PRUDIMA, Pruebas Dirigidas por Modelos para Sistemas Distribuidos: Estudio de caso curso de Sistemas Distribuidos Universidad del Cauca

ANEXOS

Rubén Javier Gaviria Agredo
Luis Carlos Pito Díaz

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Sistemas
Grupo IDIS – Grupo de Investigación y Desarrollo en Ingeniería del Software
Popayán, Octubre de 2013

PRUDIMA, Pruebas Dirigidas por Modelos para Sistemas Distribuidos: Estudio de caso curso de Sistemas Distribuidos Universidad del Cauca

ANEXOS

Rubén Javier Gaviria Agredo
Luis Carlos Pito Díaz

Director: Ing. Pablo Augusto Magé Imbachí

Codirector: PhD. Julio Ariel Hurtado

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Sistemas
Grupo IDIS – Grupo de Investigación y Desarrollo en Ingeniería del Software
Popayán, Octubre de 2013

TABLA DE CONTENIDO

1. ANEXO A

1.1. PRUEBAS SOFTWARE

Para describir la evolución de las pruebas clásicas de software que se han realizado en la industria iniciaremos por el proceso de prueba manual (Manual Testing), pasando por los procesos de automatización de este proceso.

1.1.1. Proceso Manual

Para realizar el proceso de pruebas manuales es necesario partir del Plan de pruebas el cual da una visión general de alto nivel de tipo de pruebas y cobertura que se necesitara probar en el sistema. El diseño de la prueba se realiza de manera manual a partir requerimientos del sistema, generando los casos de pruebas deseados. Estos Casos de pruebas pueden ser de alto nivel y dar poca precisión a la prueba que se quiere ejecutar, ya que la persona encargada de ejecutarlos en este caso el probador Manual. No puede garantizar la cobertura sistemática de la funcionalidad del sistema [1]. Pero este método resulta muy costo, ya que para realizar la ejecución de la prueba es necesario que el probador ingrese las entradas al sistema y reproducir cada vez que se realice un cambio, por minúsculo que sea; para esto se generó un método llamado *Captura/Reproducción* de intentos, el cual graba la interacción del usuario y la simula cada vez que sea necesaria. Este método tiene el principal inconveniente de no poderse adaptar de manera trasparente a los cambios de la interfaz por ejemplo de realizarse un ajuste a un control de sistema, de una casilla de verificación a un botón de opciones es necesario realizar nuevamente el caso de prueba.

1.1.2. Proceso basados en scripts

Este tipo de proceso de prueba se basa en la generación de un script encargado de ejecutar uno o más casos de prueba y transmitirle al sistema bajo prueba los valores de entrada en espera de los valores de salida sean capturados por los puntos de control y verificar si las salidas del sistema son las esperadas. Esto implica que la generación de la prueba es una tarea de programación en un lenguaje de programación estándar o de pruebas especiales como Testing and Test Control Notation (TTCN-3). Pero el trabajo de mantener los script que ejecuten las pruebas se convierte en una tarea difícil de realizar, es por esta razón que nace el enfoque *Keyword-Driven Automated Testing Process*, el cual generan una tabla de pruebas asociada a una palabra clave. Con esto busca superar los problemas de mantenimiento y de abstracción de los casos de pruebas, utilizando script parametrizados por cada caso de prueba.

1.1.3. Pruebas basadas en modelos

Basadas en el diseño de casos de pruebas, bajo un modelo abstracto en donde el diseñador de la prueba genera los posibles escenarios por medio de un modelo abstracto. La principal ventaja de este tipo de prueba es generar una gran variedad de conjuntos de pruebas desde el mismo modelo a probar. Este tipo de pruebas se detalla en la monografía.

2. ANEXO B

2.1. HERRAMIENTAS PARA TRANSFORMACIONES

A continuación se exponen el estudio de herramientas para transformaciones:

2.1.1. Query/Views/Transformation (QVT)

Es la propuesta por la OMG[2] para mitigar el problema de la transformación de modelos, este se trata de un estándar para la definición de trasformaciones sobre modelos MOF[3]. Este estándar tiene tres abstracciones fundamentas:

- **Consultas (*Queries*):** es una consulta o expresión que se evalúa en el modelo. Es el mecanismo encargado de darle dinamismo a las transformaciones. Los resultados de las consultas son una o varias instancias de los tipos en el modelo transformado.
- **Vistas (*Views*):** es una vista del modelo obtenido en su totalidad a partir de otro modelo base.
- **Transformaciones (*Trasnformations*):** es una generación de un modelo a partir de otro modelo. Los modelos pueden ser dependientes o independientes, según exista una relación que los mantenga sincronizados.

2.1.2. ATLAS - Transformation Lenguage (ATL)

Esta es la propuesta realizada por el *Institut nacional de recherche en informatique et en automatique* (INRIA) en el 2002, en la actualidad este proyecto se enmarca en el proyecto *Generative Modeling Technologies* (GMT) de *Eclipse Software Fundation*, dentro de la rama de *Modeling Project*.

Esta propuesta define un lenguaje de transformación tanto por su metamodelo como por la sintaxis textual, de manera hibrida; la manipulación de modelos se basa en MOF [4].

2.1.3. Visual Automated model Trasnformation (VIATRA)

Es un framework de proyección general para dar sustento a todo el ciclo de vida de los procesos de transformación de modelos. Define un conjunto de especificaciones propias, al margen de las realizadas por la OMG. Además Tiene

la posibilidad de ejecutar transformaciones de modelo a modelo y de modelo a texto [5].

2.1.4. Motor de Transformaciones de BOA 2

Este motor es la evolución de framework BOA, esta propuesta presentada por la empresa *Open Canarias*, reforma el motor de transformación implementado por QVT[6], esta modificación se centra en las transformaciones de modelo a modelo de los modelos EMF [7], soportando características de trazabilidad, compilación incrementa además de transaccionalidad.

2.1.5. Motores de plantillas

Los motores de plantilla se han utilizado ampliamente en la generación de código en aplicaciones Web, para esta labores se han desarrollado varios motores de Scripting que permiten ejecutar del lado del servidor cierto código escrito en algún leguaje y después pasarlo a un motor o interprete que se encargará de generar documentos HTML para que el cliente los visualice en un navegador web.

Los principales motores de generación de este tipo son *Java Server Pages* (JSP) para *JEEE*; *Active Server Page* (ASP) para *ASP.NET* o *PHP*. En el entorno de *Modeling* podemos encontrar motores como el *Velocity* del proyecto *Apache Jakarta* y *JET* de *Eclipse Software Fundation* [8]. Los cuales sirven de soporte a otras herramientas de transformación descritas anteriormente.

3. ANEXO C

3.1. TIPOS DE ENFOQUES DE PRUEBAS DIRIGIDOS POR MODELOS

Una cuestión importante a la hora de aplicar *Model Based Testing* (MBT)[9] es definir su enfoque, ya que va de la mano con los requerimientos del proyecto de software a desarrollar y al cual se le aplicara esta técnica de pruebas. Lo anterior para escoger adecuadamente las tecnologías a utilizar y así poder estimar el esfuerzo y recursos necesarios para su exitoso despliegue.

Existen múltiples enfoques para aplicar MBT, los cuales dependen de los distintos campos que serán descritos a continuación:

- ***Dominio del software*:** define el alcance en el cual el enfoque MBT puede ser aplicado, el cual dependen del dominio del sistema bajo prueba (en sus siglas en inglés, SUT). Se revisan conceptos como el entorno de despliegue y ejecución, su paradigma de desarrollo por ejemplo: sistemas distribuidos, sistemas orientados a objetos, aplicaciones web, sistemas embebidos, entre otros.

- ***Nivel de automatización*:** trata sobre la cantidad de pasos que automáticamente son realizados por el conjunto de herramientas que se soportan el uso MBT. Se define la siguiente fórmula para medir el grado de automatización.

$$Grado\ Automatizacón = \frac{número\ \ de\ pasos\ automatizados}{número\ de\ pasos\ a\ realizar}$$

  Dicho grado de automatización se puede categorizar en tres niveles que son: alto el que más se aproxime a 1, medio el que este un rango promedio de la escala y bajo el más próximo a 0. Sin embargo este criterio puede ser relativo, ya que si uno de los pasos no automatizados conlleva un gran esfuerzo o gastos de recursos, no es posible asegurar que el nivel de automatización está absolutamente ligado ha dicho valor.

- ***Nivel de pruebas*:** se refiere al tipo de prueba que se llevaran a cabo, las cuales corresponden a las ya vistas en el apartado de pruebas de software, sección 1.1, que son: pruebas de sistema, de integración, unitarias o de componente y de regresión. Exceptuando las pruebas de aceptación porque estas requieren del factor humano para ser evaluadas.

- ***Herramientas de soporte*:** se buscan si existen herramientas que soporten el resto de criterios escogidos en la etapa de definición del enfoque a aplicar, ya que esto es un factor de riesgo para el proyecto software a desarrollar. Las herramientas deberían tener un buen nivel de automatización, que nos conlleve a la reducción de esfuerzo y costos.

- ***Comportamiento del modelo*:** representa las características del sistema que pueden ser probados por un enfoque MBT, a partir de los modelos del SUT que es donde se plasma la información abstraída del sistema. Se tienen tres aspectos a tener en cuenta respecto al modelo:

  - Facilidad y automatización de su desarrollo.
  - Completitud de información necesaria para la generación de pruebas
  - Facilidad de extracción de casos de prueba

- ***Modelos usados para la generación de los casos de prueba*:** se discierne entre las diferentes técnicas de modelamiento, como pueden ser las basadas en Lenguaje de Modelado Unificado (en sus siglas en inglés, *Unified Modeling Lenguage* UML)[10] y las que no. Dentro de las basadas en UML encontramos los modelos: diagramas de estados, actividades, colaboración, secuencia, clases, despliegue, entre otros. Y los que no son basados en UML, como los modelos de máquinas de estado finito (FSM), especificación Z, entre otros.

- ***Criterio de cubrimiento de pruebas*:** se definen aquí las reglas usadas para generar los casos de prueba, a partir del modelo de pruebas. Se tienen dos criterios que son: flujo de control y flujo de datos. Que definen el esfuerzo y calidad de los resultados obtenidos por el enfoque MBT.

- ***Criterio de generación de casos de prueba*:** presenta los pasos que requiere un enfoque MBT, para generar los casos de pruebas a partir del modelo de comportamiento o su ejecución.

- ***Técnicas de pruebas*:** se refiere a las diferentes técnicas que se utilizan tanto para la generación o ejecución de casos de pruebas, la generación de datos para las mismas, o validaciones sobre el modelo de pruebas. Como lo son: *"graph search algorithms", "random testing", "evolutionary testing", "constraint solving", "model checking", "static analysis", "abstract interpretation", "partition testing", y "slicing".*

4. ANEXO D

4.1. METAMODELO ECORE

En el presente proyecto de investigación se tomó como lenguaje abstracto para definir los metamodelos el lenguaje Ecore. Ecore es una poderosa herramienta para el diseño y la arquitectura por modelos (MDA), basado en EMOF [7], parte de la especificación MOF 2.0 [3], para la definición de modelos. El cual integra y representa diferentes dominios en la representación de modelos. Ecore utiliza cuatro clases denominadas a continuación:

- ***EClass:*** *representa una clase, con cero o más atributos y cero o más referencias.*
- ***EAttribute:*** *representa un atributo que tiene un nombre y un tipo.*
- ***Ereferencia:*** *representa un extremo de una asociación entre dos clases. Tiene bandera para indicar si representa una contención y una clase de referencia al que se apunta.*
- ***EDataType:*** *representa el tipo de un atributo, por ejemplo, int y float.*

Las anteriores clases son utilizadas, para representar clases, atributos, relaciones y datos primitivos respectivamente y su relación semántica dentro de Ecore. En la Figura 1, se representa el conjunto simplificado de Metamodelo Ecore, extraído desde [7].

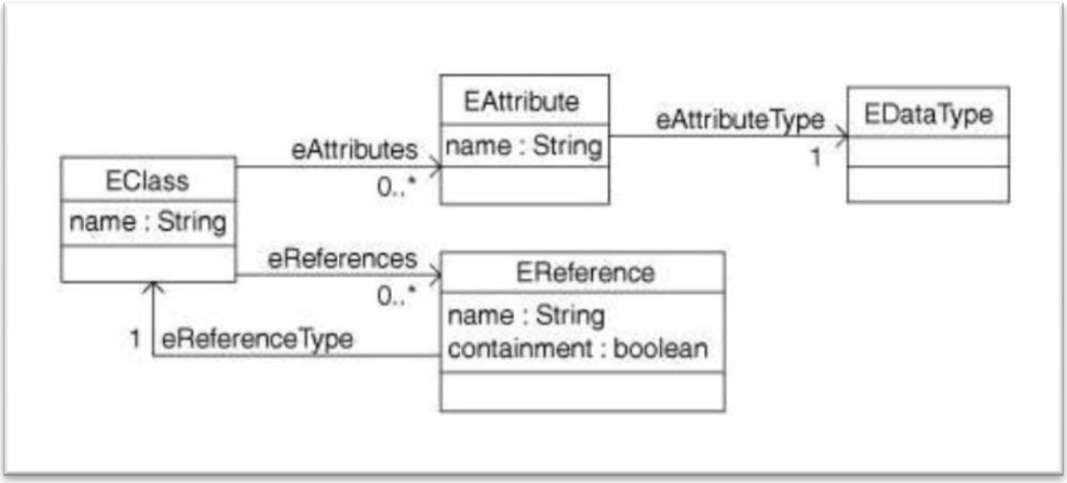


Figura 1 - Subconjunto Simplificado del Metamodelo ECORE

5. ANEXO E

5.1. COMPARACIÓN HERRAMIENTAS DE TRANSFORMACIÓN DE MODELO A TEXTO

Las transformaciones de modelo a texto son una parte importante en los enfoque de desarrollo software dirigidos por modelos. Esta tecnología permite tener artefactos capaces de traducir automáticamente modelos en formatos textuales. En la actualidad existen muchas tecnologías para realizar transformaciones de modelo a texto, dentro de la presente investigación se han analizado un conjunto de lenguajes y herramientas para la transformación de modelo a texto, las cuales se presentaran a continuación.

5.1.1. Acceleo

Esta herramienta es un generador de modelos a textos a través del enfoque *Model Driven Architecture* (MDA), desarrollada inicialmente por la empresa francesa *Obeo*. Esta herramienta implementa la especificación del lenguaje estándar de transformación de modelo a texto desarrollado por la *OMG* [2], denominado *MOFM2T*. Esta herramienta se puede utilizar por medio de un *plugin* de eclipse con Licencia Pública Eclipse (EPL), desarrollado bajo el proyecto de *Eclipse Modeling Framework*[8].

Algunas de las características más importantes de esta herramienta son las siguientes:

- Generación de código por medio de cualquier tipo de metamodelo compatible con *EMF* como *UML1*, *UML2* e incluso metamodelos adaptados a un dominio especifico.
- Permite realizar la generación de código por medio de plantillas modulares, en el idioma estándar *Model Transformation Language* (*MTL)*.
- Permite realizar llamadas al sistema y librerías externas a través de Java.
- Pone a disposición un editor con características de: código resaltado de sintaxis, autocompletado de código, detección de errores, refactorización entre otras.
-

5.1.2. MOFScript

Es una herramienta para la transformación de modelo a texto, basado en reglas y presentado como candidato para ser el lenguaje estándar en la *OMG*. Tiene una licencia de tipo *EPL* y se puede instalar como *plugin* para *Eclipse* teniendo también la posibilidad de ejecutarla como una aplicación Java independiente.

Algunas características de esta herramienta se nombran a continuación:

- Se puede especificar mecanismos de control básicos como bucles y sentencias.
- Puede manipular cadenas de texto.
- Editor con autocompletado, resaltado de sintaxis y detección en vivo de errores de compilación.

La principal desventaja de esta herramienta es que no utiliza un lenguaje estandarizado para realizar las transformaciones y tiene una regular documentación.

5.1.3. Xpand

Es un lenguaje de trasformación de modelos a texto, que se puede instalar como plugin de Eclipse con licencia EPL. Se base en plantillas que utiliza variables para la generación de texto.

Dentro de lasplantillas se pueden utilizar: variables, bucles y sentencias de control. Además permite invocar funciones construidas en lenguaje Java. Este lenguaje se distribuye con una herramienta llamada *Xtest* destinada para la creación de lenguaje textuales de dominio específicos.

La principal desventaja de esta herramienta es que no utiliza un lenguaje estandarizado para realizar las transformaciones y tiene una regular documentación.

5.1.4. JET

Denominada Java Emitter Template (JET), es un proyecto desarrollado bajo el proyecto de Eclipse Modeling Framework[8], centrado en el proceso de transformación de modelo a texto basado en Meta-Object Facility (*MOF)*. JET funciona a partir de una serie de plantillas que son transformadas a clases Java para posteriormente ser ejecutadas. Una ventaja de esta herramienta frente a las anteriormente analizadas es la posibilidad de incluir directamente en ellas código Java.

La principal desventaja de esta herramienta es que no utiliza un lenguaje estandarizado para realizar las transformaciones y tiene una regular documentación.

6. ANEXO F

6.1. Plantillas de transformación en Acceleo

A continuación se describen en plantillas de alto nivel, los requisitos que debe cumplir en cada una de las plantilla de transformación.

Nombre del archivo:	generate.mtl
Paquete:	prudima.model2java.main
Regla de transformación:	Dado un modelo de UML que contenga diagramas de clases y de secuencia, invocara las plantillas específicas para cada elemento del modelo.

Nombre del archivo:	agentCaseTestJavaFile.mtl
Paquete:	prudima.model2java.files.prudimaimpl
Regla de transformación:	Dado un elemento *Interaction* de UML el cual tenga aplicado el estereotipo <<Test>> se debe crear un archivo .java el cual contendrá el código que permita ejecutar una prueba unitaria o funcional.

Nombre del archivo:	agentJadeJavaFile.mtl
Paquete:	prudima.model2java.files.prudimaimpl
Regla de transformación:	Dado un elemento *Lifeline de UML* que tenga aplicado el estereotipo <<MockClient>> se debe crear un archivo .java que contendrá el código que represente un agente software.

Nombre del archivo:	agentManagerJavaFile.mtl
Paquete:	prudima.model2java.files.prudimaimpl
Regla de transformación:	Dado un elemento *Interface* de UML que tenga aplicado el estereotipo <<RemoteInterface>> se debe crear un archivo .java que contendrá el código que permita realizar las invocaciones remotas definidas por los elementos de tipo *Operation* de UML que tengan aplicados el estereotipo <<RemoteService>>, los cuales están contenidos dentro de la Interface.

Nombre del archivo:	excBatFile.mtl
Paquete:	Model2java.acceleo.main
Regla de transformación:	Dado un modelo de UML se debe crearun archivo con extensión .bat el cual contengan la invocación al SUT.

Nombre del archivo:	prudimaMainJavaFile.mtl
Paquete:	prudima.model2java.files.prudimaimpl
Regla de transformación:	Dado un *modelo de UML* se debe crear un archivo .java que contenga el código que invoque la parte estática del sistema PRUDIMA definido por la propiedad nombre del modelo.

Nombre del archivo:	testControllerJavaFile.mtl
Paquete:	prudima.model2java.files.prudimaimpl
Regla de transformación:	Dado un *modelo de UML* se debe crear un archivo .java con el código que permite invocar cada prueba unitaria o funcional.

Nombre del archivo:	testManagerJavaFile.mtl
Paquete:	prudima.model2java.files.prudimaimpl
Regla de transformación:	Dado un *modelo de UML* se debe crear un archivo .java con el código que permita desplegar la plataforma de agentes y las pruebas.

Nombre del archivo:	utilitiesRMIJavaFile.mtl
Paquete:	prudima.model2java.files.prudimaimpl

Regla de transformación:	Dado un elemento *Interface de UML* se debe crear un archivo .java con el código que permita invocar localizar un servicio remoto.

Nombre del archivo:	properties.mtl
Paquete:	prudima.model2java.properties
Regla de transformación:	No realiza ninguna transformación, únicamente contiene cadenas constantes que son utilizadas por el resto de plantillas.

Nombre del archivo:	fileUtils.mtl
Paquete:	prudima.model2java.common
Regla de transformación:	Dado un modelo de UML permite obtener valores que se genera en cadenas de texto que son utilizadas por el resto de plantillas.

Nombre del archivo:	xmlGenerator.mtl
Paquete:	prudima.model2java.common
Regla de transformación:	Dado un modelo de UML se debe generar un archivo .xml, donde se realiza un mapeo del modelo en UML ha XML.

7. ANEXO G

7.1. ESPECIFICACIÓN DE REQUERIMIENTOS DE SOFTWARE (SRS)

7.1.1. Introducción

El proceso de construcción del prototipo de herramienta PRUDIMA pretende soportar el proceso de MBT para pruebas a sistemas distribuidos (SD), denominado P-MBT-SD. El prototipo de la herramienta está integrado por varios componentes los cuales logran llevar a cabo el diseño, despliegue, ejecución y reporte de pruebas para un sistema distribuido basado en el modelo cliente-servidor.

En las secciones siguientes se describirán los componentes, sus relaciones y todo el proceso de construcción del mismo.

7.1.2. Propósito

El propósito de este SRS es describir los artefactos que se produjeron en las diferentes iteraciones de desarrollo del prototipo. Además, de que las personas involucradas en el área de construcción de software comprendan cómo fue concebido y diseñado el sistema.

7.1.3. Alcance

El software a desarrollar es un prototipo de herramienta CASE, al cual se le dio el nombre de PRUDIMA, con soporte al procedimiento de MBT a SD que fue definido en este trabajo. Esta herramienta permite modelar y desplegar pruebas para sistemas distribuidos construidos con tecnología RMI, utilizando modelos soportados en diagramas de clase y secuencia de UML, a los cuales se les aplicará el perfil UML de PRUDIMA, también definido en este trabajo.

Le herramienta pretende disminuir el consumo de recursos de tiempo del docente encargado evaluar las implementaciones de las prácticas de sistemas distribuidos construidos con tecnología RMI, que son desarrollados por parte de los estudiantes de la asignatura de laboratorio de sistemas distribuidos de la Universidad del Cauca. Al mismo tiempo, poder brindar al estudiante de una herramienta que permita evaluar de manera temprana su sistema en desarrollo.

7.1.4. Resumen

El resto del siguiente documento contiene en detalle las descripciones de las diferentes funcionalidades del sistema. En la sección7.1.8.1, la descripción del sistema a nivel macro, como las interfaces gráficas para los usuarios finales. En la sección 7.1.8, se encuentra las especificaciones de los requerimientos funcionales.

7.1.5. Descripción General

A continuación se realiza la descripción general del software.

7.1.5.1. Perspectiva del producto

El prototipo software debe contar con una interfaz de usuario amigable que brinde las funcionalidades expuestas dentro de los requerimientos, por medio de formularios y botones amigables.

En la sección 7.1.8.1 se muestra como seria la interfaz del prototipo.

7.1.5.2. Funciones del producto

Aquí se presenta la lista de funcionalidades que realizará el prototipo a implementar:

- Proveer un modelador de diagramas de clases y UML para el modelado de las pruebas.
- Generación del código a partir de modelos.
- Construir un motor de ejecución de las pruebas.
- Gestión de los datos de pruebas.
- Generar el reporte de las pruebas ejecutadas.

7.1.6. Características De Usuario

En esta sección se describen los tipos de actores que interactuaran con el sistema ejecutando las distintas funcionalidades.

Actor	Modelador
Descripción	Un usuario modelador (docente ó estudiante) podrá generar los modelos de las pruebas de acuerdo a los requerimientos a evaluar, teniendo en cuenta las limitaciones del sistema en cuanto a expresividad y alcance expresados este documento.

Tabla 1- *Descripción usuario Modelador*

Actor	Tester
Descripción	Este usuario, podrá ejecutar las pruebas generadas por el usuario modelador y realizar la carga de los datos de los diferentes casos de pruebas, con los datos que se consideren necesarios para probar la aplicación.

Tabla 2 - *Descripción usuario Tester*

Actor	Analista
Descripción	El usuario Analista (docente ó estudiante) es el encargado de analizar el informe de pruebas ejecutadas contra el sistema bajo prueba, entregado por el Tester. Cuyo análisis, comprende el cubrimiento de las pruebas y la correlación con los requerimientos del sistema.

Tabla 3 – *Descripción usuario Analista*

7.1.7. Restricciones generales

El prototipo de la herramienta PRUDIMA, solo soporta algunos de los elementos de la especificación del lenguaje de modelado UML. A continuación se listan los elementos soportados, aclarando que los elementos no presentes en la lista no se soportarán.

Elementos de UML soportados: *Model, Interaction, Class, Association, Interface, Operation, Parameter, Message, Lifeline, Signature*, *Generalization*, *Implementation*.

7.1.8. ESPECIFICACION DE REQUERIMIENTOS

Id. de requisito	R-01		
Nombre	Generar Prueba		
Fuente	Proceso MBT-SD		
Prioridad	☒ Alta/Esencial	☐ Media/Deseado	☐ Baja/Opcional
Descripción	El prototipo debe ofrecer un mecanismo para ayudar a modelar la prueba por medio de lenguaje de modelado UML [11], esto se deberá realizar por medio de un editor gráfico. Además de permitir la aplicación del perfil UML de PRUDIMA.		
Prerrequisito	Ninguno		
Manejo de errores	E1: Error al ingresar la prueba. El sistema deberá validar la sintaxis del modelo generado, en caso de que exista algún problema el sistema lanzará un mensaje de error.		

Tabla 4 - *Requerimiento Generar prueba*

Id. de requisito	R-02		
Nombre	Validación del modelo		
Fuente	Proceso MBT-SD		
Prioridad	☒ Alta/Esencial	☐ Media/Deseado	☐ Baja/Opcional
Descripción	Debe existir un mecanismo de validación, encargado de evaluar el modelo ingresado, donde éste deberá verificar la sintaxis empleada y la aplicación de los perfiles adecuados a los componentes del modelo.		
Prerrequisito	Ninguno		
Manejo de errores	E1: Error al ingresar el modelo. El sistema deberá notificar que el modelo no ha sido ingresado para realizar su validación		

Tabla 5 - *Requerimiento Validación del modelo*

Id. de requisito	R-03		
Nombre	Generación de la aplicación para pruebas al SUT		
Fuente	Proceso MBT-SD		
Prioridad	☒ Alta/Esencial	☐ Media/Deseado	☐ Baja/Opcional
Descripción	El prototipo deberá generar el código fuente de una aplicación encargada de ejecutar las diferentes pruebas definidas en el modelo.		
Prerrequisito	Modelo a generar		

Manejo de errores	E1: No se puede generar el código por problemas en el modelo. El prototipo deberá desplegar el mensaje de error, por problemas con el modelo ingresado. E2: Problemas al generar el código. De encontrarse un error la aplicación deberá informar del problema por medio de un mensaje de error.

Tabla 6 - *Requerimiento Generación del código a probar*

Id. de requisito	R-04		
Nombre	Generar Caso de prueba		
Fuente	Proceso MBT-SD		
Prioridad	☒ Alta/Esencial	☐ Media/Deseado	☐ Baja/Opcional
Descripción	La aplicación deberá gestionar los casos de pruebas ingresados en el modelo, permitiendo la funcionalidad de agregar y eliminar los casos de uso que el usuario desea evaluar.		
Prerrequisito	Modelo a generar		
Manejo de errores	E1: Problemas al agregar un caso de prueba. El prototipo deberá informar al usuario de dicho problema y de ser posible presentar un informe con la razón. E2: Problemas al eliminar un caso de prueba. El prototipo deberá informar al usuario de dicho inconveniente y de ser posible presentar un informe con los detallesla razón.		

Tabla 7 - *Requerimiento Generar Caso de prueba*

Id. de requisito	R-05		
Nombre	Carga de datos de prueba		
Fuente	Proceso MBT-SD		
Prioridad	☒ Alta/Esencial	☐ Media/Deseado	☐ Baja/Opcional
Descripción	El prototipo deberá permitir la gestión de los datos que se desean probar a partir de los modelos de pruebas ingresados. Esta gestión debe realizarse con la ayuda de una interfaz gráfica.		
Prerrequisito	Modelo a generar y casos de pruebas		
Manejo de errores	E1: No se puede ingresar el valor. El prototipo deberá informar que existe un problema para ingresar dicho valor.		

Tabla 8 - *Carga de datos*

Id. de requisito	R-06		
Nombre	Ejecutar las pruebas		
Fuente	Entrevista con el usuario final		
Prioridad	☒ Alta/Esencial	☐ Media/Deseado	☐ Baja/Opcional

Descripción	El usuario deberá tener la funcionalidad de ejecutar los casos de pruebas desde la interfaz gráfica de la aplicación.
Prerrequisito	Modelo generado, casos de pruebas generados y datos cargados.
Manejo de errores	E1: No se puede ejecutar las pruebas. El prototipo deberá informar que no se pudieron ejecutar las pruebas.

Tabla 9 - *Requerimiento Ejecutar la prueba*

Id. de requisito	R-07		
Nombre	Visualizar el reporte		
Fuente	Proceso MBT-SD		
Prioridad	☒ Alta/Esencial	☐ Media/Deseado	☐ Baja/Opcional
Descripción	El usuario tendrá la opción de visualizar el resultado de las pruebas por medio de la interfaz gráfica del prototipo. En ella deberá expresar que pruebas fueron exitosas y cuáles no.		
Prerrequisito	Pruebas ejecutadas		
Manejo de errores	E1: Problemas al cargar el reporte, el reporte no puede ser visualizado. El prototipo deberá informar al usuario con un mensaje de error que no se pudo cargar el reporte.		

Tabla 10 - *Requerimiento Visualizar el reporte*

7.1.8.1. Requerimientos de interfaz externa

En las figuras **Figura 2**, **Figura 3** y **Figura 4** presentan algunos *mockups* diseñados tener una idea inicial de cómo los actores interactuarán con el sistema.

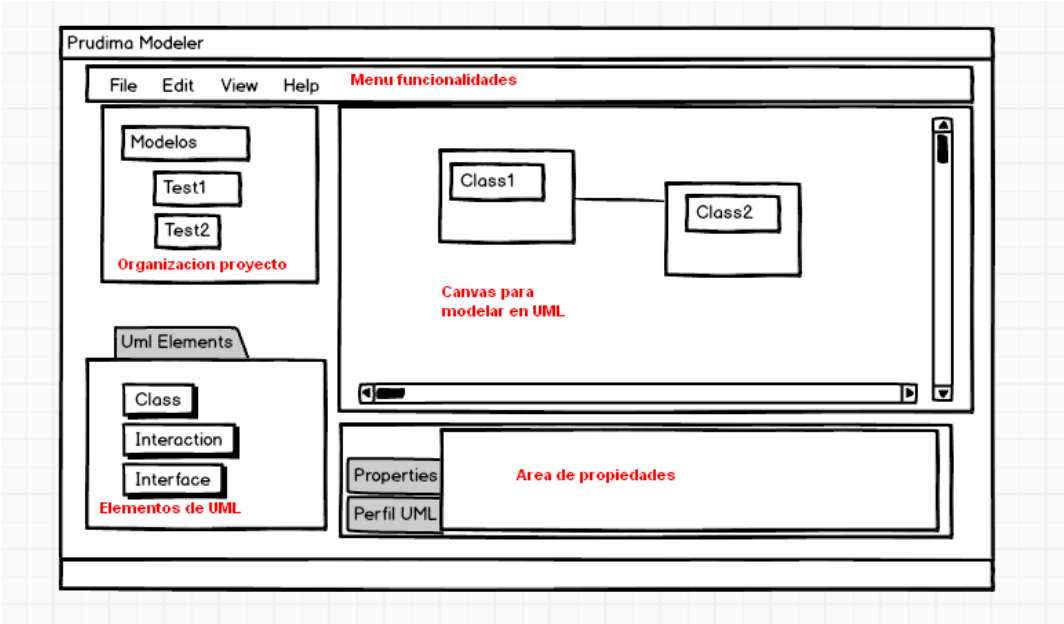


Figura 2 - *GUI para modelar pruebas*

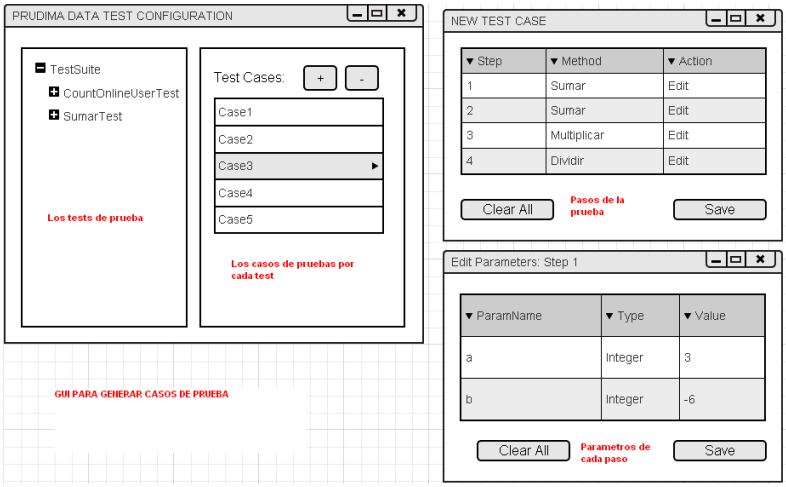


Figura 3 - *GUI para crear casos de prueba*

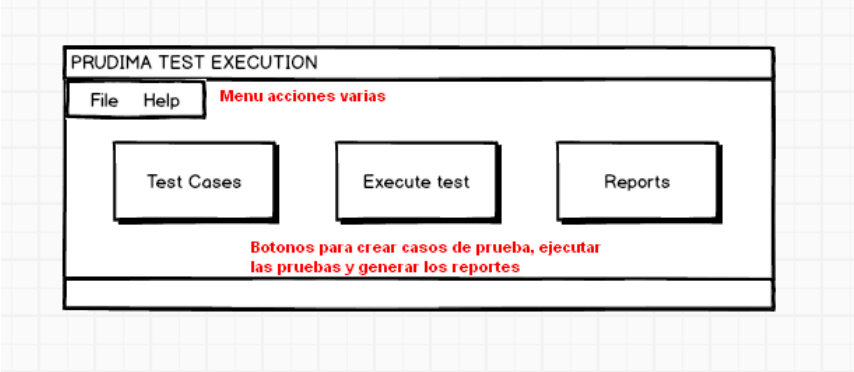


Figura 4 - *GUI para proceso de testing*

7.1.8.2. Interfaces de comunicación

Comunicación a través de tecnología RMI, utilizando la plataforma de agentes JADE.

7.1.8.3. Requerimientos funcionales

A continuación se presenta la lista de requerimientos funcionales, que deberá cumplir la herramienta a implementar:

- Generar una aplicación que permita modelar las pruebas al SUT por medio de UML y el perfil UML de PRUDIMA.
- Validar el modelo creado
- Generar los casos de prueba.
- Cargar los datos de pruebas.
- Ejecutar las pruebas de funcionalidad y de unidad al SUT.
- Generar los reportes de las pruebas ejecutadas.

7.1.8.4. Casos de uso

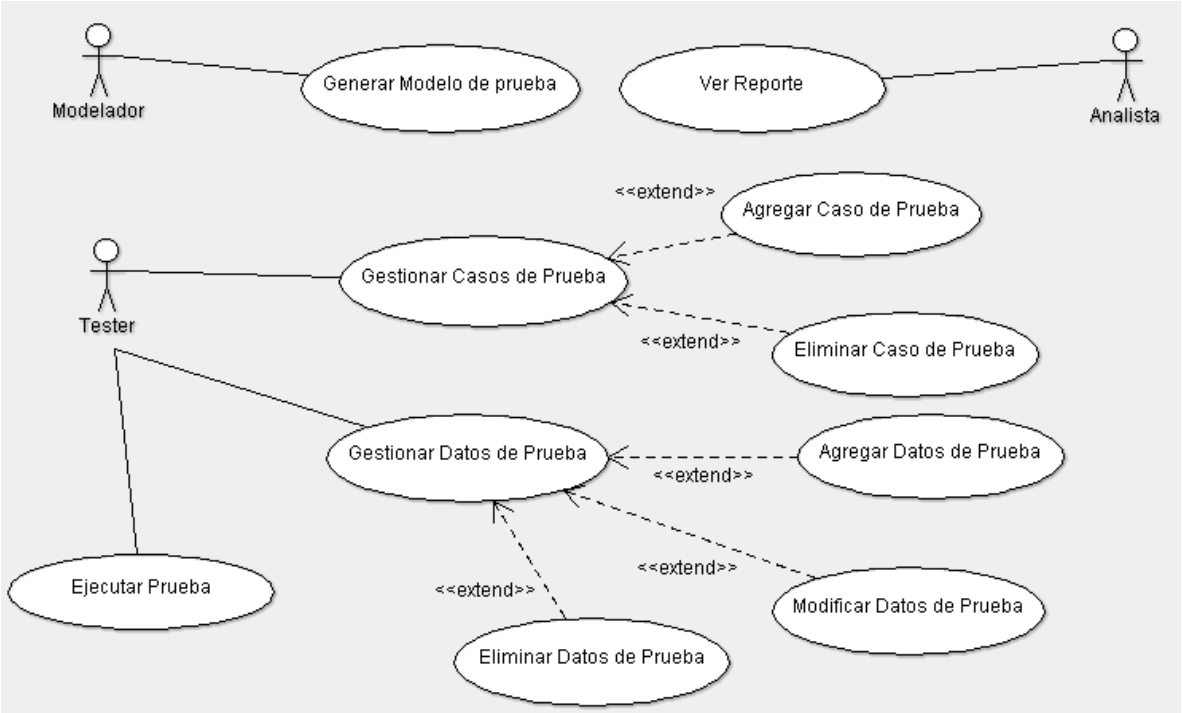


Figura 5 - *Diagrama de casos de uso de PRUDIMA*

En esta sección se presentan el diagrama de casos de uso **Figura 5** y algunas vistas del diagrama de clases global de la solución de PRUDIMA, el cual, no se puede mostrar completamente debido a la restricción de espacio en el documento. Por lo tanto, se mostraran las clases más relevantes de cada componente del sistema.

A continuación se presentan los diagramas de clases de la solución de PRUDIMA en su parte estática (PE), Figura 6 y Figura 7; además del diagrama de agentes Figura 8:

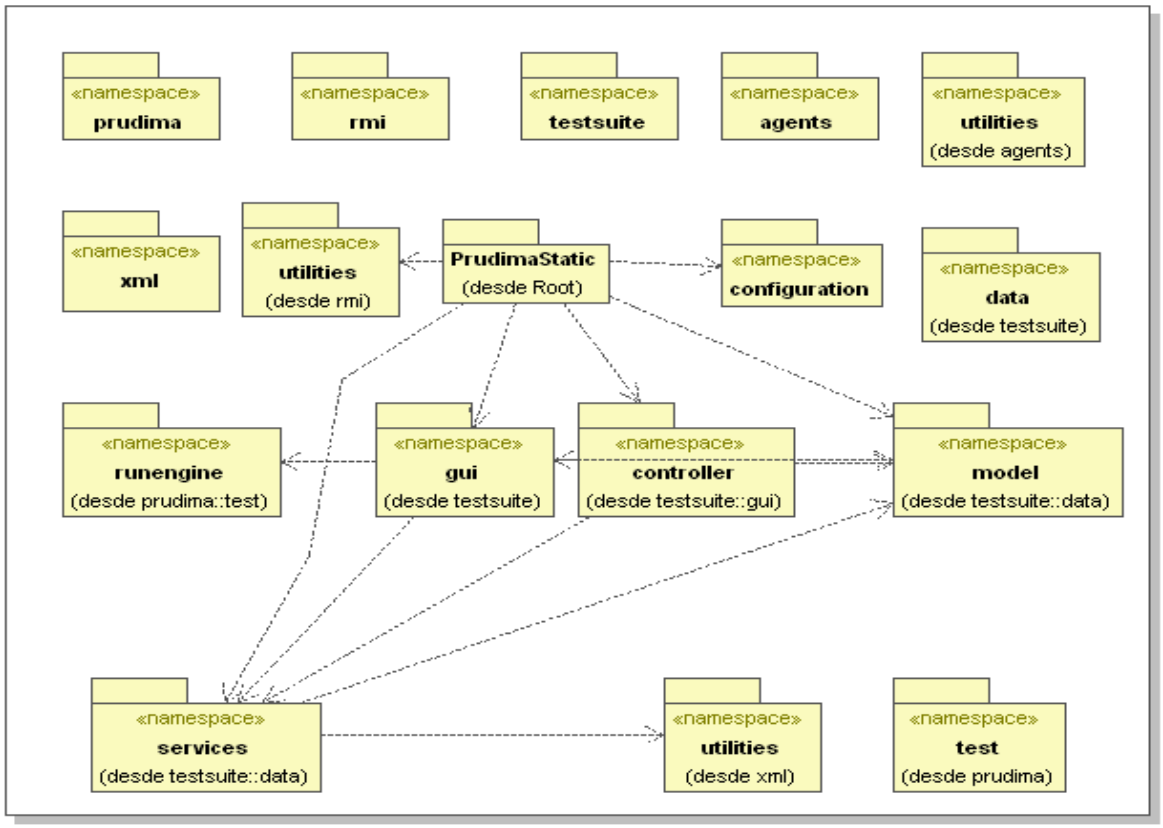


Figura 6 - *Diagrama de paquetes PE*

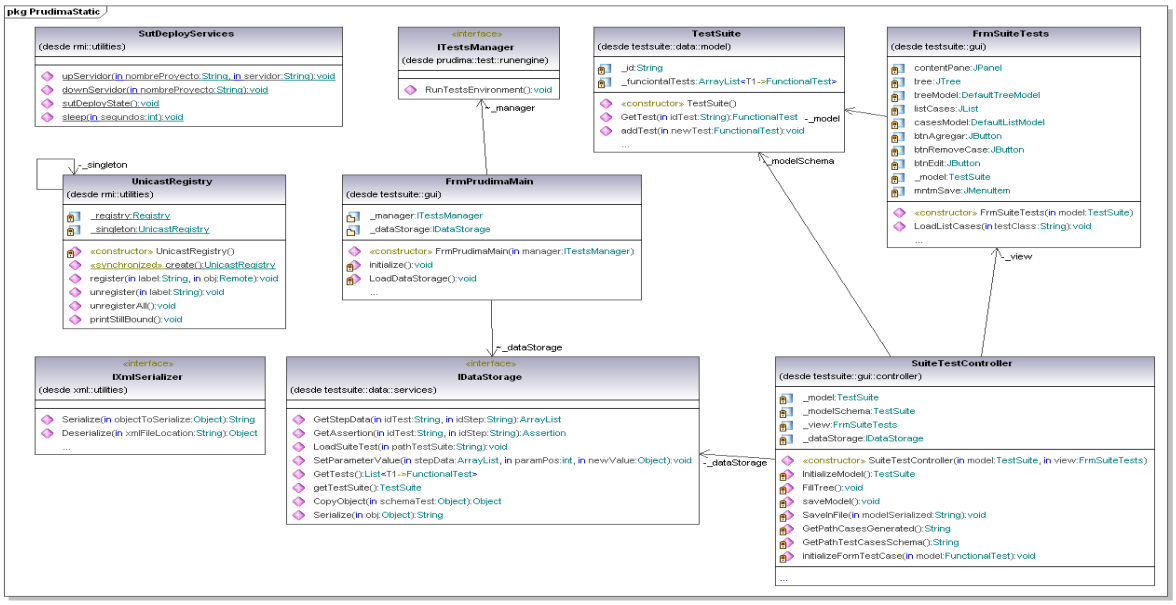


Figura 7 – *PE – Despliegue y manejo persistencia*

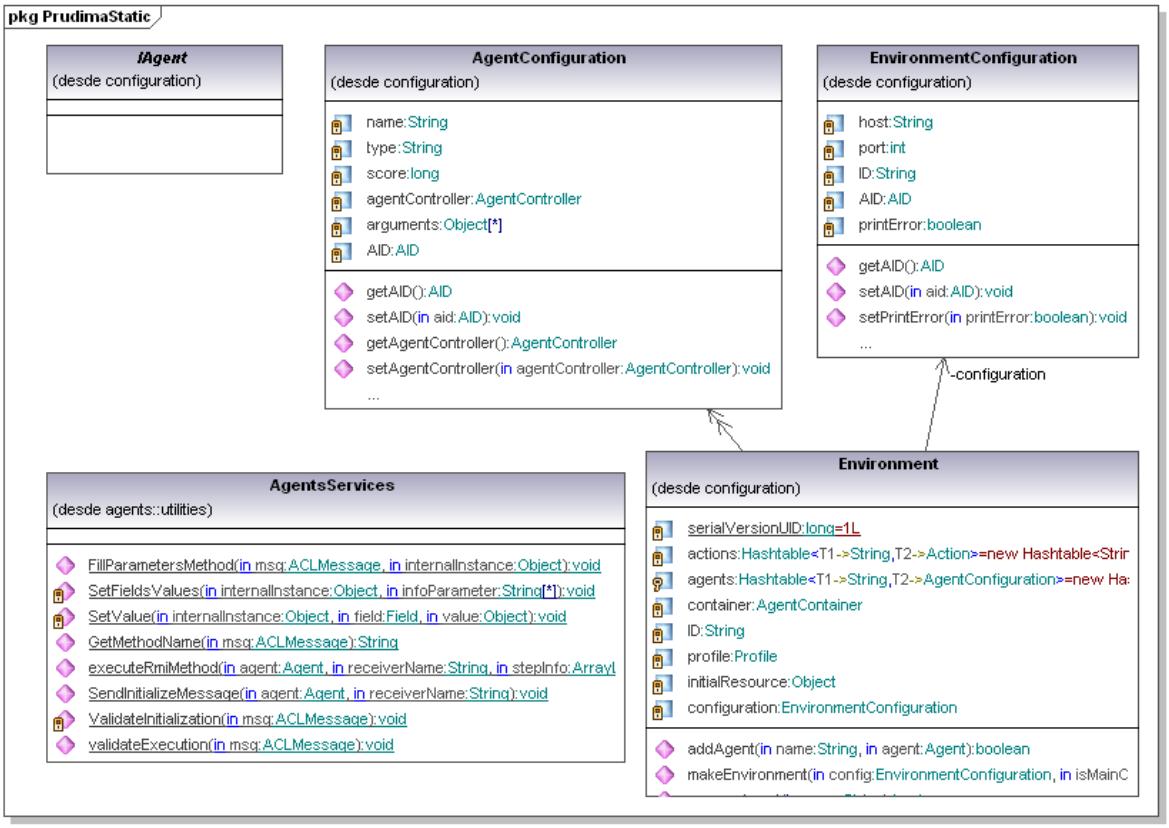


Figura 8 - *PE Plataforma de agentes*

7.1.8.5. Requerimientos no funcionales

- **Fiabilidad:** el prototipo debe ser tolerante a fallos, siempre y cuando los componentes hardware y las condiciones sean las óptimas, para realizar y ejecutar las pruebas al SUT.
- **Disponibilidad:** el sistema debe tener un alto grado de disponibilidad en el momento de cargar los datos a probar, implementar y ejecutar las pruebas.
- **Escalabilidad:** el prototipo debe ser construido de manera evolutiva e incremental, de tal manera que se puedan agregar nuevas funcionalidades y que algunos nuevos requerimientos relacionados puedan ser incorporados, de manera que se afecte lo menos posible el código existente.
- **Integración con el IDE de Eclipse:** el prototipo debe tener integración con el IDE del entorno de desarrollo Eclipse, para facilitar la generación y ejecución de las pruebas.
- **Instalación:** el sistema debe ser independiente de la plataforma, además de instalarse de manera manual.
- **Mantenimiento:** el prototipo debe estar debidamente documentado.

8. ANEXO H

8.1. CRITERIOS PARA ESCOGER LA HERRAMIENTA DE MODELADO DE UML

A continuación se listan los criterios presentados en la Tabla 11 - Comparación entre herramientas de modelado que serán tenidos en cuenta para la selección de la herramienta que apoyara el proceso de modelado.

Tipo de licencia: este criterio nos permite identificar el tipo de licencia que tiene la HM-UML escogida.

Integración con el IDE de Eclipse: permite identificar si la herramienta modeladora cuenta con la integración con el IDE de desarrollo.
Visualización gráfica de los modelos: cuenta con la opción de visualizar gráficamente los modelos construidos.

Soporte para uso de perfiles UML: la herramienta soporta el uso de perfiles UML construidos en el meta-metamodelo *Ecore*, haciendo posible la extensión de los elementos de UML.

Soporte para diagramas de secuencia y clases de UML: la herramienta cuenta con el soporte para modelar diagramas de clases y de secuencia de UML.

Plataformas disponibles: nos indica en que plataformas o sistema operativo se puede instalar la herramienta para modelar.

Nombre herramienta	Tipo de licencia	Integración con el IDE de Eclipse	Visualización gráfica de los modelos	Soporte para uso de perfiles UML	Soporte para diagramas de secuencia y clases de UML	Plataformas disponibles
Papyrus	**SI - EPL**	**SI**	**SI**	**SI**	**SI**	**Todas donde esté instalado Java**
Modelador Nativo Eclipse	SI - EPL	SI	NO	SI	SI	Todas donde esté instalado Java
Umbrello UML	SI - GPL	NO	SI	No	SI	GNU/Linux, Windows con KDEWin y Mac OS X
Poseidon UML	NO - Licencia de distribuidores	SI	SI	Si	SI	Linux, Windows, MacOS X

Rational Rose	NO – Licencia de distribuidores	NO	SI	Si	SI	Windows
Power Designer	NO – Licencia de distribuidores	NO	SI	Si	SI	Windows

Tabla 11 - Comparación entre herramientas de modelado

Para la selección de la herramienta que apoyará el proceso de modelado se buscó que esta tuviese un tipo de licencia de software libre, además de poder integrarse adecuadamente al IDE de desarrollo de Eclipse, ya que el uso de un modelador externo, acarrea una carga cognitiva adicional. También contara con un modelador gráfico y que permita el manejo de perfiles. Dada las anteriores consideraciones se tomó la decisión de utilizar Papyrus. Esto debido a que cumple con todas las consideraciones.

9. ANEXO I

9.1. GUIA DE CASO DE ESTUDIO DE PRUDIMA PRUEBAS ENFOQUE MBT - MODELADO

Introducción

El paradigma de pruebas dirigidas por modelos permite la generación automática de casos de pruebas a partir de modelos construidos en las fases de desarrollo de software. Siguiendo el enfoque Model Driven Architecture (MDA), se utilizan un conjunto de transformaciones a los modelos, donde finalmente se obtiene un código ejecutable que permite probar el sistema en desarrollo. Con MBT se pueden realizar diferentes tipos de pruebas tales como las pruebas unitarias, de sistema, de regresión, de aceptación, entre otras.

En esta práctica nos enfocaremos en el diseño de pruebas del tipo *unitarias* y de *sistema.*

Objetivo

El objetivo de este caso de estudio es verificar el procedimiento de MBT definido en este trabajo, a través de la comparación entre la aplicación de las pruebas dirigidas por modelos y el uso del enfoque tradicional de pruebas manual en el contexto académico del laboratorio de sistemas distribuidos de la Universidad del Cauca.

Sistema a probar

El caso de estudio analiza un sistema denominado *Sistema de elecciones.* El sistema se escoge porque fue utilizado el cual fue implementado por los autores del presente trabajo de grado y se basa en el caso de estudio del trabajo de

investigación de Lopez Hoyos [1][2], para el curso de laboratorios de sistemas distribuidos, del cual, se obtuvieron los requerimientos y la documentación necesarios a nivel de diseño y técnicos . En este sistema se utiliza la tecnología de comunicación de invocación remota (*Remote Method Invocation* - RMI) y se provee unos servicios que permiten llevar a cabo las votaciones por parte de los *Votantes* y consultar los resultados de las votaciones por parte de los *Periodistas*.

En este sistema existen 2 tipos de usuarios, cada uno de ellos puede ejecutar diferentes funcionalidades descritas a continuación:

1. ***Votante***:
   - Puede validar su código contra el sistema.
   - Puede ingresar el voto por un candidato dado su código.

2. ***Periodista***:
   - Puede validar su código contra el sistema.
   - Puede consultar la votación de un candidato particular dado su código.

A continuación en la **Figura 9**, se presenta un diagrama de casos de uso del sistema:

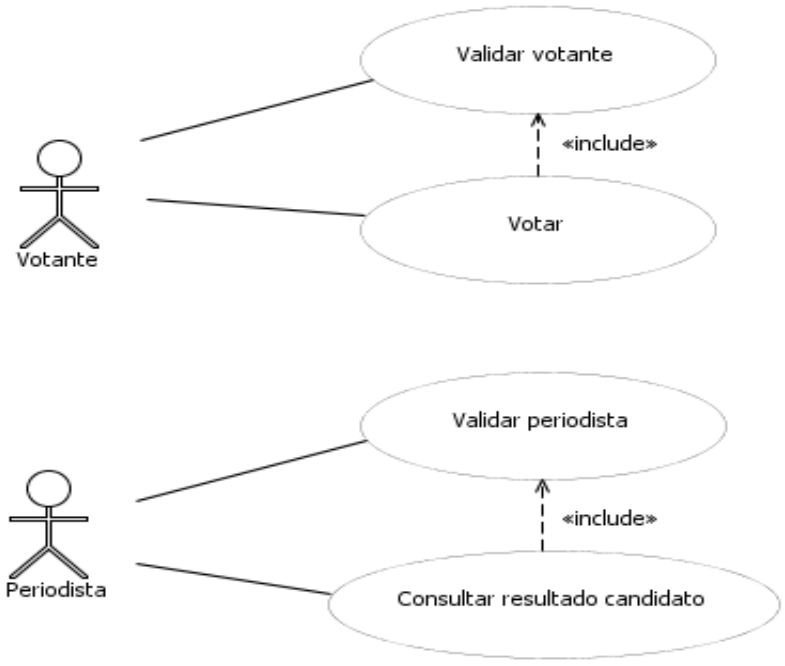


Figura 9 *- Diagrama de casos de uso Sistema de elecciones*

Requerimientos del sistema a probar

En la siguiente tabla se presentan los requerimientos que tendrán cobertura en las pruebas:

Id	Descripción del requerimiento	Tipo de Prueba
1	El sistema debe validar el código del votante. El sistema retornara el valor *true* si el **código** del votante es validado correctamente, de lo contrario el sistema retornará el valor *false*. Los códigos de los votantes van de 300 a 1000.	Unitaria
2	Se debe permitir a un votante ya validado en el sistema, **ingresar el voto** por un candidato. Esto se realizara enviando al sistema el código del candidato a votar. Para los códigos validos ver tabla 2. Los votantes no pueden repetir el derecho al voto, por lo tanto si es la primera vez que vota el sistema retornará el valor true, en las siguientes veces que intente votar, el sistema retornará *false.*	Sistema
3	Para que un periodista pueda consultar la información contenida en el sistema, deberá ser validado. El sistema retornara el valor *true* si el código del periodista es validado correctamente, de lo contrario el sistema retornará el valor *false*. Los códigos de los periodistas están dentro del rango de :200-299	Unitaria
4	A los periodistas ya validados en el sistema, se les permite consultar la votación de un candidato particular. Esto se realizara enviando al sistema el código del candidato a votar. Para los códigos validos ver tabla 2.	Sistema

Tabla 12 - *Lista de requerimientos*

Para los requerimientos 2 y 4, los códigos válidos para los candidatos son:

Nombre	***Código***
Campo L	107
Zambrano S	109

López J	111
Castro F	113
Ortega T	115
Vernaza R	117

Tabla 13 *- Códigos de candidatos*

GUIA PRÁCTICA DE UTILIZACIÓN DE HERRAMIENTA PRUDIMA

Con el fin de cumplir con el tiempo definido para el caso de estudio de modelado, se realizó esta guía práctica de utilización del prototipo de la herramienta Prudima que permite abordar el proceso de MBT. Esta guía permitirá conocer las diferentes funcionalidades del sistema que debemos tener en cuenta cuando se realice formalmente el caso de estudio.

Pre-requisitos

Para realizar esta práctica se debe tener el siguiente entorno de trabajo, con los plugins nombrados a continuación:

- Eclipse Modeling Tools
  Versión: Juno Service Release 2
  Build id: 20130225-0426
    - Módulos pre-instalados
        - Papyrus
        - Acceleo

Desarrollo de la práctica

1. **Inicio de IDE de trabajo**

Para el desarrollo de la práctica debemos iniciar el entorno de trabajo Eclipse Juno, para esto ubicamos el archivo “eclipse.exe”, como se presenta en la Figura 10.

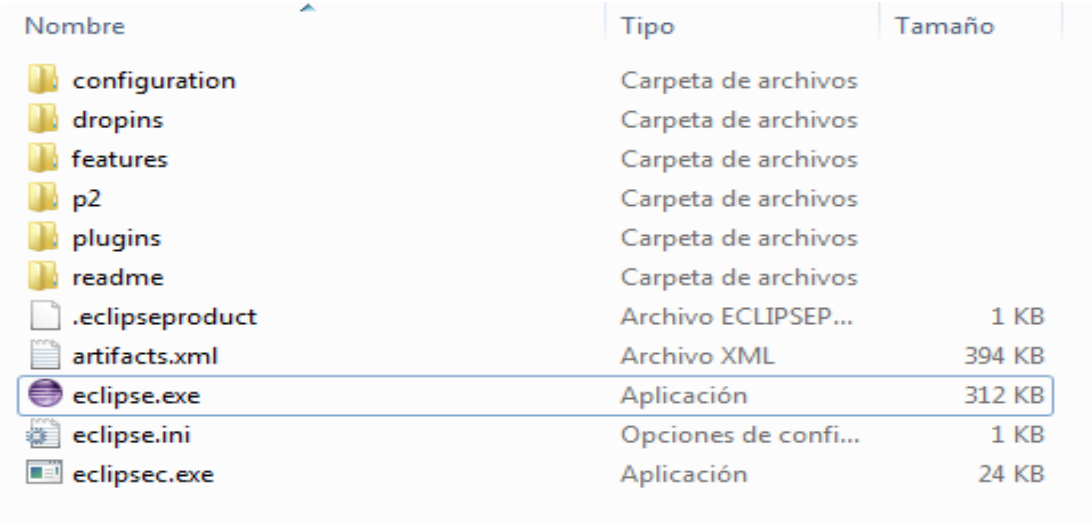


Figura 10 Archivo para iniciar Eclipse Juno

Una vez iniciado el Eclipse, debemos seleccionar el espacio de trabajo, para esto se nos presentara una ventana como en la Figura 11, en donde podremos configurar la carpeta que deseamos como entrono de trabajo.

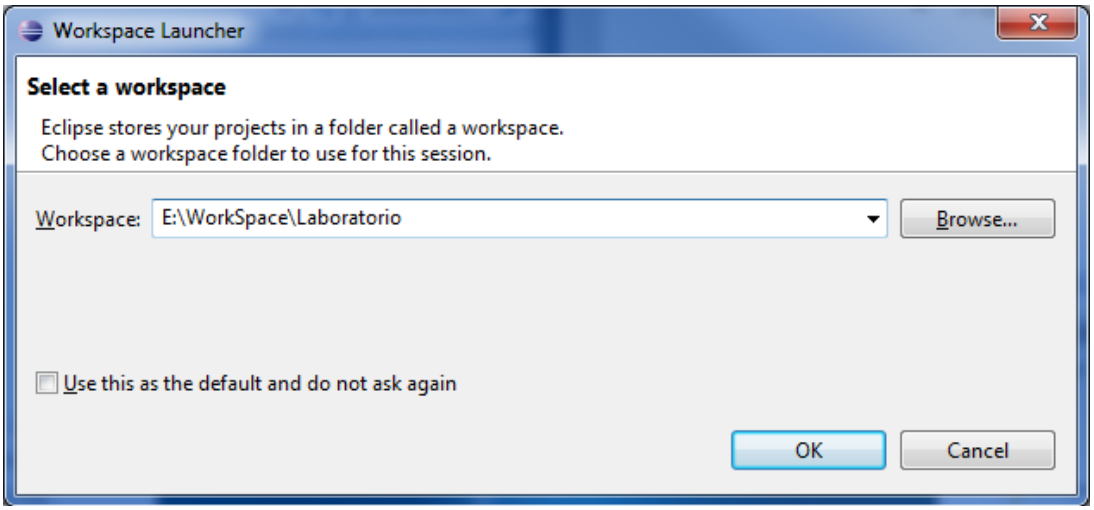


Figura 11 Selección del espacio de trabajo

Cuando el programa se haya cargado correctamente se nos presentara una zona de trabajo, como lo muestra la Figura 12. En ella se puede observar la perspectiva nombrada *Java*, en donde se presenta la distribución de las ventanas más utilizadas.

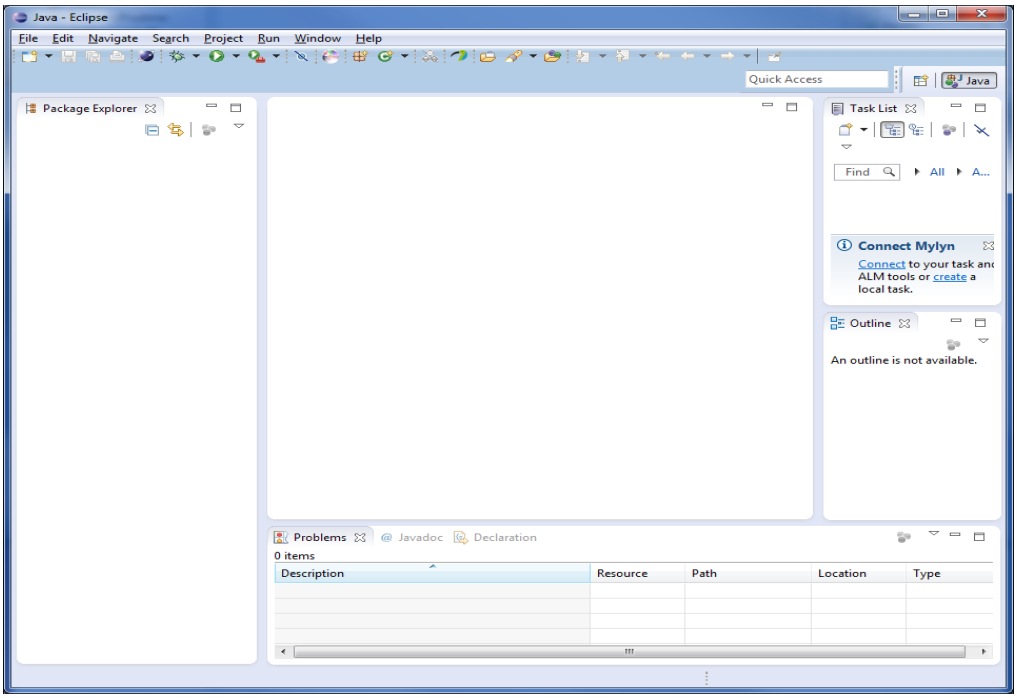


Figura 12 Zona de trabajo

2. Cambio de perspectiva

Para nuestro caso necesitamos realizar el cambio de perspectiva. Para esto daremos clic en icono presentado en la Figura 13, ubicado en la parte superior derecha de la zona de trabajo.

Figura 13 Icono cambio de perspectiva

Luego de dar clic, se nos presenta una ventana emergente, ver Figura 14, con el listado ordenado alfabéticamente de las perspectivas disponibles en nuestra instalación de Eclipse. Ahora debemos seleccionar la perspectiva con el nombre de “Papyrus”, y dar clic en “OK”

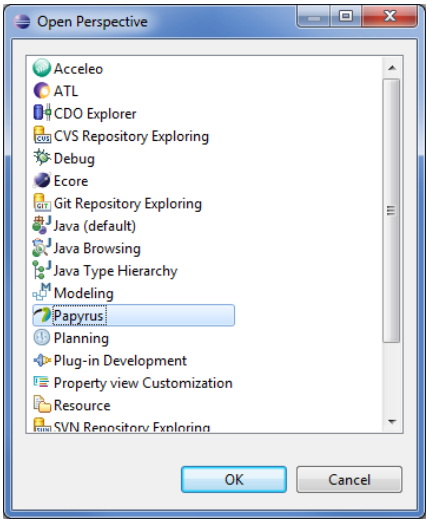


Figura 14 Lista de perspectivas

Esto realizara una distribución de la zona de trabajo, tal y como se presentara en la Figura 15.

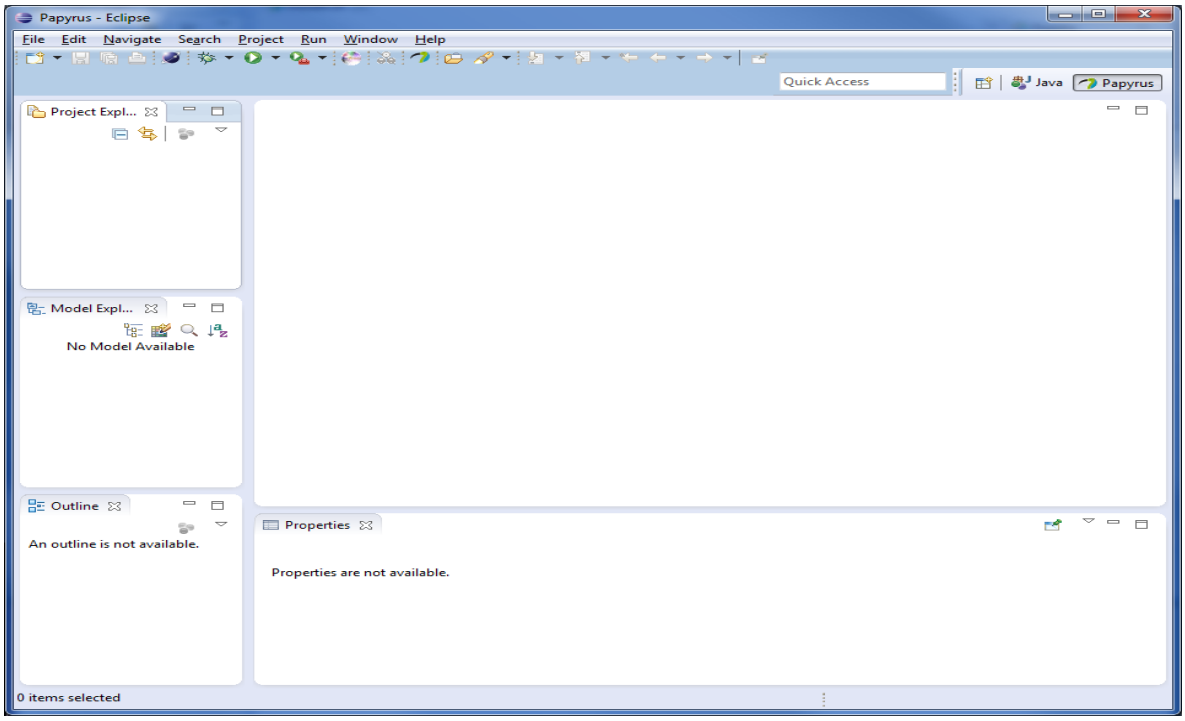


Figura 15 Zona de trabajo perspectiva Papyrus

3. Creación de un proyecto en Prudima

Para crear un proyecto en Prudima, debemos ubicar la ventana “Project Explorer”, parte superior izquierda de la zona de trabajo. Daremos clic derecho en la zona

blanca seleccionaremos la opción “New” >> “Other…”. Como se presenta en la Figura 16.

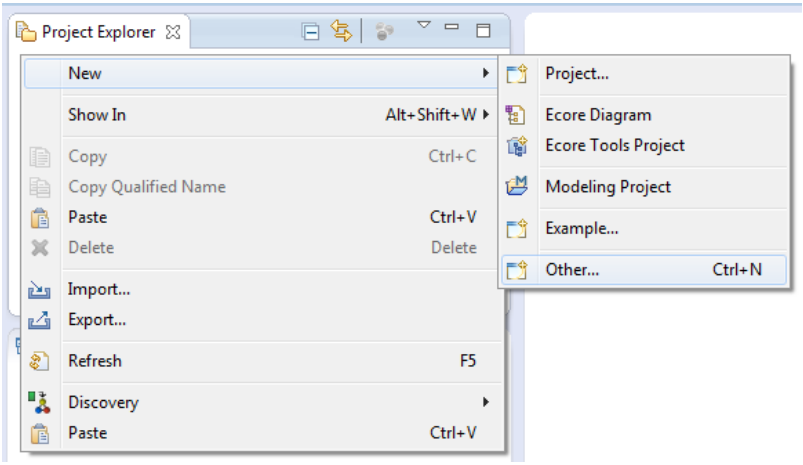


Figura 16 Selección Other

Ahora se nos presenta una ventana emergente para seleccionar el tipo de proyecto a crear.
Debemos ingresar en la entrada de texto de tipo filtro “Wizards” la palabra *Prudima*, es nos aplicara el filtro sobre todos los proyectos que contengan esa cadena, paso 1 Figura 17, ahora debemos seleccionar “Prudima Project ” paso 2 Figura 17 y finalmente dar clic en el botón “Next”, ubicado en la parte inferior de la ventana.

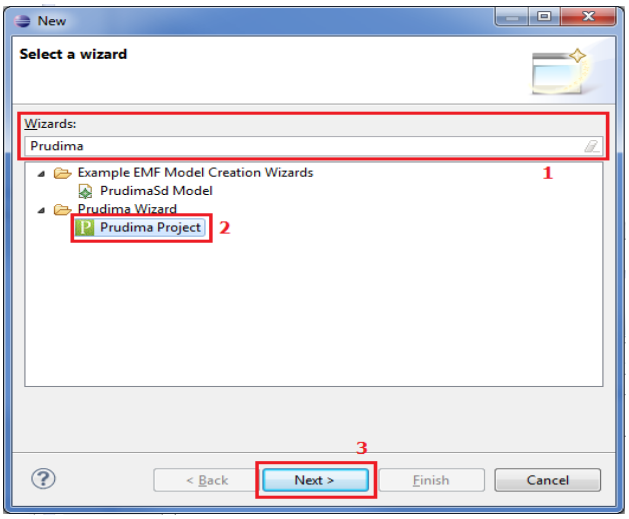


Figura 17 Selección del Papyrus Project

Después debemos configurar el nombre del proyecto, en nuestro caso nombraremos el proyecto “SUTElecciones”, tal y como se presenta en la Figura 18.

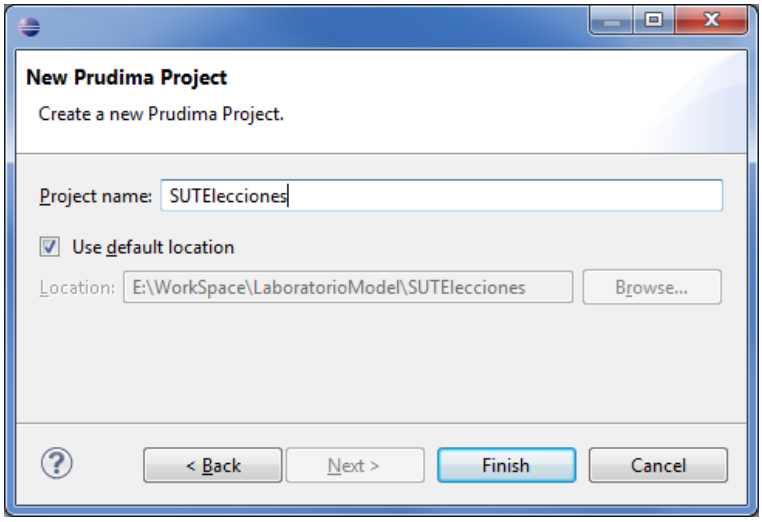


Figura 18 Configuración del proyecto

Ahora no ubicamos en la carpeta *model* dentro del proyecto damos clic derecho sobre esta y nos desplegará un menú el que aparece en la Figura 19.

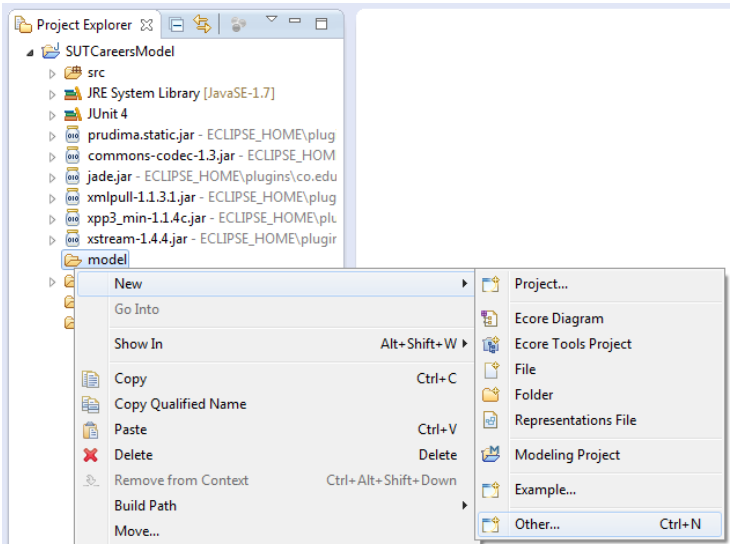


Figura 19 Creamos el modelo

Debemos ingresar en la entrada de texto de tipo filtro “Wizards” la palabra “*Papyrus”*, es nos aplicara el filtro sobre todos los proyectos que contengan esa cadena, paso 1 Figura 20, ahora debemos seleccionar “Prudima Project ” paso 2 Figura 20 y finalmente dar clic en el botón “Next”, ubicado en la parte inferior de la ventana.

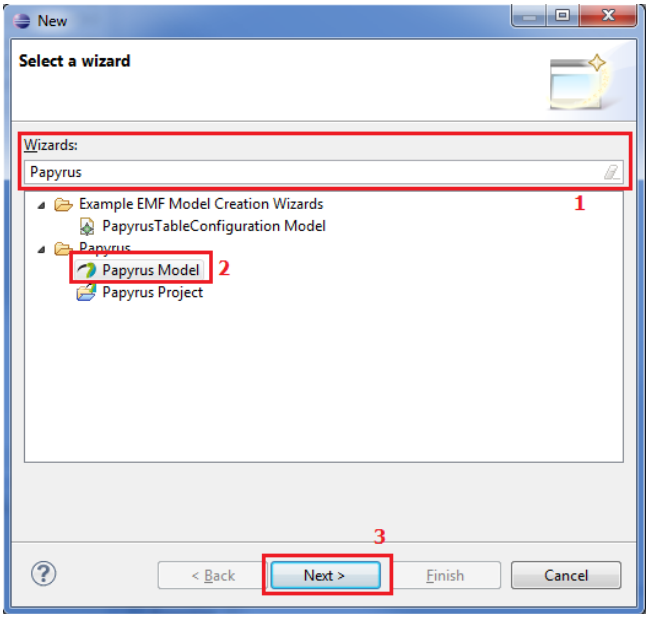


Figura 20 Seleccionamos Papyrus Model

Ahora en el cuadro de texto "File name" nombraremos el modelo "modelElecciones.di" y daremos clic en el botón "*Next >*"

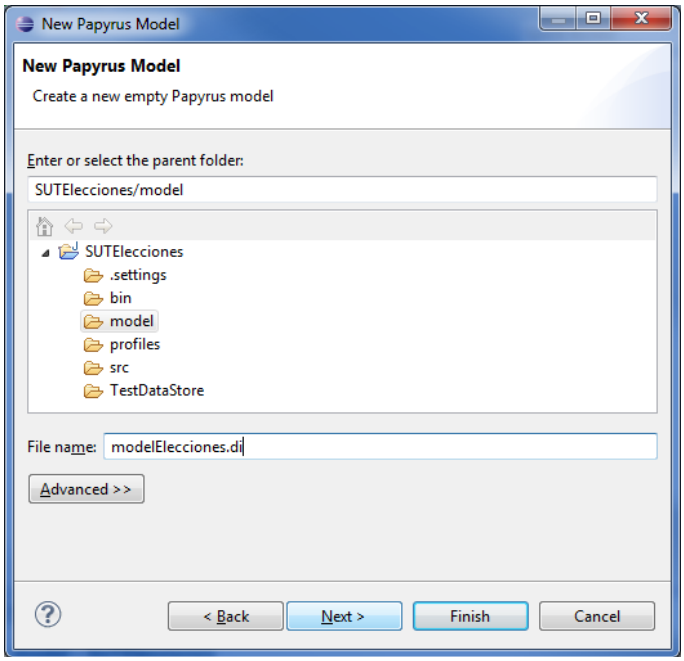


Figura 21 Nombramos el modelo

Ahora seleccionaremos el lenguaje del diagrama, seleccionando la opción "UML", luego daremos clic en el botón "*Next >*", como se observa en la Figura 22.

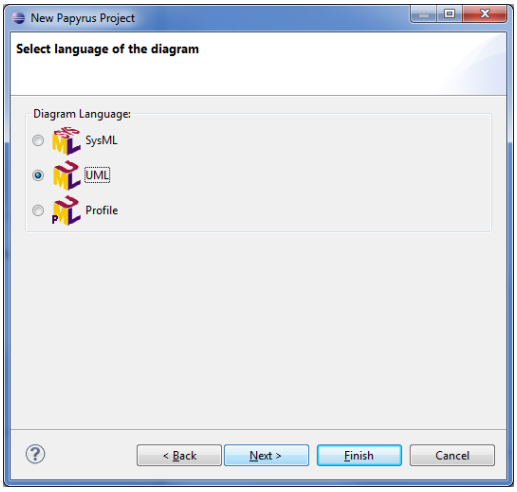


Figura 22 Selección del lenguaje del diagrama

Ahora debemos configurar la informacion inicial del modelo, por medio de los siguientes pasos: primer debemos nombrar el modelo en la casilla de texto “Diagram Name“ nombrándolo “ClassDiagram”, luego marcar el tipo de diagrama “UML Class Diagram”, luego marcaremos la carga de las primitivas seleccionando la opción “A UML model with …”, como se observa en la Figura 23.

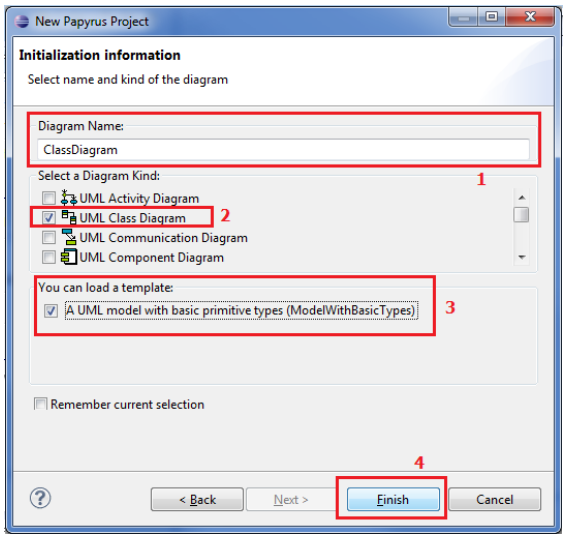


Figura 23 Configuración del diagrama

Como resultado obtendremos el proyecto en Papyrus, como se presenta en la Figura 24.

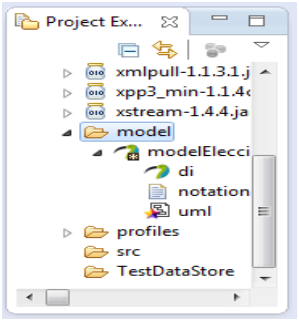


Figura 24 Proyecto en Papyrus generado

4. Distribución del espacio de trabajo en Papyrus

En la Figura 25 que se presenta a continuación encontramos las zonas de trabajo que nos presta la perspectiva Papyrus estas serán descritas de manera detallada, seleccionada en pasos anteriores.

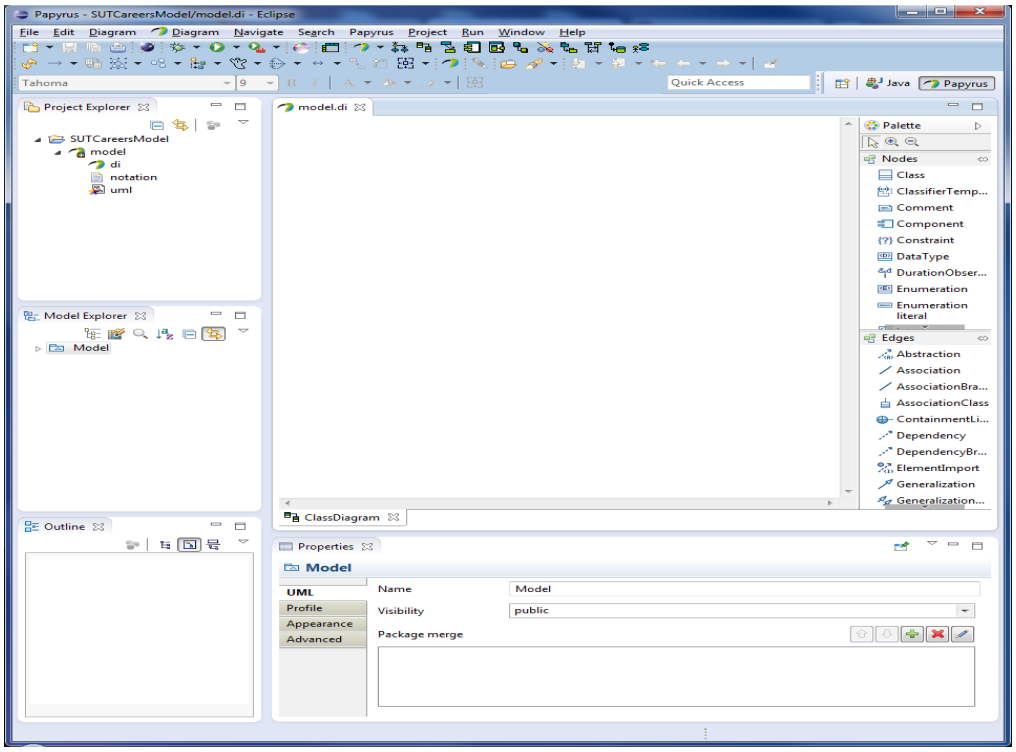
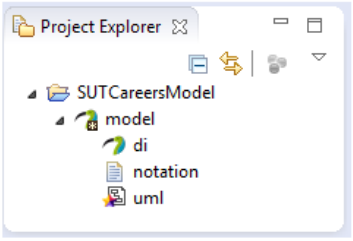


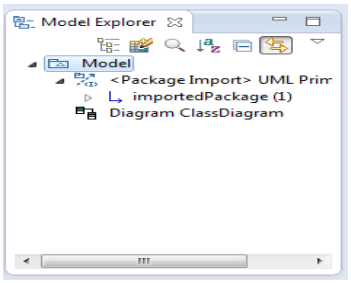
Figura 25 Espacio de trabajo modelo en Papyrus

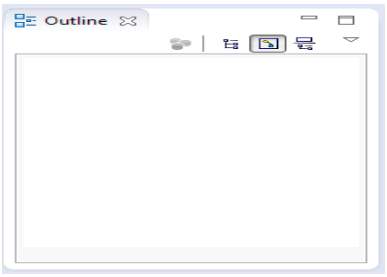


Project Explorer

Dentro de esta zona se puede encontrar los proyectos que se encuentran vinculados actualmente al *Workspaces*, en el que estemos trabajando. Este se encuentra ubicado en la parte izquierda de la zona de trabajo.

Model Explorer
Este exportador nos presenta los diferentes elementos que componen el modelo que estemos trabajando, esta presentación la realiza en forma de jerárquica.





Outline
Esta zona nos presenta una vista global del modelo que estemos trabajando.

Espacio de modelado

Esta zona de trabajo se divide en dos, la parte izquierda nos representa el espacio donde debemos expresar nuestro modelo, en la parte inferior se presenta a manera de pestañas los diferentes diagramas que representan nuestro modelo. Ya en la parte derecha encontramos los diferentes elementos que se disponen para representar el diagrama que estemos trabajando actualmente.

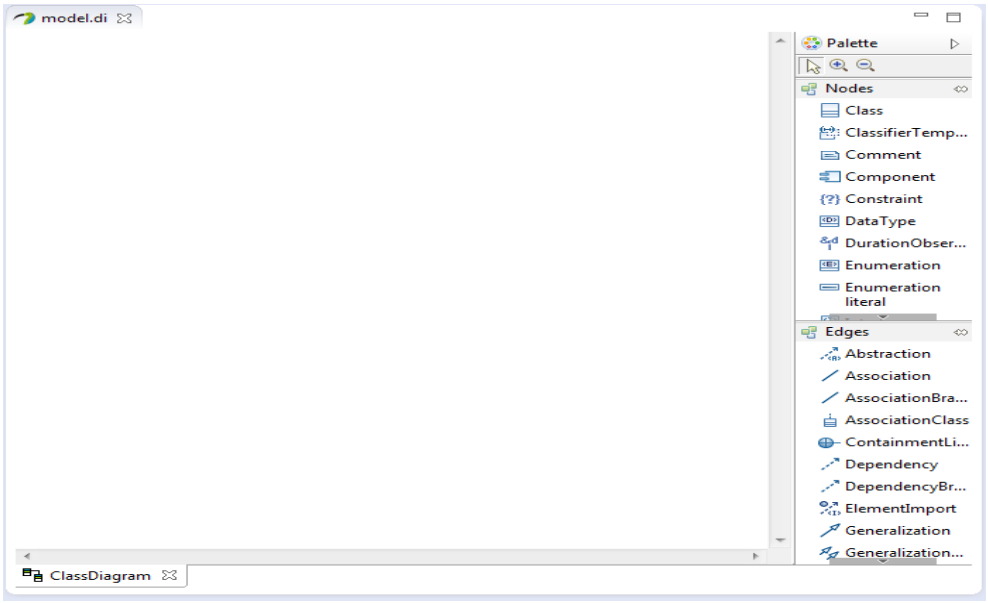


Figura 26 Espacio de modelado

Properties

En esta zona de trabajo encontramos un panel cuyo objetivo es ayudar a especificar las propiedades del elemento que tengamos seleccionado actualmente en el espacio de modelado. Es decir que si tenemos seleccionada una clase, en esta ventana “Properties” podremos editar la información de esta.

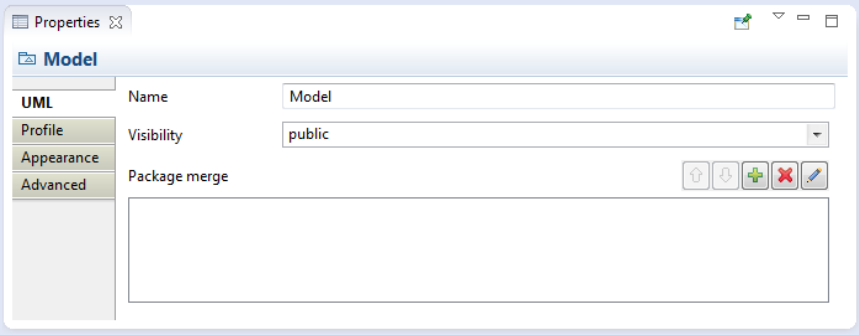


Figura 27 Propiedades del elemento

5. Aplicación del Perfil PrudimaSD al modelo

Para este paso primero debemos abrir el modelo nombrado “modelElecciones.di” que se encuentra dentro del proyecto “SUTElecciones”, esto nos presentara un espacio de trabajo en donde podemos realizar el diseño Figura 25.

Daremos clic en la parte blanca de la zona de modelado, esto generara que cambie la información de la ventana “Properties”, como se ve en la figura 15. Ahora iremos a la pestaña “Profile” ubicada en la parte izquierda de dicha ventana.

```
package careers.system.services;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ICareersSystem  extends Remote {
	public String Login (String userName, String password) throws
RemoteException;
	public Boolean Logout (String userName) throws RemoteException;
	public String AddUser (String userName, String password) throws
RemoteException;
	public String DelUser (String userName) throws RemoteException;
	public String AssignRole (String userName, Integer rolName) throws
RemoteException;
	public Integer ViewUsersOnline (Integer roleType) throws
RemoteException;
	public String AddCareer (String careerName) throws
```

```
RemoteException;
        public Boolean DelCareer (String careerName) throws
RemoteException;
        public String ViewListCareers () throws RemoteException;
}
```

Tabla 14 Código fuente de la clase interfaz del SUT

Muchas gracias por su colaboración.

9.2. GUIA DE CASO DE ESTUDIO DE PRUDIMA - PRUEBAS ENFOQUE MBT

Introducción

El paradigma de pruebas dirigidas por modelos permite la generación automática de casos de pruebas a partir de modelos construidos en las fases de desarrollo de software. Siguiendo el enfoque MDA, se utilizan una o un conjunto de transformaciones a los modelos, donde finalmente se obtiene un código ejecutable que permite probar el sistema en desarrollo. Con MBT se pueden realizar diferentes tipos de pruebas tales como las pruebas unitarias, de sistema, de regresión, de aceptación, entre otras.
En esta práctica nos enfocaremos en las **unitarias y de sistema**.

Objetivo

El objetivo de este caso de estudio es verificar el proceso definido en este trabajo, a través de la comparación entre la aplicación de las pruebas dirigidas por modelos y el uso del enfoque tradicional de pruebas manual en el contexto académico del laboratorio de sistemas distribuidos de la Universidad del Cauca.

Sistema a probar

El caso de estudio analiza un sistema denominado *CareersSystem (Sistema de carreras)* desarrollado por los autores del presente trabajo de grado y se basa en las prácticas propuestas por parte del docente del curso de laboratorios de sistemas distribuidos, del cual, se obtuvieron los requerimientos y la documentación necesarios a nivel de diseño y técnicos. En este sistema se utiliza la tecnología de comunicación de invocación remota (*Remote Method Invocation* - RMI) y se proveen servicios de gestión de usuarios del sistema y de carreras profesionales que luego pueden ser consultadas por los usuarios del mismo.

En este sistema existen 3 tipos de usuarios, cada uno de ellos puede ejecutar diferentes funcionalidades descritar a continuación:

1) ***Administrator***: puede agregar o eliminar usuarios, tanto del tipo *Secretary* como *Student* en el sistema.
   Existe un único usuario por defecto llamado "admin" con password "admin123" con rol de Administrador.
2) ***Secretary***: puede agregar o eliminar carreras en el sistema.
3) ***Student***: puede visualizar las carreras del sistema

A continuación se presenta un diagrama de casos de uso del sistema:

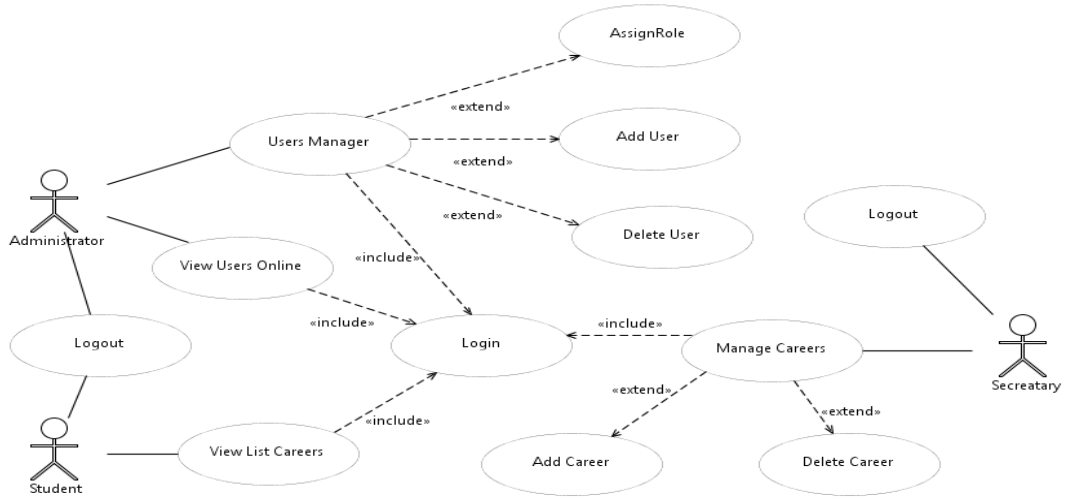


Figura 28 - Diagrama de casos de uso CareersSystem

Requerimientos del sistema a probar

En la siguiente tabla se presentan los requerimientos que tendrán cobertura en las pruebas:

Id	Descripción requerimiento	Tipo de Prueba
1	El número de máximo de usuarios del tipo *Secretary (tipo 1)* en el servidor es de 2, en caso de que ya existan dos usuario y un tercero intente hacer Login en el sistema deberá de retornar el mensaje: "Users exceeded"	Sistema
2	Al crear un usuario por parte del actor *Administrator,* el nombre del usuario deberá ser de una longitud mínima de 8 y máxima de 10 caracteres, y el mensaje del servidor deberá ser "User registered"	Sistema
3	Al crear un usuario por parte del actor *Administrator,* la contraseña del usuario deberá ser de una longitud mínima	Sistema

	de 4 y máxima de 8 caracteres alfanuméricos y debe contener al menos 1 carácter especial de entre esta lista(@,#,$,&,?,*).	
4	El actor *Administrator* no podrá eliminar un usuario, del tipo *Secretary* que este autenticado en el sistema y deberá retornar el mensaje “User is online”.	Sistema
5	No es válido tener dos usuarios con el mismo nombre dentro del sistema, cuando el administrador intente agregar uno repetido saldrá el mensaje “User already exists”	Sistema
6	La autenticación del usuario Administrador solo deberá ocurrir si el nombre de usuario es “admin” (todo en minúsculas) y el sistema retornará “admin is authenticated”	Unitaria

Nota: *Los anteriores requerimientos ya se encuentran modelados y cargados en el laboratorio.*

Desarrollo de la práctica

Paso 1 - Agregamos el proyecto en la configuración de PRUDIMA

Para realizar este paso primero debemos especificar el proyecto de la siguiente forma, seleccionamos el Proyecto Lab01 o Lab02 respectivamente. Dentro del <<Packege Explorer>>, luego damos clic derecho sobre el proyecto, seleccionamos la opción <<Properties>>.

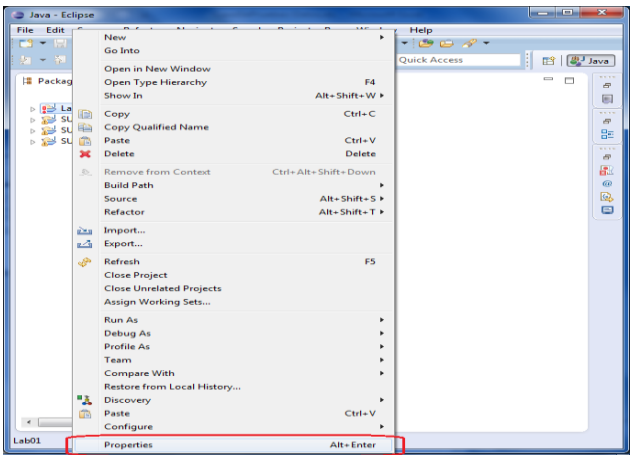


Figura 29 - Seleccionar la opción Properties en el proyecto

Ahora en el menú lateral ubicado en la parte derecha, vamos a la opción <<Java Build Path>>, después vamos a la pestaña <<Projects>>, luego seleccionamos el botón ubicado en la columna derecha <<Add…>>

Nota: *Si existe una vinculación a un proyecto anterior debemos borrarla para esto la seleccionamos y a continuación damos clic sobre el botón ubicado en la parte derecha <<Remove>>*

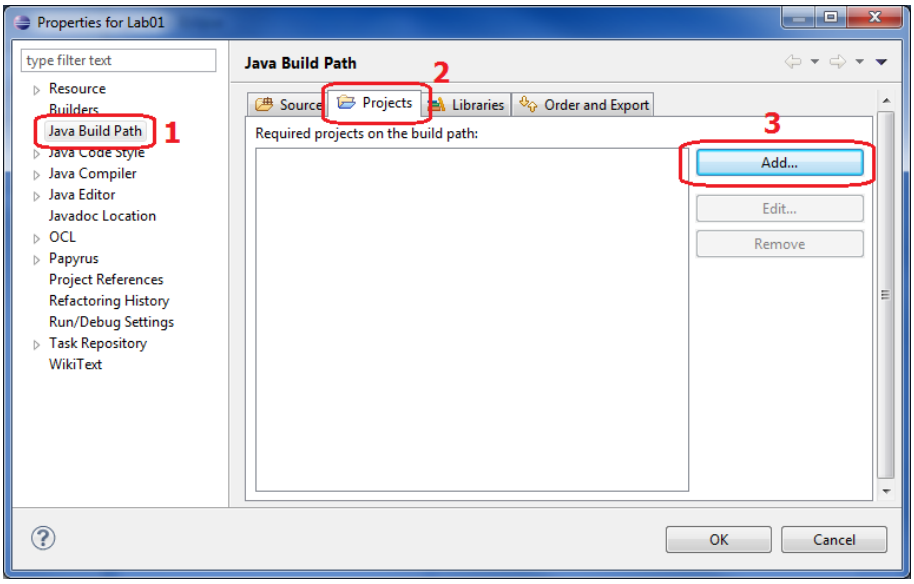


Figura 30 - Configuración Java Build Path

Seleccionamos el proyecto que deseamos añadir, para esto marcamos el cuadro de selección y luego damos clic sobre el botón <<OK>> ubicado en la parte inferior, como se ve en la siguiente figura.

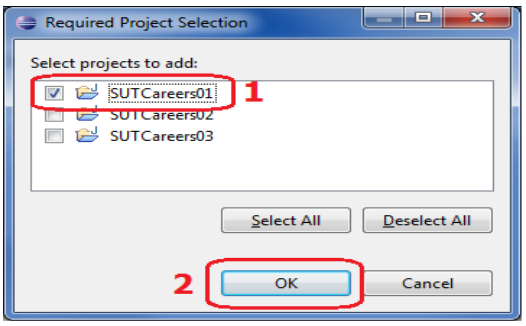


Figura 31 - Selección del proyecto

Damos <<OK>> para aprobar la selección del proyecto en la ventana de Java Build Path, como se ve en la Figura 31.

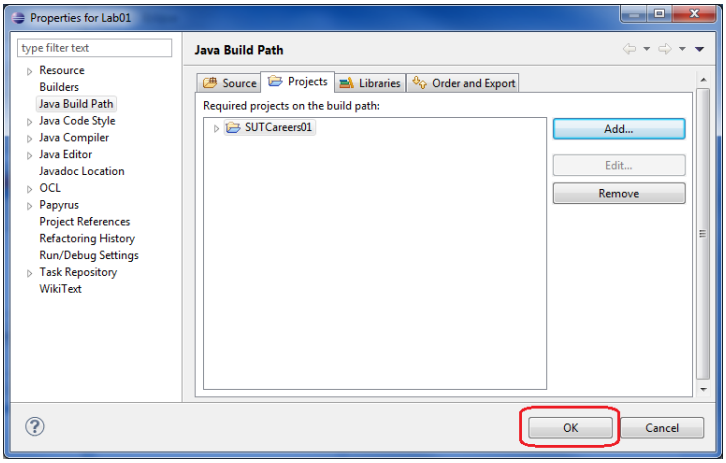


Figura 32 - Confirmamos el proyecto

Ahora debemos editar el archivo Config.ini ubicado en la siguiente ruta [Workspaces]\Lab01\TestDataStore\config.ini
Para esto abrimos el archivo e ingresamos el nombre del proyecto, este debe estar sin espacios, comas ni puntos.

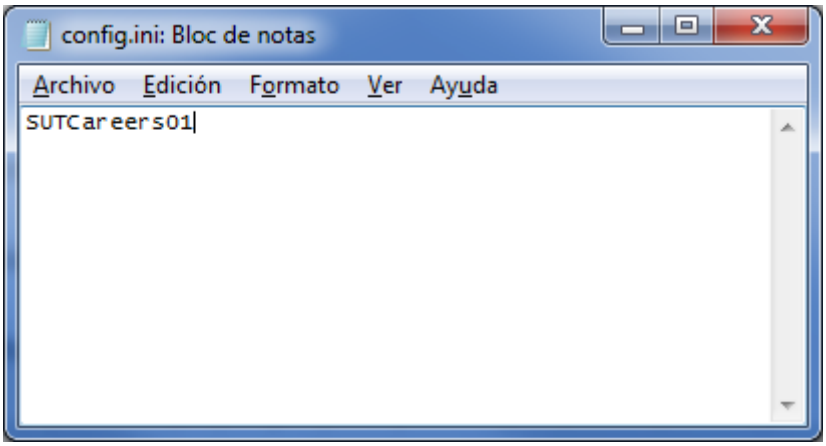


Figura 33 - Archivo config.ini

Ya tenemos nuestro proyecto SUT configurado y podemos lanzar las pruebas.

Paso 2 - Para lanzar la herramienta PRUDIMA

Primer debemos ubicar el archivo PrudimaMain.java, ubicado en [Workspaces]\src\manager\PrudimaMain.java
Luego sobre el este debemos darle clic derecho y seleccionar la opción <<Run As>> y seleccionamos la opción <<Java Application>> como se muestra en la figura.

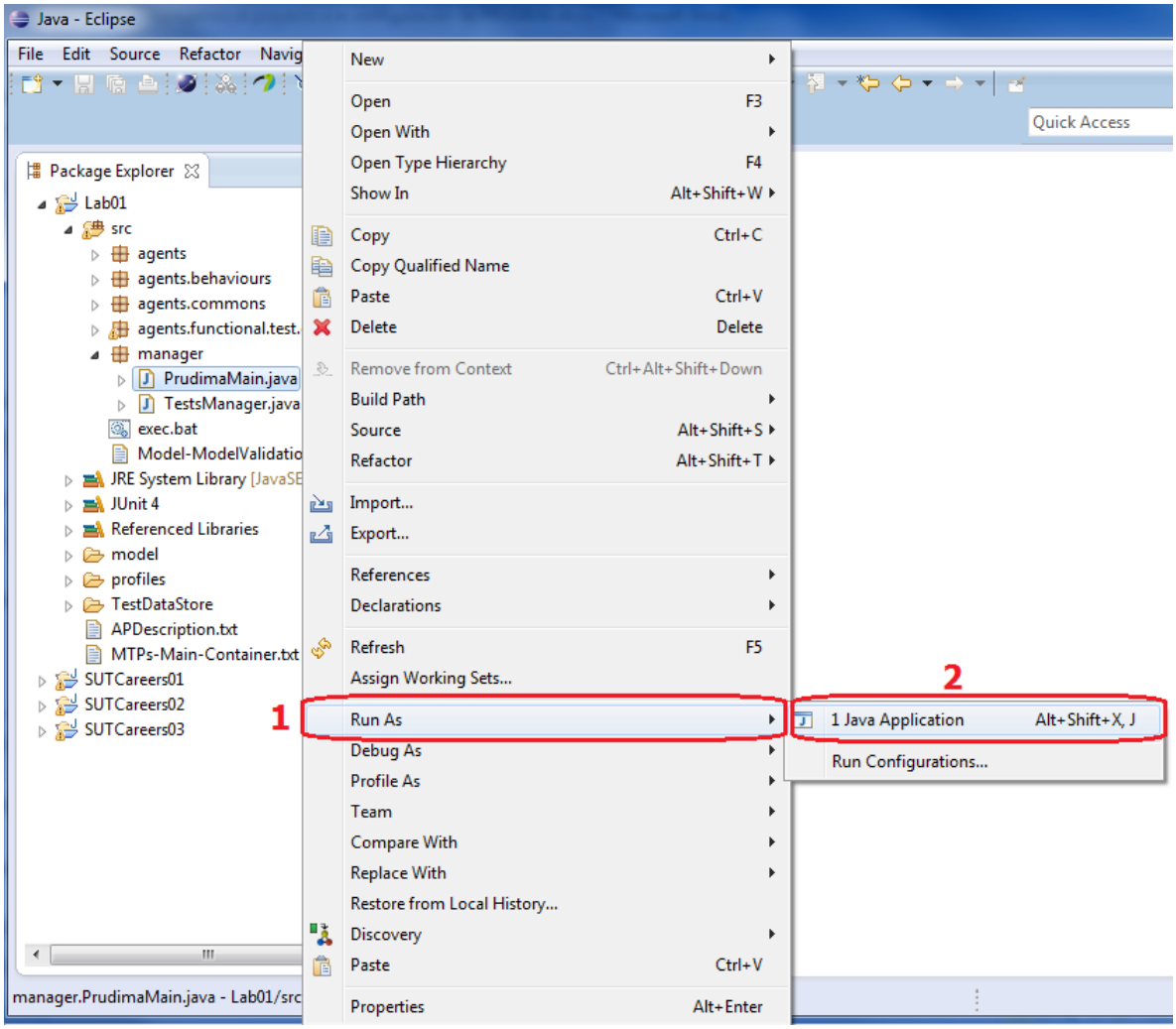


Figura 34 - Opción Run As

La cual nos lanzara la herramienta Prudima y se nos presentara una ventana con las siguientes opciones.

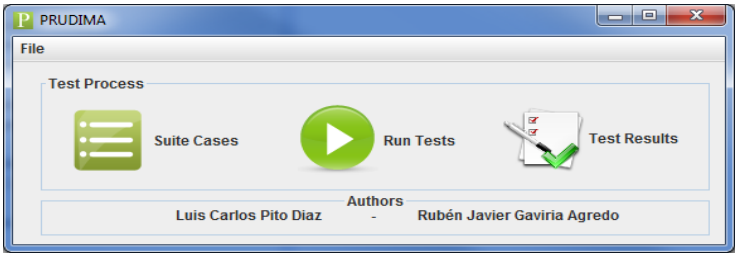


Figura 35 - Menú principal PRUDIMA

Suite Cases: Opción donde se ingresan los diferentes casos de pruebas.
Run Tests: Opción que nos permite lanzar la aplicación y ejecutar los casos de prueba.
Test Results: Esta opción nos permite visualizar los resultados de las pruebas.

Paso 3 – Carga de datos en PRUDIMA

Para ingresar los casos de prueba primero debemos seleccionar dentro del **menú principal de PRUDIMA** la opción <<Suite Cases>>, esta nos desplegará los diferentes **Casos de Pruebas** que la herramienta ejecutara.

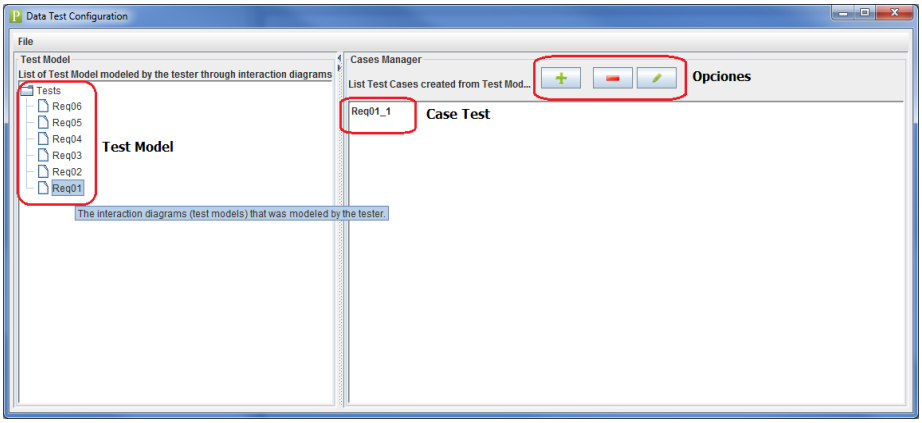


Figura 36 - Data Test Configuration

En el lado derecho de esta pantalla se nos presenta los modelos de pruebas que se han configurado previamente a la herramienta por medio de modelos, en el caso de este laboratorio se ha ingresado 6 diferentes modelos de pruebas que aparecen en el lado izquierdo de la pantalla, como se ve en la Figura 36. Para trabajar con los **Casos de Prueba** de un modelo de prueba primero debemos seleccionar el modelo de prueba dándole doble clic. Luego se nos actualizara la lista de los casos de prueba disponibles en la parte derecha de la pantalla, como se ve en la Figura 36.

Ahora debemos trabajar con los casos de prueba. Para esto se dispuso en la parte superior una serie de botones que nos ayudaran con la interacción.

 Agrega los casos de pruebas al **Test Model**

 Borra el caso de prueba seleccionado del **Test Model**

 Lanza la ventana de edición para el caso de prueba seleccionado

Ahora para agregar un caso de prueba al **Test Model**, debemos seleccionar el botón <<+>>, el cual nos crea un caso de prueba por defecto. Para editarlo debemos seleccionar el caso de prueba y luego dar clic sobre el icono del lápiz, esto nos abre una ventana con los pasos que se van a ejecutar dentro de esta prueba, como se ve en la figura 10. Para ingresar los valores a probar ahora debemos escoger el paso al cual le queremos ingresar valores, para esto seleccionamos el paso y después demos clic sobre el botón edit ubicado en la parte inferior.

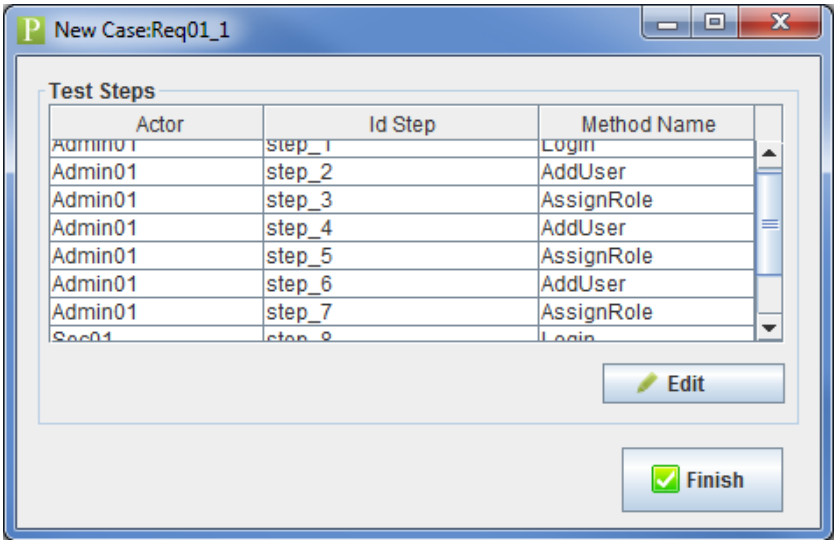


Figura 37 - Visualización caso de prueba

Esto nos lanzara una nueva ventana indicándonos los valores que debemos ingresarle a la prueba para esto cambiamos el valor de la última columna llamada *Value*, con los valores entregados dentro del archivo en Excel. Luego debemos guardar para esto damos clic sobre el botón <<Save>> ubicado debajo de la lista de parámetros.

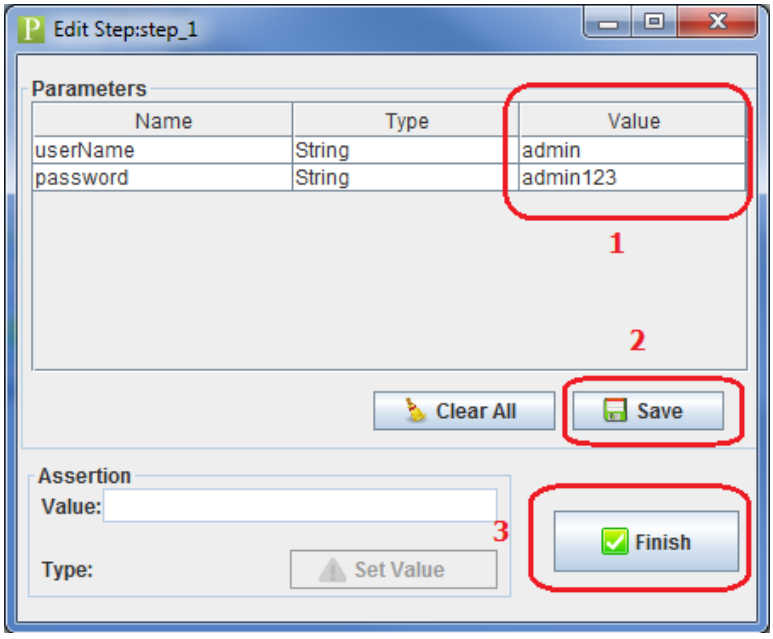


Figura 38 - Visualización Parámetros de un Step

Para ingresar una Assercion debemos ingresarla dentro de la casilla Assertion Value y luego guardar el valor de esta por medio del botón <<Set Value>>

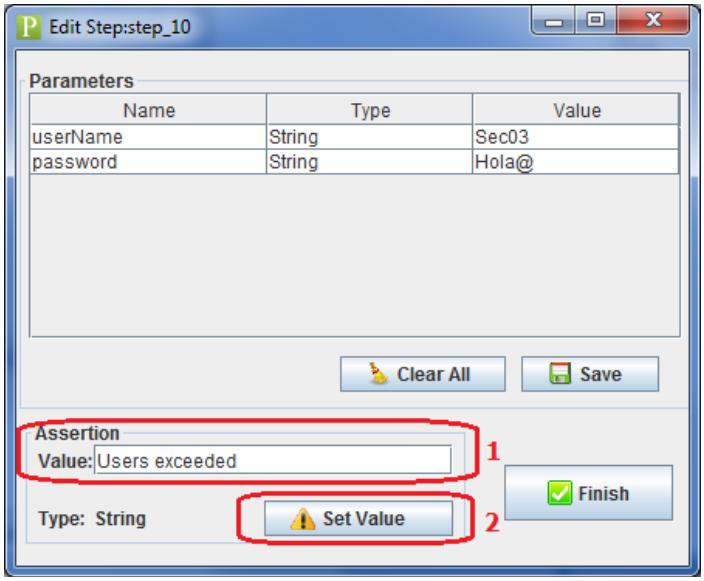


Figura 39 - Ingresar Assertion

Nota: *tenga en cuenta que las validaciones realizadas por el sistema son sensibles a los espacios, por este motivo por favor cerciore que no exista un espacio al final de la cadena si este no es necesario o requerido.*

Una vez modificado los valores del Caso de prueba debemos guardar los valores de todos los modelos de prueba, estos para esto vamos al menú superior a <<File>> y luego damos clic en <<Save>>.

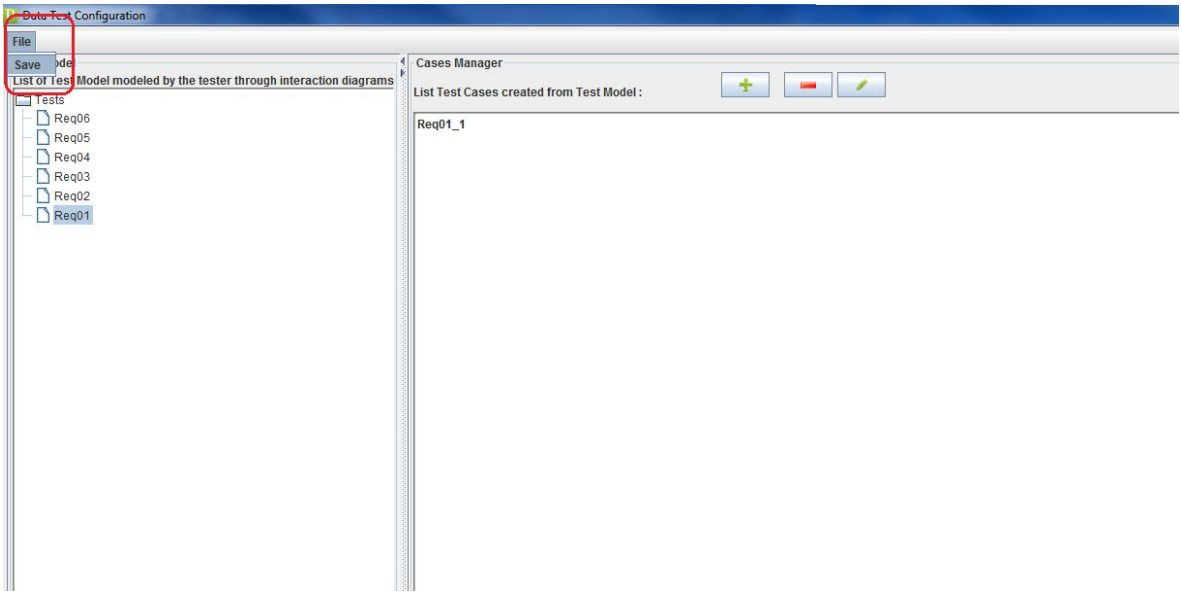


Figura 40 - Guardar los Model Test

Paso 4 – Ejecutar y analizar los datos

Una vez se haya ingresado todos los casos de pruebas expuestos en el archivo en Excel guardamos por última vez y cerramos la ventana. Luego regresamos al menú principal para iniciar la prueba. Para esto damos clic sobre el botón <<Run Test>>, Figura 35. El cual lanza la aplicación, esperamos alrededor de 2 a 3 minutos a que ejecute cada una de las pruebas.

Finalizado este paso procedemos a abrir el resultado obtenido en la opción <<Resul test>> anotamos los valores de la pruebas fallidas, cuales pruebas fallaron y la razón del fallo.

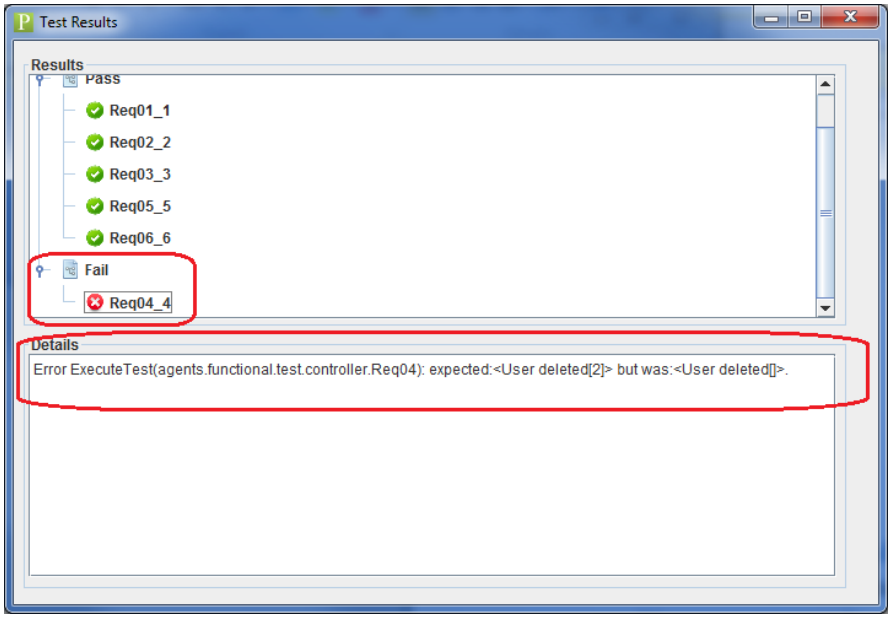


Figura 41 - Test Result

Luego procedemos a cambiar de SUT, para esto ejecutamos el paso 1, paso 2 y el paso 4 expuestos en esta guía.

Una vez terminado estos pasos, se le indicara los pasos a seguir.

Muchas gracias por su colaboración.

9.3. GUIA DE CASO DE ESTUDIO DE PRUDIMA - PRUEBAS MANUALES

Introducción

El paradigma de pruebas dirigidas por modelos permite la generación automática de casos de pruebas a partir de modelos construidos en las fases de desarrollo de software. Siguiendo el enfoque MDA, se utilizan una o un conjunto de transformaciones a los modelos, donde finalmente se obtiene un código ejecutable que permite probar el sistema en desarrollo. Con MBT se pueden realizar diferentes tipos de pruebas tales como las pruebas unitarias, de sistema, de regresión, de aceptación, entre otras.
En esta práctica nos enfocaremos en las unitarias y de sistema.

Objetivo

El objetivo de este caso de estudio es verificar el proceso definido en este trabajo, a través de la comparación entre la aplicación de las pruebas dirigidas por modelos y el uso del enfoque tradicional de pruebas manual en el contexto académico del laboratorio de sistemas distribuidos de la Universidad del Cauca.

Sistema a probar

El caso de estudio analiza un sistema denominado *CareersSystem (Sistema de carreras)* desarrollado por los autores del presente trabajo de grado y se basa en las prácticas propuestas por parte del docente del curso de laboratorios de sistemas distribuidos, del cual, se obtuvieron los requerimientos y la documentación necesarios a nivel de diseño y técnicos. En este sistema se utiliza la tecnología de comunicación de invocación remota (*Remote Method Invocation* - RMI) y se proveen servicios de gestión de usuarios del sistema y de carreras profesionales que luego pueden ser consultadas por los usuarios del mismo.
En este sistema existen 3 tipos de usuarios, cada uno de ellos puede ejecutar diferentes funcionalidades descritar a continuación:

4) ***Administrator***: puede agregar o eliminar usuarios, tanto del tipo *Secretary* como *Student* en el sistema.
   Existe un único usuario por defecto llamado “admin” con password “admin123” con rol de Administrador.
5) ***Secretary***: puede agregar o eliminar carreras en el sistema.
6) ***Student***: puede visualizar las carreras del sistema

A continuación se presenta un diagrama de casos de uso del sistema:

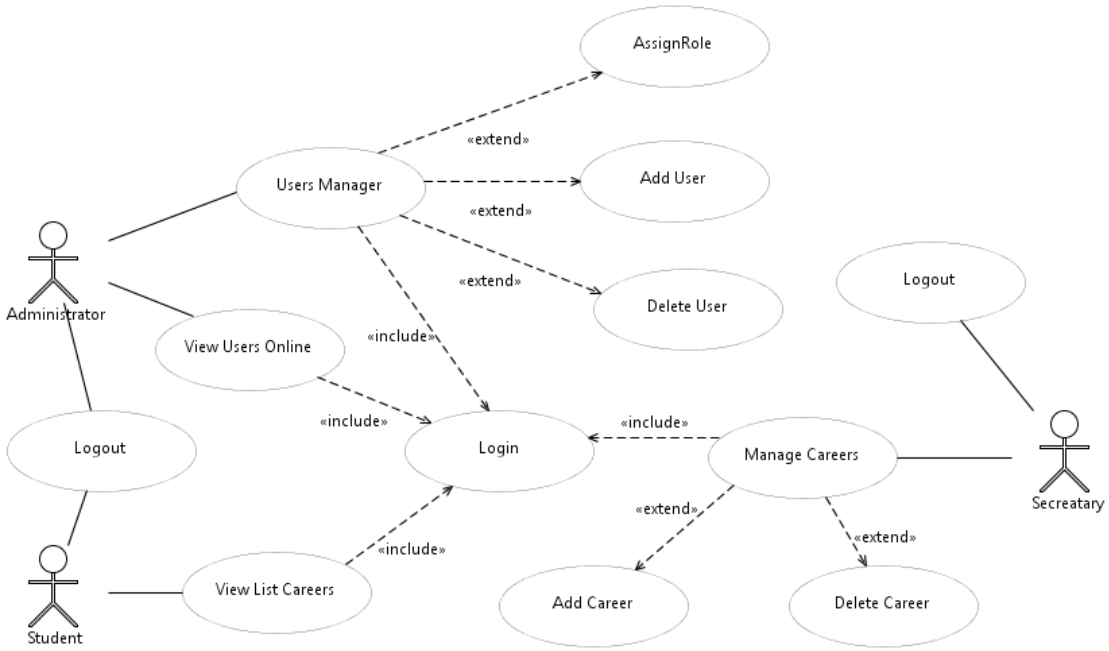
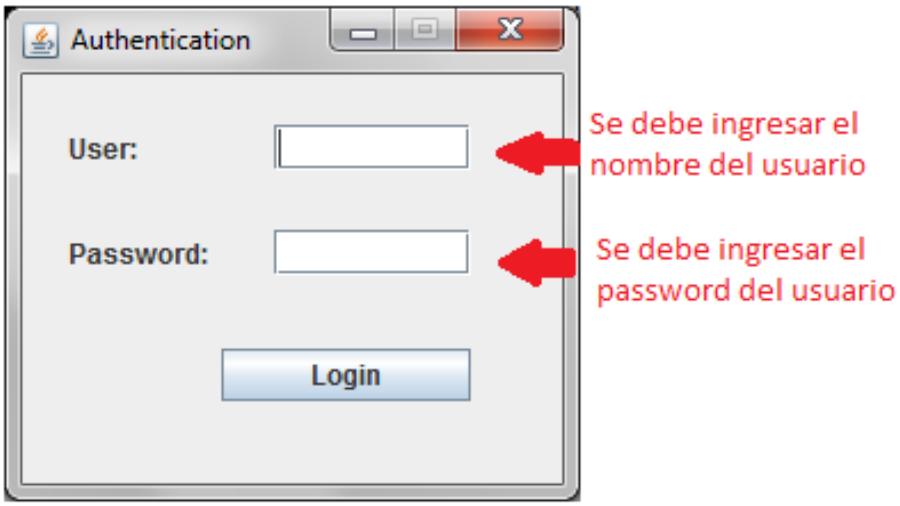


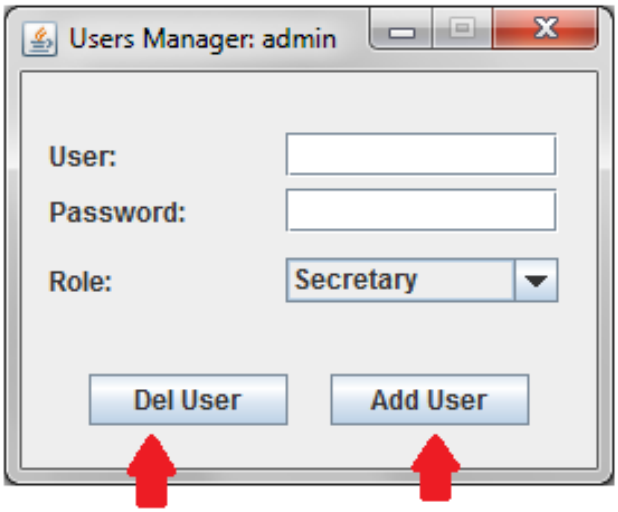
Figura 42 Diagrama casos de uso CareersSystem

Interfaz gráfica del cliente del sistema a probar

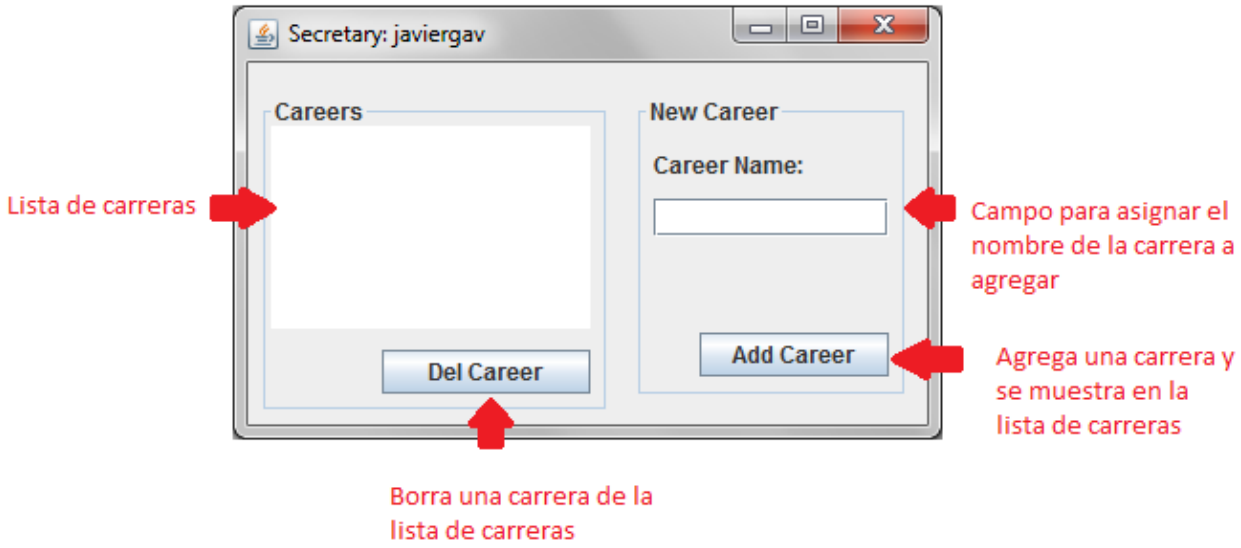
Forma Authentication



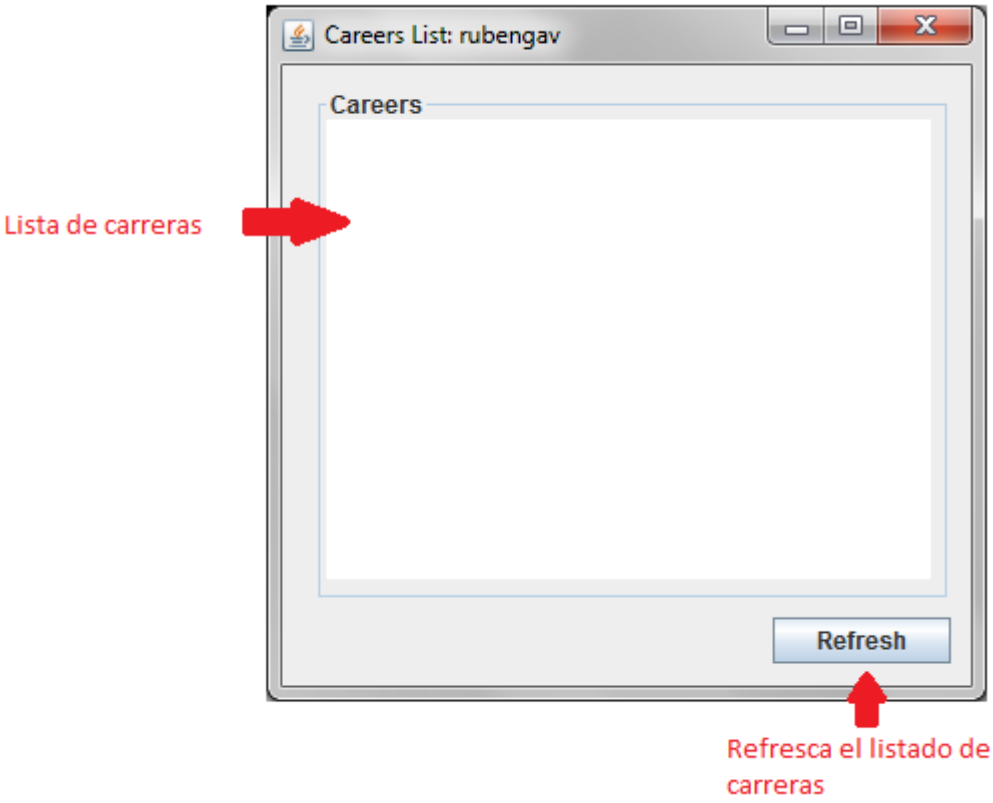
Forma Users Manager (del usuario admin)



Forma de Secretary (solo usuario de tipo Secretary)



Forma Careers List (solo usuario de tipo Student)



Requerimientos del sistema a probar

En la siguiente tabla se presentan los requerimientos que tendrán cobertura en las pruebas:

Id	Descripción requerimiento	Tipo de Prueba
1	El número de máximo de usuarios del tipo *Secretary (tipo 1)* en el servidor es de 2, en caso de que ya existan dos usuario y un tercero intente hacer Login en el sistema deberá de retornar el mensaje: “Users exceeded”	Sistema
2	Al crear un usuario por parte del actor *Administrator,* el nombre del usuario deberá ser de una longitud mínima de 8 y máxima de 10 caracteres, y el mensaje del servidor deberá ser “User registered”	Sistema
3	Al crear un usuario por parte del actor *Administrator,* la contraseña del usuario deberá ser de una longitud mínima de 4 y máxima de 8 caracteres alfanuméricos y debe	Sistema

	contener al menos 1 carácter especial de entre esta lista(@,#,$,&,?,*).	
4	El actor *Administrator* no podrá eliminar un usuario, del tipo *Secretary* que este autenticado en el sistema y deberá retornar el mensaje “User is online”.	Sistema
5	No es válido tener dos usuarios con el mismo nombre dentro del sistema, cuando el administrador intente agregar uno repetido saldrá el mensaje “User already exists”	Sistema
6	La autenticación del usuario Administrador solo deberá ocurrir si el nombre de usuario es “admin” (todo en minisculas) y el sistema retornará “admin is authenticated”	Unitaria

DESARROLLO DE LA PRÁCTICA

Guía de ejecución del sistema

Ejecutar el servidor

Para ejecutar el servidor del sistema se debe ubicar en la carpeta bin del proyecto a probar y ejecutar los scripts en el orden siguiente:

1) iniciarRmiRegistry.bat
2) iniciarServidor.bat

Ejecución del cliente

Para ejecutar el cliente se debe ubicar en la carpeta bin del proyecto a probar y ejecutar el siguiente script:

1) iniciarCliente.bat

Se debe tener en cuenta que se pueden iniciar varios clientes dependiendo de las necesidades de las pruebas.

Casos de pruebas

Una vez tengamos el servidor ejecutándose, se deben iniciar el cliente y empezar a ejecutar los siguientes casos de pruebas. Se debe tener en cuenta que para cada caso de prueba se para la ejecución del servidor y volverlo a ejecutar, ya que si no se realiza esta tarea, el sistema queda en un estado inconsistente.

Casos de prueba que abarcan los requerimientos del sistema

Se debe rellenar los datos de las formas con los datos del archivo de excel CasosDePruebasManuales.xls

10. ANEXO J

A continuación en la Tabla 15 Tiempo enfoque manual, se presentan los tiempos tomados por participantes de la primera sesión, para los enfoques manual. [12]

Grupo	**Servidor**		**Ejecución**				
	sti	**stf**	**eti**	**etf**	**stf-sti**	**etf-eti**	**Subtotal tiempos (min)**
G1	10:01:00	10:07:00	10:07:00	10:16:00	00:06:00	00:09:00	00:15:00
G2	10:21:00	10:24:00	10:27:00	10:33:00	00:03:00	00:06:00	00:09:00
G3	10:39:00	10:41:00	10:45:00	10:54:00	00:02:00	00:09:00	00:11:00
G4	11:04:00	11:07:00	11:40:00	11:44:00	00:03:00	00:04:00	00:07:00
G5	11:57:00	11:59:00			00:02:00	00:00:00	00:02:00
G6	13:37:00	13:39:00	13:42:00	13:46:00	00:02:00	00:04:00	00:06:00
G7	14:11:00	14:14:00	14:16:00	14:19:00	00:03:00	00:03:00	00:06:00
G8	14:29:00	14:31:00	14:03:00	14:32:00	00:02:00	00:29:00	00:31:00
G9	14:41:00	14:43:00	14:49:00	14:53:00	00:02:00	00:04:00	00:06:00
G10	17:01:00	17:09:00			00:08:00	00:00:00	00:08:00
G11	17:14:00	17:16:00	17:19:00	17:25:00	00:02:00	00:06:00	00:08:00
G12	18:48:00	18:50:00			00:02:00	00:00:00	00:02:00
		TIEMPO TOTAL PARA PRUEBAS DE 204 PRUEBAS					1:51:00

Tabla 15 Tiempo enfoque manual

11. ANEXO K

11.1. Pasos de instalación

Para finalizar esta fase de transición se definió el proceso de despliegue de forma que fuese lo más amigable posible para el usuario final. Para este proceso se debe manualmente, instalar los *plugins* y librerías estáticas de PRUDIMA, a la versión de la plataforma Eclipse en su versión 4.2 llamada Juno.

Para esto, se definieron los siguientes pasos:

1. Copiar los *plugins* de PRUDIMA dentro de la carpeta *plugins* de la instalación de Eclipse:
    - co.edu.unicauca.prudima.acceleo.ui_1.0.0.201308251813.jar
    - co.edu.unicauca.prudima.acceleo_1.0.0.201308251813.jar
    - co.edu.unicauca.prudima.xml.ui_1.0.0.201308251813.jar
    - co.edu.unicauca.prudima.xml_1.0.0.201308251813.jar

Para esto debemos abrir el explorador de carpetas e ingresar a la ruta donde se encuentre instalada nuestra versión de eclipse, una vez ahí, ingresamos a la carpeta plugins, como se presenta en la siguiente figura.

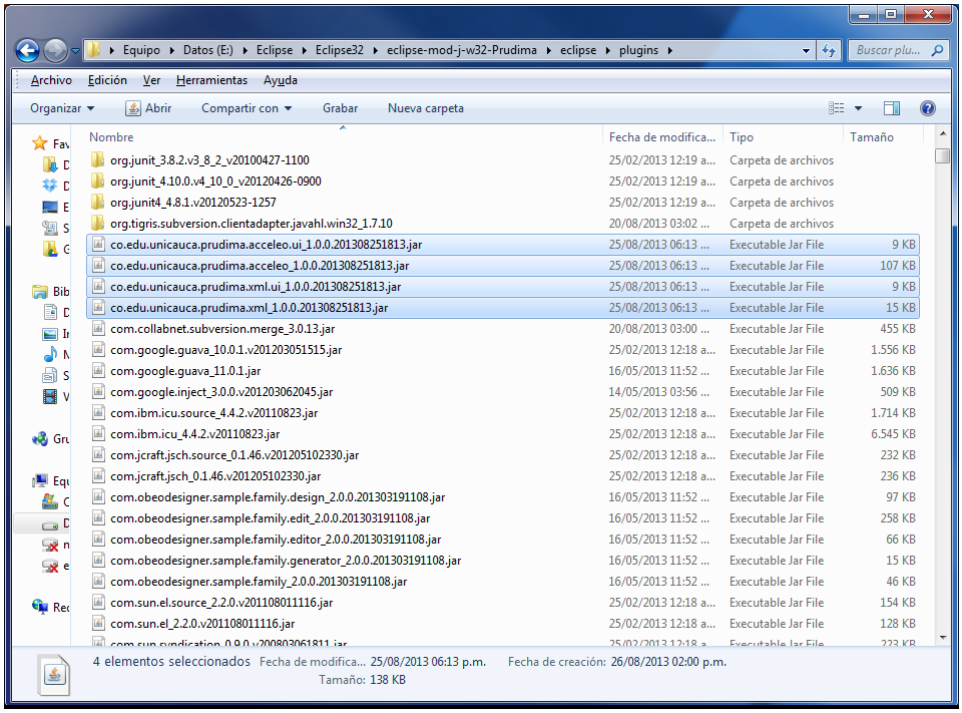


Figura 43 - Carpeta plugins de Eclipse adicionar Plugins

2. Luego, debemos abrir el explorador de carpetas e ingresar a la ruta donde se encuentre instalada nuestra versión de eclipse, una vez ahí, ingresamos a la carpeta plugins, copiamos la carpeta *"co.edu.unicauca.prudima.libs"* de PRUDIMA dentro de esta carpeta. Como se muestra en la siguiente figura.
   Nota: Dentro de esta carpeta se encuentran las librerías:

   - commons-codec-1.3.jar
   - jade.jar
   - xmlpull-1.1.3.1.jar
   - xpp3_min-1.1.4c.jar
   - xstream-1.4.4.jar

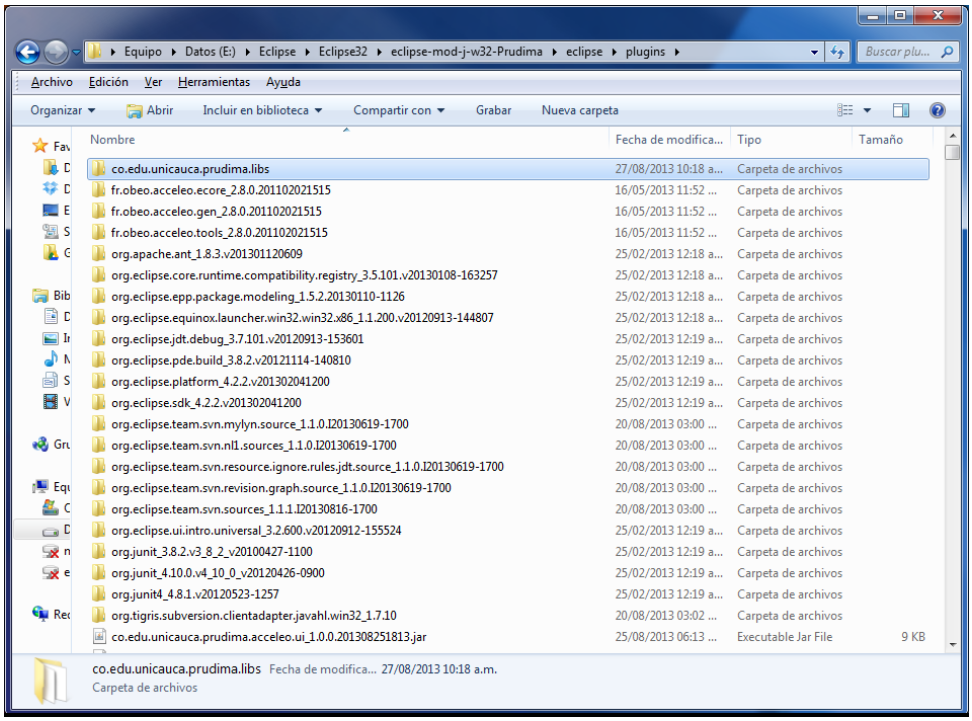


Figura 44 - *Carpeta plugins de Eclipse adicionar carpeta*

3. Se debe reiniciar la plataforma de desarrollo Eclipse para que el framework encuentre e integre los nuevos *plugins.*

REFERENCIAS

[1] M. Utting and B. Legeard, *Practical model-based testing a tools approach*, 1st ed. San Francisco: Elsevier Inc, 2006, p. 455.

[2] M. D. A. G. Version, A. Kennedy, K. Carter, and W. F. X. Technologies, “MDA Guide Version 1.0.1,” no. June, 2003.

[3] O. Object Management Group, “MOF 2.0 Query/ View/ Transformation.” 2005.

[4] I. A. Nantes, “Atlas Transformation Language. ATL User Manual,” 2006.

[5] E. S. Fundation, “The VIATRA2 Model Transformation Framework.”

[6] O. M. Group, “MOF QVT. Final Adopted Specification. ptc/05-11-01,” 2005.

[7] P. M. Steinberg Dave , Budinsky Frank, *EMF: Eclipse Modeling Framework (2nd Edition) 2nd Edition*. 2008, p. 744.

[8] *Eclipse Modeling Framework: A Developer’s Guide*. Addison-Wesley Professional, 2004, p. 680.

[9] J. Tretmans, “Model-Based Testing and Some Steps towards Test-Based Modelling,” in in *Formal Methods for Eternal Networked Software Systems*, vol. 6659, M. Bernardo and V. Issarny, Eds. Springer Berlin / Heidelberg, 2011, pp. 297–326.

[10] OMG, “UML Testing Profile.” 2005.

[11] Object Management Group, “Unified Modeling Language, Infrastructure,” 2011.

[12] Gineth Andrea López Hoyos, “Modelado y propuesta de Mejora de Procesos de Desarrollo para un entorno académico,” 2013.