@throws** SendEmailException

\* **@throws** InvalidUserAccountException

\* **@author** Gabriel Leonardo Diaz, 11.09.2013.

\*/

**public** **void** testInsertDiagrammer () **throws** InvalidUserAccountException,

SendEmailException {

String email = "leonar248@gmail.com";

Diagrammer diagrammer = **new** Diagrammer();

diagrammer.setFirstName("Nombre Diagramador");

diagrammer.setLastName("Apellido Diagramador");

diagrammer.setEmail(email);

diagrammer.setPassword("12345");

diagrammer.setGender(EGender.*MALE*);

userService.insertDiagrammer(diagrammer);

Diagrammer afterInsert = userService.getDiagrammerByEmail(email);

**try** {

**assert** (afterInsert != **null**) :

"Error: The diagrammer was not created";

**assert** (afterInsert.getAccountStatus()==EDiagrammerAccountStatus.*INACTIVATED*):

"Error: The account status must be INACTIVATED after inserting";

**assert** (email.equals(afterInsert.getEmail())) :

"Error: The email does not match with the saved value.";

}

**finally** {

userService.deleteDiagrammer(diagrammer.getKey());

}

}

Una vez se invoca el servicio con la información necesaria, se verifican los resultados del mismo mediante sentencias *assert.* Este es el mismo mecanismo usado para todas las pruebas unitarias realizadas en cada iteración. Por último se borra la información creada por la prueba unitaria de tal manera que el ambiente permanezca en el mismo estado con el cual se inició.

Las siguientes imágenes muestra el resultado de las pruebas unitarias hechas para cada servicio.

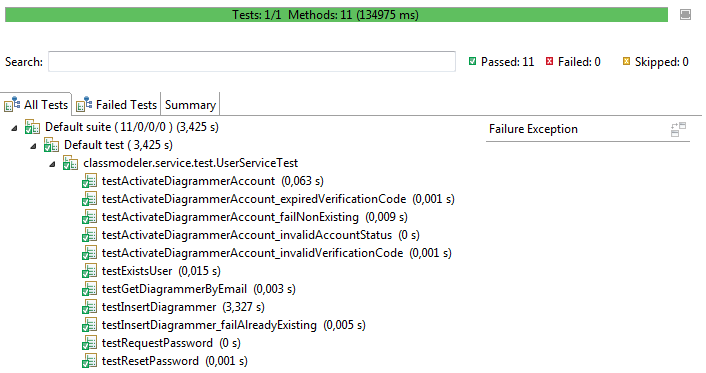


Figura . Resultados de las pruebas unitarias del servicio *UserServiceTest*

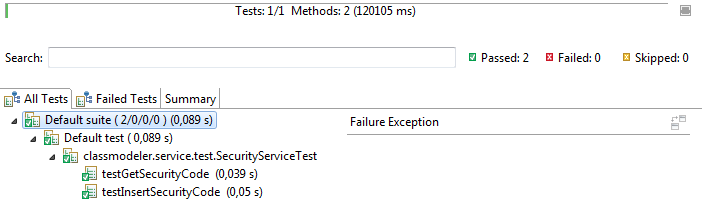


Figura . Resultados de las pruebas unitarias del servicio *SecurityServiceTest*

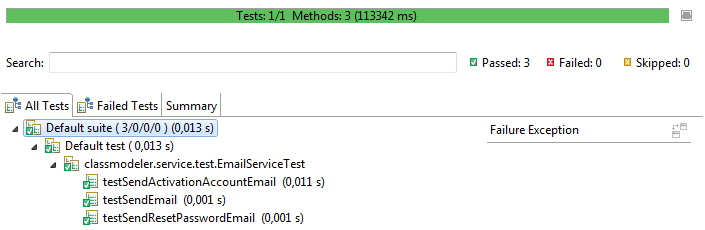


Figura . Resultados de las pruebas unitarias del servicio *EmailServiceTest*

### Iteración 2

Esta iteración incluye únicamente 2 historias de usuarios, H6 y H9. Estas historias abarcan las funcionalidades de Iniciar Sesión y Crear Diagrama respectivamente, lo que significa que una vez terminada esta iteración, un diagramador podrá iniciar sesión en el sistema y podrá crear un nuevo diagrama para el diagramador. A continuación se describe paso a paso las actividades realizadas y los componentes desarrollados para cumplir con los requerimientos.

#### Actividades a Realizar

* Investigar el mecanismo para mantener sesiones de usuario en una aplicación Java Web JSF.
* Investigar sobre aspectos de seguridad en el desarrollo de aplicaciones Web Java, tales como evitar ingresos no autorizados o forzar el inicio de sesión para acceder a determinada página.
* Diseñar la interfaz gráfica para el formulario de inicio de sesión.
* Modelar e Implementar los servicios necesarios para el inicio de sesión.
* Implementar el mecanismo de inicio de sesión, considerando los diferentes tipos de usuarios que pueden existir en la aplicación.
* Implementar los mecanismos de seguridad para proteger la aplicación de accesos no autorizados.
* Diseñar una interfaz gráfica para listar los diagramas creados por un diagramador.
* Implementar la búsqueda de los diagramas que pueden ser accedidos por el diagramador.
* Diseñar la interfaz gráfica para crear un diagrama.
* Modelar e Implementar los servicios necesarios para crear diagramas.
* Implementar el mecanismo de guardado de los nuevos diagramas.

#### Diagrama de Clases

El diagrama de clases para esta iteración agrega algunos conceptos nuevos a la iteración anterior, más específicamente el concepto Diagrama. Este objeto del dominio encapsula la información básica de un diagrama creado por un usuario. La siguiente imagen muestra las clases del domino de la iteración 2.



Diagrama . Diagrama de clases de la iteración 2

Un diagrama puede incluir una variedad de elementos UML, tales como Clases, Interfaces, Enumeraciones y varios tipos de relaciones. Todos los elementos del diagrama son convertidos y representados mediante el estándar XMI, dicha representación es almacenada como texto en el atributo llamado *XMI* de la clase *Diagram*. Esto permite cierto nivel de compatibilidad y portabilidad de los diagramas creado por la aplicación, además de liberar la aplicación de la complejidad del meta modelo UML al almacenar no los elementos individuales sino una representación global del diagrama que puede ser convertida de vuelta en un gráfico con la notación pertinente.

Adicionalmente un diagrama contiene algunos datos de auditoria, tales como el creador del diagrama y el último diagramador que realizó una modificación al mismo. El atributo *createdBy* determina el propietario del diagrama y el atributo *modifiedBy* determina el ultimo diagramador que realizó una modificación. Estos campos de auditoria permiten conocer de manera sencilla, información relevante con respecto al manejo que han tenido los diagramas (usado más adelante cuando se permita compartir diagramas entre diferentes usuarios).

#### Modelo de Datos

En relación con el nuevo objeto de dominio añadido en esta iteración, se incluyó una nueva tabla que almacena la información de los diagramas. La siguiente imagen ilustra el modelo de datos para esta iteración.

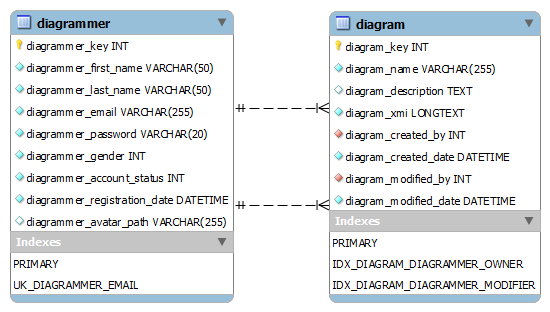


Diagrama . Modelo de datos de la iteración 2

La tabla *diagram* almacena la información de los diagramas creados por cada diagramador. Lo más representativo es la columna llamada *diagram\_xmi* la cual es de tipo LONGTEXT, esto debido a que la información almacenada allí puede ser de gran longitud, debido a que es un texto en formato XML. Las demás columnas almacenan información representativa del diagrama tal como el nombre y la descripción, además de los campos de auditoria que son llaves foráneas de la tabla *diagrammer*.

#### Interfaz grafica

La interfaz gráfica de esta iteración incluye el formulario para inicio de sesión que fue tomado desde la iteración anterior, además de la página que sirve de tablero de control para los diagramas que pertenecen el diagramador y por último el formulario para creación de diagramas.

A continuación se muestran algunas imágenes de las interfaces graficas desarrolladas para esta iteración.

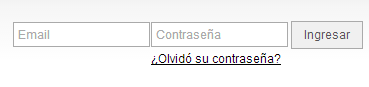


Figura . Formulario de inicio de sesión

Este formulario se encuentra en la página *logIn.xhtml*, la cual fue incrustada en la página principal *index.xhtml* desarrollada en la iteración anterior.

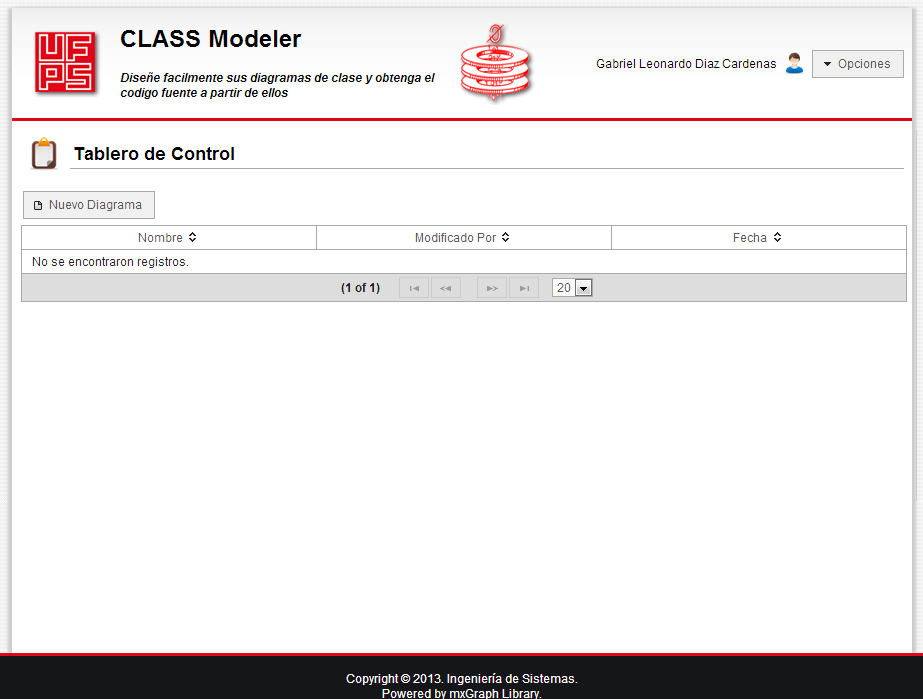


Figura . Tablero de control para listar los diagramas del usuario

El tablero de control *dashboard.xhtml* se basa al igual que la página principal en la plantilla *portal.xhtml* (ver la Figura 48), de esta “hereda” el encabezado, el pié de página y una sección central en blanco. La sección central se reemplaza con la tabla que contiene los diagramas creados por el diagramador y del encabezado se reemplaza el formulario de inicio de sesión, por el panel *userSection.xhtml* que contiene el nombre del diagramador, el avatar o imagen representativa y un menú desplegable que contendrá opciones para manipular la cuenta.

El tablero de control contiene una tabla en donde se listan todos los diagramas creados por el diagramador, esta tabla es un componente pertenece a la librería llamada PrimeFaces. Esta tabla es creada con una lista de objetos de tipo *Diagram*, y cada fila representa un objeto dentro de la lista. El componente prefabricado responde a la etiqueta <p:dataTable/> y permite añadir características, tales como filtros, paginación, selección, ordenamiento y algunos otros. La tabla se alimenta con la lista de diagramas que se encuentran en el controlador *DashboardControllerBean*. Las columnas de la tabla son fijas y se definen estáticas en el código.

A continuación se muestra una porción del código utilizado para crear este componente:

<p:dataTable id=*"diagramsTable"*

value=*"#{dashBoardController.diagrams}"*

var=*"diagram"*

emptyMessage=*"#{msgs.EMPTY\_TABLE\_MESSAGE}"*

resizableColumns=*"true"*

rowKey=*"#{diagram.key}"*

selectionMode=*"single"*

selection=*"#{dashBoardController.diagram}"*

paginator=*"true"*

paginatorPosition=*"bottom"*

rowsPerPageTemplate=*"10, 20, 30"*

rows=*"20"*>

<p:column headerText=*"#{msgs.COLUMN\_DIAGRAM\_NAME\_LABEL}"* sortBy=*"#{diagram.name}"*>

<h:commandLink title=*"#{msgs.BUTTON\_OPEN\_DESIGNER\_DIAGRAM\_LABEL}" /*>

</p:column>

<p:column headerText=*"#{msgs.COLUMN\_DIAGRAM\_MODIFIER\_LABEL}"*>

<h:outputText value=*"#{formatController.getFormattedUserName(diagram.modifiedBy)}"*>

</p:column>

<p:column headerText=*"#{msgs.COLUMN\_DIAGRAM\_MODIFIED\_DATE\_LABEL}"*>

<h:outputText value=*"#{formatController.getFormattedDate(diagram.modifiedDate)}"* />

</p:column>

</p:dataTable>

En el controlador es necesario disponer de un atributo de tipo lista con los métodos de acceso para que pueda ser invocado por el Framework en el momento de renderizar al página *dashboard.xhtml*.

Estos diagramas son cargados mediante la invocación a un servicio desarrollado para esta labor específica, el cual permite obtener los diagramas cuyo propietario es el diagramador enviado como parámetro.

* *DiagrammService.getDiagramsByDiagrammer(Diagrammer)*;

El controlador envía como parámetro el diagramador que se encuentra en sesión.

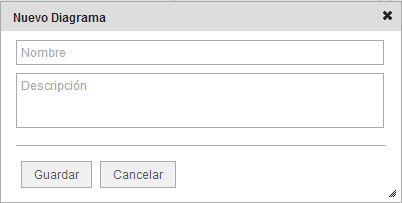


Figura . Formulario de creación de diagramas.

El formulario de creación de diagrama solicita la información básica del diagrama, nombre y descripción, este formulario se muestra en una ventana emergente al pulsar sobre el botón *Nuevo Diagrama* que se encuentra en la parte superior de la tabla. El nombre del diagrama es un campo obligatorio, sin embargo la descripción es opcional y el diagramador decide si la llena o no.

#### Seguridad

La seguridad es proporcionada por el mecanismo de sesiones en el servidor, cada sesión tiene un identificador único que se almacena en una cookie del cliente llamada *jsessionid*, esta cookie se envía en cada petición realizada y es usada para identificar la sesión del servidor creada para el usuario autorizado. Todo este mecanismo es trasparente para el desarrollador y el encargado del mantenimiento es el mismo Framework JSF.

Las sesiones son controladas por el bean manejado de JSF llamado *SessionControllerBean*, este bean tiene el ámbito @Session, por lo tanto permanece en memoria hasta que es destruido mediante la invocación del proceso de cierre de sesión o cuando se agota el tiempo máximo de inactividad en el sistema.

Esta clase almacena la información del usuario que ingresa al sistema, además sirve de suministro para los demás controladores de la aplicación que necesitan la información del usuario en sesión. La siguiente imagen muestra la estructura de la clase, los atributos y métodos que esta posee.

Otro aspecto importante de seguridad es la protección contra accesos no autorizados a páginas que requieren un inicio de sesión previo, tal como la página *dashboard.xhtml* que representa el tablero de control para los diagramas de un usuario.

Esta característica es manejada mediante un filtro web, el cual cumple la función de intersectar las peticiones enviadas desde el cliente y determinar si se trata de una petición autorizada. Este filtro es implementado por la clase *SessionFilter* que utiliza el API Java Servlet y se encuentra marcada con la anotación @WebFilter. La anotación permite añadir meta datos al filtro, tales como el nombre y los patrones de URL que este debe intersectar. Básicamente el filtro se ejecuta cada vez que se solicita la página que concuerda con los patrones definidos.

El siguiente diagrama representa el proceso de ejecución del componente desarrollado.



Diagrama . Proceso de filtro de peticiones hacia las páginas de la aplicación

Este es el proceso básico que se ejecuta cuando se solicita una página de la aplicación, si la página concuerda con el patrón “/pages/\*” y las validaciones no son satisfactorias, entonces se redirige al visitante a la página inicial *index.xhtml*. Los patrones básicamente se basan en las rutas relativas en la estructura de carpetas del proyecto web. Es decir, cualquier página que se encuentre sobre la carpeta “pages” requiere una autorización previa, por lo tanto para acceder a cualquiera de ellas se debe haber iniciado sesión.

#### Servicios

Los servicios definidos para esta iteración básicamente comprenden el manejo del inicio de sesión y los métodos que permiten manejar diagramas. Esta iteración agrega dos servicios adicionales llamados *SessionService* y *DiagramService*.



Diagrama . Diagrama de clases de los servicios para la iteración 2

A continuación se describe mediante diagramas de secuencia, el proceso básico de éxito de las dos historias de usuario de esta iteración.



Diagrama . Diagrama de secuencia de la historia de usuario H6

**Descripción del proceso:**

1. Un diagramador accede a la página principal de la aplicación con el objetivo de ingresar y utilizar sus servicios.
2. En el formulario de inicio de sesión *logIn.xhtml*, se solicita el email y la contraseña de la cuenta.
3. El diagramador ingresa la información y pulsa sobre el botón *Ingresar*, tras esta acción se envía una petición al servidor que es capturada por el Servlet de JSF y transfiere el flujo al método *process()* de clase controladora *LogInControllerBean*, esta clase procesa la petición y valida que los datos sean correcto, es decir, que el email y la contraseña no sean vacíos. En caso de no pasar la validación, se cancela la operación y se notifica a la vista.
4. Una vez se realiza la validación y el resultado es satisfactorio se transfiere el flujo a la clase principal para el manejo de sesiones, llamada *SessionControllerBean* la cual almacena los objetos pertinentes de sesión. Esta clase hace uso del servicio *SessionService*, quien tiene la lógica necesaria para determinar si las credenciales usadas para ingresar (es decir, el email y la contraseña) pertenecen a una cuenta de diagramador valida.
5. El servicio invocado es *SessionService.logIn(String, String)*, el cual busca la cuenta de diagramador asociada al email recibido y verifica en primera instancia que la cuenta exista, si no existen entonces se arroja una excepción. Como segundo paso verifica que la cuenta se encuentre activada. De no ser así, se arroja una excepción de tipo *InvalidDiagrammerAccountException*, la cual aborta el proceso de inicio de sesión; esta excepción es capturada en el controlador *LogInControllerBean* quien a su vez notifica a la vista para informarle al diagramador que su cuenta no ha sido activada.
6. Si la cuenta se encuentra en estado ACTIVATED, se procede a validar la contraseña, la cual es comparada con la que se encuentra almacenada en base de datos. Si la comparación determina que la contraseña es inválida entonces se aborta el proceso con otro tipo de excepción.
7. Por ultimo habiendo pasado todas las validaciones se retorna la cuenta de usuario que pertenece a dicha persona y se devuelve el flujo al controlador *SessionControllerBean*, el cual procede a almacenar la cuenta en memoria y a mantenerla allí mientras la sesión se encuentre activa.
8. Se redirige hacia la vista *dashboard.xhtml* que muestra el tablero de control con los diagramas pertenecientes al diagramador que inició sesión.



Figura . Diagrama de secuencia de la historia de usuario H9

**Descripción del proceso**

1. El proceso inicia cuando el diagramador pulsa sobre el botón *Nuevo Diagrama*. Inmediatamente se abre un panel de tipo modal que contiene el formulario de creación de diagramas.
2. El formulario presenta dos cajas de texto, una para el nombre del diagrama y la otra para la descripción del mismo.
3. El diagramador llena los campos y pulsa sobre el botón *Guardar*.
4. Los datos son enviados al servidor en una petición HTTP y son procesados por el controlador *DiagramControllerBean*.
5. Luego de validar que los datos sean correctos, es decir, que el nombre del diagrama no este vacio, el controlador invoca al servicio *DiagramService.insertDiagramm().*
6. Este servicio recibe un objeto de tipo *Diagram*, el cual debe contener como minimo el nombre, y el diagramador que es propietario del diagrama. Estos datos son configurados por el controlador un momento antes de invocar el servicio. El propietario es el diagramador en sesión, por lo tanto este objeto se toma del controlador de la sesión *SessionControllerBean*.
7. Internamente el servicio insertDiagram(), se encarga de poner los campos adicionales como la representación XMI para un diagrama vacio, la fecha de creación y modificación.
8. La información se persiste en base de datos y se retorna el nuevo objeto con todos los campos asignados.
9. El controlador DiagramControllerBean, recibe el objeto y lo añade al listado de diagramas que se muestran en la tabla

#### Pruebas de unidad

Las pruebas unitarias de los servicios de esta iteración se distribuyen de la siguiente manera.

|  |  |
| --- | --- |
| **Componente** | **Pruebas de unidad** |
| SessionService | testLogIn(); |
| testLogIn\_failNonExistingAccount(); |
| testLogIn\_failNonActivatedAccount(); |
| DiagramService | testInsertDiagram(); |
| testInsertDiagram\_failNonExistingAccount(); |

Tabla . Listado de pruebas de unidad para la iteración 2

**Resultados de las pruebas de unidad**

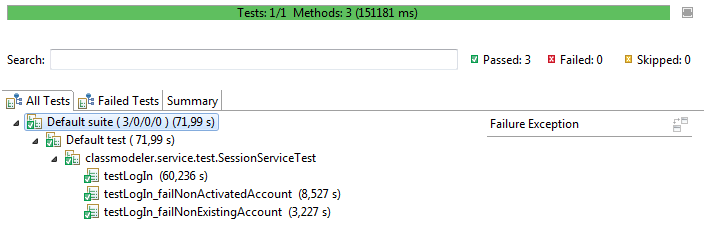


Figura . Resultados de las pruebas de unidad del servicio SessionService

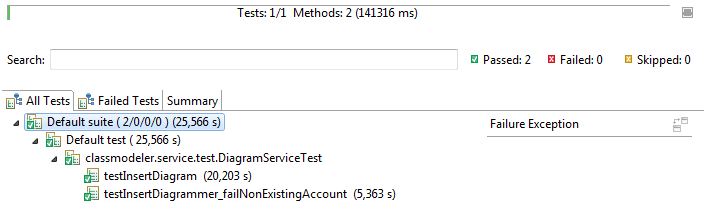


Figura . Resultados de las pruebas unitarias del servicio DiagramService

### Iteración 3

### Iteración 4

### Iteración 5

### Iteración 6

### Iteración 7

### Iteración 8

### Iteración 9

## DESPLIEGUE

El despliegue de la aplicación consiste en copiar el código fuente compilado, en el servidor de aplicaciones. Este código fuente puede estar empaquetado en un archivo .EAR[[24]](#footnote-24), o también se puede hacer mediante el copiado manual de todos los archivos en una carpeta del servidor.

Básicamente este proceso se debe hacer una vez el servidor de aplicaciones Glassfish este en ejecución, inclusive el mismo servidor provee algunos mecanismos para el despliegue de la aplicación.

La segunda tarea importante para el despliegue de la aplicación es la instalación del script de base de datos, el cual contiene la estructura de tablas usadas para almacenar la información persistente de la aplicación. Esta tarea incluye configurar el Data Source para MySQL en el servidor de aplicaciones Glassfish.

Por último es necesario realizar algunas configuraciones pequeñas antes de desplegar la aplicación, tal como configurar la cuenta de correo que servirá de remitente al momento de enviar correos desde la aplicación.

En los anexos del proyecto se encuentra el documento Guía de Instalación llamado *CLASS Modeler – Installation Guide.pdf*, dicho documento sirve de guía para realizar las configuraciones necesarias y desplegar correctamente la aplicación.

El siguiente diagrama de despliegue obedece a la arquitectura física del proyecto con los

# CONCLUSIONES

# RECOMENDACIONES

# BIBLIOGRAFIA

1. **Amaya Torrado, Yegny Karina y Herrera Angarita, Lady Torcoroma.** *Identificación de las causas que generan problemas en el aprendizaje de fundamentos de programación de computadores en las facultades de ingenieira de sistemas de las universidades de la ciudad de Cúcuta.* Cúcuta N. de S. : Universidad Francisco de Paula Santander, Biblioteca Eduardo Cote Lamus, 2003. TIS 371.334A489i.

2. **Sommerville, Ian.** *Ingenieria de Software.* Madrid : Pearson Education S.A., 2005. ISBN 8478290741.

3. **Rumbaugh, James, Booch, Grady y Jacobson, Ivar.** *El lenguaje unificado de modelado, Manual de Referencia.* Madrid : Pearson Education S.A., 2000. ISBN 8478290370.

4. **Dominguez, José Alberto.** UML: Lenguaje Unificado de Modelado. [En línea] 14 de Marzo de 2009. [Citado el: 8 de Octubre de 2011.] http://www.que-informatica.com/index.php/tag/uml.

5. **Pressman, Roger.** *Ingenieria de Software un enfoque práctico.* Mexico : McGraw Hill, 2005. ISBN 9701054733.

6. **Creative Commons.** Atribución No Comercial Compartir igual a 2.5 (Colombia). [En línea] http://creativecommons.org/licenses/by-nc-sa/2.5/co/legalcode.

7. **Sun Microsystems.** Licencia de la tecnologia Java SE. [En línea] http://www.bhweb.es/cdlibre/pdf/jvm.pdf.

8. **Free Software Fundation Inc.** GNU General Public Licence. [En línea] 23 de Junio de 2007. http://www.gnu.org/licenses/gpl.txt.

9. **Universidad Francisco de Paula Santander, Programa de Ingenieria.** Pensum académico vigente. [En línea] 2011. http://ingsistemas.ufps.edu.co/images/Descargas/pensum115.pdf.

10. **Sparks, Geoffrey.** Enterprise Architect User Guide. [En línea] 3 de 2012. http://www.sparxsystems.com.au/bin/EAUserGuide.pdf.

11. **StarUML.** StarUML 5.0 Guía de Usuario (Versión en Ingles). [En línea] http://staruml.sourceforge.net/docs/user-guide(en)/toc.html.

12. **Ramirez, Alejandro, y otros.** ArgoUML User Manual: A tutorial and reference description. [En línea] 2011. http://argouml-downloads.tigris.org/nonav/argouml-0.34/manual-0.34.pdf.

13. **Group, OMG Object Management.** OMG Unified Modeling Language Superstructure. [En línea] 6 de Agosto de 2011. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/. formal/2011-08-06.

14. **Canós, Jose H., Letelier, Patricio y Penadés, Carmen Maria.** *Metodologías Ágiles para el desarrollo de software.* Valencia : Universidad Politécnica de Valencia.

15. **Geary, David y Horstmann, Cay.** *Core Java Server Faces.* Boston : Prentince Hall, 2010. ISBN 9780137012893.

16. **Civici, Catagay.** PrimeFaces Documentation. *PrimeFaces Guia de Usuario.* [En línea] 24 de 02 de 2013. [Citado el: 08 de 03 de 2013.] http://www.primefaces.org/documentation.html.

17. **Keith, Mike y Schincariol, Merrick.** *Pro JPA 2 Mastering the Java Persistence API.* New York : Apress, 2009. ISBN 9781430219569.

18. **Aluk, Deepak, Cupri, Jhon y Malks, Dan.** *Core J2EE Patterns Best Practices and Design Strategies.* California : Prentice Hall Professional, 2003. ISBN 0131422464.

1. CASE (Computer Aided Software Engineering, Ingeniería de software asistida por computadora), son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo los costos del mismo en términos de tiempo, esfuerzo y dinero. [↑](#footnote-ref-1)
2. Página web oficial de Enterprise Architect, <http://www.sparxsystems.com/> [↑](#footnote-ref-2)
3. Tomado del portal web del programa Ingeniería de Sistemas de la Universidad Francisco de Paula Santander, en la sección Presentación del Programa – Objeto de Estudio.

   Disponible en : <http://ingsistemas.ufps.edu.co/presentacion/objeto-estudio.html> [↑](#footnote-ref-3)
4. Cloud Computing, definición, características, tipos y ventajas, información disponible en la sección 2.2.2 del documento. [↑](#footnote-ref-4)
5. Google Drive es parte de la nube de servicios proporcionada por la empresa Google para sus usuarios, a través de esta nube los usuarios pueden disponer de un kit de herramientas entre los que se incluye un procesador de texto, hojas de cálculo, diseñador de presentaciones, entre otras. [↑](#footnote-ref-5)
6. Backend se le llama a la parte de la aplicación que se ejecuta en el servidor, la cual está encargada de procesar los datos enviados desde el cliente (Frontend) y generar una respuesta. [↑](#footnote-ref-6)
7. API (“Application Programming Interface”, en español “Interfaz de programación de aplicaciones”), se refiere a un conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizada por otro software. [↑](#footnote-ref-7)
8. GWT es un Framework para la construcción de aplicaciones web en lenguaje Java, el cual permite trabajan con la tecnología AJAX (*Asynchronous Javascript and XML*) de forma trasparente para el desarrollador, evitándole tener que lidiar con los aspectos complejos y tediosos de la tecnología; además de ser compatible con la mayoría de navegadores web, lo cual es verdaderamente significativo ya que el código escrito funciona igual para todos. [↑](#footnote-ref-8)
9. ORM Object-Relational Mapping, se refiere la técnica de programación para convertir datos entre el sistema de tipos utilizado en lenguajes de programación orientados a objetos y el utilizado en una base de datos relacional, mediante un motor de persistencia. [↑](#footnote-ref-9)
10. JOYANES AGUILAR, Luis. FERNANDEZ AZUELA, Matilde. Java 2, Manual de programación. Mc Graw Hill. ISBN 8448131932. 2001, p3. [↑](#footnote-ref-10)
11. Más información en: <http://www.oracle.com/technetwork/java/javaee/documentation/index.html> [↑](#footnote-ref-11)
12. Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. [↑](#footnote-ref-12)
13. Release es una versión de lanzamiento del software, es decir, que el software se hace público. En ocasiones una versión del software puede ser una Release Candidate o candidata a lanzamiento; es decir, es una versión previa al lanzamiento definitivo de dicho software. [↑](#footnote-ref-13)
14. Universidad Francisco de Paula Santander. Estatuto estudiantil. Acuerdo 065, agosto 26, 1996.

    Disponible en : <http://www.ufps.edu.co/ufpsnuevo/archivos/reglamentacion/acuerdo065.pdf> [↑](#footnote-ref-14)
15. El Object Management Group u OMG (o por su nombre en español, Grupo de Gestión de Objetos) es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA. Es una organización sin ánimo de lucro que promueve el uso de tecnología orientada a objetos mediante guías y especificaciones para las mismas. El grupo está formado por diversas compañías y organizaciones con distintos privilegios dentro de la misma. [↑](#footnote-ref-15)
16. DRY (Del ingles Don’t Repeat Yourselft). Es un principio en el desarrollo de software enfocado en reducir la repetición de información de todo tipo, especialmente usado en arquitecturas multicapa. [↑](#footnote-ref-16)
17. RIA (Del ingles Rich Internet Applications). Son aplicaciones web que tienen la mayoría de las características de las aplicaciones de escritorio tradicionales. Estas aplicaciones usan un navegador web estándar para ejecutarse y por medio de complementos o mediante una máquina virtual se agregan características adicionales. [↑](#footnote-ref-17)
18. Información tomada de la Wikipedia online en la dirección URL: http://es.wikipedia.org/wiki/Enterprise\_JavaBeans [↑](#footnote-ref-18)
19. POJO (acrónimo de Plan Old Java Object) es una sigla utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un Framework en especial. Este acrónimo surge como una reacción en el mundo Java a los Frameworks cada vez más complejos, y que requieren un complicado andamiaje que esconde el problema que realmente se está modelando. [↑](#footnote-ref-19)
20. Una Anotación Java es una forma de añadir metadatos al código fuente Java que están disponibles para la aplicación en tiempo de ejecución. Las anotaciones pueden añadirse a clases, métodos, campos, parámetros, variables locales y paquetes. Las anotaciones Java son completamente accesibles al programador mientras que el software se ejecuta usando *Reflexión*. Las anotaciones son comúnmente precedidas del carácter @. [↑](#footnote-ref-20)
21. Se denomina Connection Pool (agrupamiento de conexiones) al manejo de una colección de conexiones abiertas a una base de datos de manera que puedan ser reutilizadas al realizar múltiples consultas o actualizaciones (tomado de <http://es.wikipedia.org/wiki/Connection_pool>). [↑](#footnote-ref-21)
22. Inyección de Dependencias (en inglés Dependency Injection, DI) es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase quien cree el objeto. [↑](#footnote-ref-22)
23. Para más información sobre TestNG puede consultar la página principal de la herramienta: http://testng.org/doc/index.html [↑](#footnote-ref-23)
24. EAR (Enterprise Archive) es un formato de archivo utilizado por Java EE para el empaquetado de uno o más módulos en un solo archivo para que el despliegue de los diferentes módulos en un servidor de aplicaciones que sucede al mismo tiempo y de manera coherente. También contiene los archivos XML llamados descriptores de despliegue que describen cómo implementar los módulos. [↑](#footnote-ref-24)