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

Monografía para optar al título de
Ingeniero de Sistemas

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

Monografía para optar al título de
Ingeniero de Sistemas

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

NOTA DE ACEPTACION

_____
_____
_____
_____
_____

_____
Firma de Jurado:

_____
Firma de Jurado:

Popayán, Cauca 10 de octubre de 2013

AGRADECIMIENTOS

Doy gracias a mi papá Heriberto Gaviria Sanchez, por ser mi amigo y el constante apoyo de mi vida, a mi mamá María Elena Agredo Trujillo, que no está hoy entre nosotros, pero que todos los días está en mi mente y corazón, el amor que me dio a lo largo de su vida ha sido mi fuente de energía y ganas de seguir adelante. A mis hermanas Dory Esmeralda Gaviria Agredo y Janneth Cristina Gaviria Agredo por sus palabras de aliento día tras día. A mi novia Diana Maribel Pezo A., por toda su colaboración y apoyo incondicional. Y a toda mi familia que están siempre pendientes de mí.

Agradezco a mi amigo y compañero de tesis, Luis Carlos Pito, por su colaboración y disposición en este trabajo de grado.

Rubén Javier Gaviria Agredo

A Dios, por darme la vida, sus bendiciones diarias y el mejor regalo que podría tener, mi familia.
A mis padres, Sara y José por su apoyo incondicional en todas las decisiones de mi vida. Siempre creyendo en mí, dándome su amor, comprensión y paciencia.
A mis hermanas quienes siempre me hay acompañado en mis triunfos y fracasos.
A mi familia, a mi novia, amigos y personas que ya no están presentes en mi vida, por el apoyo incondicional y por todos los valores que infundieron en mí para formar un hombre de principios.

A mi compañero de tesis pero sobre todo amigo y compadre por su apoyo en todos los proyectos que se me ocurren.

A la Universidad del Cauca, compañeros de trabajo y docentes por las enseñanzas y momentos vividos, determinantes en mi vida profesional.

Luis Carlos Pito Díaz

Gracias al Ingeniero Pablo Magé y Julio Ariel Hurtado por la dirección de este proyecto, por la disposición que tuvieron y la ayuda en los momentos adecuados.

A nuestros compañeros de universidad por su colaboración en los momentos necesarios durante la realización de la investigación.

Y muchas gracias a todas aquellas personas que colaboraron o participaron, de una u otra forma, en la realización de este trabajo.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Actualmente el software y los servicios derivados que éste presta se tornan cada vez más críticos para la vida de los seres humanos [1], ya que están presentes directa ó indirectamente en muchas de las actividades diarias donde ponen a prueba la calidad de dichos productos y servicios. Estos se evalúan de manera intuitiva y subjetiva, dependiendo del nivel de satisfacción de quien los consume. La calidad es una cualidad que cualquier servicio o producto debe poseer y mantener para asegurar su permanencia en el mercado [2].

A través del tiempo, en los entornos académicos e industriales de software, se ha trabajado para tratar de mejorar los niveles de calidad en todos los procesos que se encuentren involucrados a la hora de generar productos software. Por medio de la experiencia obtenida en los proyectos exitosos y fallidos, ha ido evolucionando la calidad de una forma que hasta hoy en día, intenta ir en concordancia con las complejidades de los problemas que se resuelven por medio de la ingeniera de software. Distintas etapas en los procesos de calidad se fueron dando a medida que la complejidad en los proyectos de software fueron aumentando, cronológicamente se pueden distinguir así: el control de la calidad, el aseguramiento de la calidad, la gestión de la calidad y la calidad total. En esta última es donde cada vez se involucran más actores (individuos, organizaciones y consumidores) persiguiendo el mayor nivel de calidad tanto a nivel de producto como en los procesos de producción [3].

No obstante, los defectos, errores y fallas en el software seguirán existiendo, ya que no hay un proceso mágico que permita garantizar la ausencia de los mismos. Sin embargo, junto a la evolución en los conceptos de calidad, también surgieron en paralelo técnicas, métodos, metodologías y herramientas las cuales permiten llevar a cabo las actividades de verificación y validación que cada concepto conlleva, por ejemplo, a nivel de producto se crearon técnicas como pruebas de caja negra y blanca, pruebas de regresión, integración, conducidos por enfoques tanto manuales como automatizados y a nivel de procesos se crearon estándares como CMM, CMMI, ISO 9000, ISO 967, entre otros.

Capítulo 1

1. CONTEXTUALIZACIÓN

1.1. CONTEXTO DEL PROBLEMA

En la Universidad del Cauca el grupo de Investigación y Desarrollo en Ingeniería de Software (IDIS), viene trabajando en la mejora de procesos desde el año 2004 en proyectos como SIMEP-SW, COMPETISOF y en el proyecto "Entorno Colaborativo de Apoyo a la Mejora de Procesos para la Industria de Software Colombiana". Es así como surge la idea de aplicar las iniciativas de mejoramiento en el mismo contexto académico. El curso de sistemas distribuidos viene adelantando un programa de mejoramiento siguiendo el enfoque Agile SPI, obtenido en el proyecto SIMEP-SW, dentro del cual se ha identificado el proceso de pruebas como uno de los procesos claves a mejorar. Particularmente, dentro del entorno académico de desarrollo de software que se orienta en la asignatura de Laboratorio de Sistemas Distribuidos, dictada en el séptimo semestre del programa de Ingeniera de Sistemas de la Universidad del Cauca, se ha presentado la necesidad de implementar mecanismos que permitan automatizar el proceso de evaluación de prácticas realizadas por los estudiantes con el fin de disminuir la gran cantidad de tiempo que se invierte a las pruebas manuales que se deben realizar a cada grupo del laboratorio, buscando que el estudiante entregue un mejor producto.

1.2. PREGUNTA DE INVESTIGACIÓN

Por lo anterior surge la pregunta de investigación*: ¿Cómo incrementar los niveles de calidad en los productos de software en el contexto académico del laboratorio de Sistemas Distribuidos?*

1.3. APORTES DEL PROYECTO DE GRADO

El presente trabajo se enfoca en cómo incrementar los niveles de calidad de productos de software mediante la evaluación practica de las pruebas dirigidas por modelos, en inglés "*Model Based Testing - MBT*", soportadas por agentes software en el contexto de los sistemas distribuidos. Para ello, se siguió un enfoque MDE, en el cual se definió primero, un conjunto de metamodelos para modelar de

manera abstracta tanto los sistemas distribuidos como sus pruebas. Posteriormente, se definió un proceso práctico en el cual, se utilizan los metamodelos previamente construidos, para el correcto despliegue y análisis de resultados de las pruebas. Finalmente, se determina la validez del proceso junto a los metamodelos por medio de un estudio de caso, basado en un prototipo de herramienta CASE que utilizaron los *testers*, frente al enfoque manual que se utilizaba tradicionalmente. La herramienta CASE fue desarrollada en el marco de esta tesis.

1.4. OBJETIVOS

A continuación se describen el objetivo general y los objetivos específicos:

1.4.1. Objetivo general

- Contribuir a la mejora de los procesos de construcción de software, a través de la evaluación practica de las pruebas dirigidas por modelos dentro del ciclo de producción de software, para incrementar los niveles de calidad requeridos en las prácticas académicas de software distribuido.

1.4.2. Objetivos específicos

- Obtener un conjunto de meta-modelos que definan la sintaxis y semántica abstracta tanto para el modelado de sistemas distribuidos como para el modelado de los casos de pruebas de dichos sistemas.
- Definir un proceso práctico para las pruebas de sistemas distribuidos basándose en el uso de modelos construidos a partir de los metamodelos previamente obtenidos.
- Determinar la validez del proceso de pruebas dirigidos por modelos, a través de un prototipo de herramienta CASE y de modelos de casos de prueba previamente diseñados, en un entorno de ejecución soportado por agentes software, donde la herramienta realizará análisis y reportes de las pruebas, para medir el número de fallos presentados en el software en construcción.

1.5. ORGANIZACIÓN DEL DOCUMENTO

El presente documento se encuentra organizado por capítulos de la siguiente manera:

En el primer capítulo se contextualizó el problema, se definió la pregunta de investigación y los objetivos.

En el segundo capítulo se presenta el marco teórico, mostrando las bases conceptuales para el desarrollo del presente trabajo, y el estado del arte.

El tercer capítulo trata todo lo relacionado a la caracterización de PRUDIMA, donde se estableció la arquitectura, se especificaron y construyeron los meta-modelos para modelar las pruebas a los SD, se definieron los tipos de transformaciones utilizadas, su construcción e implantación y finalmente el proceso en el cual se definen las actividades para llevar a cabo el proceso de pruebas.

En el capítulo cuarto se detalla la implementación del prototipo de herramienta case PRUDIMA, basado en una metodología ágil de construcción de software.

En el quinto capítulo se presenta la preparación, evaluación y validación para el proceso de pruebas dirigidos por modelos a SD a través de en un estudio de caso.

En el capítulo sexto se presenta el análisis del cumplimiento de los objetivos.

Finalmente, en el capítulo séptimo, se presenta un resumen, contribuciones, limitaciones y los trabajos futuros.

Capítulo 2

2. MARCO TEÓRICO Y ESTADO DEL ARTE

Este capítulo contiene preliminares y definiciones de las áreas involucradas en el desarrollo del trabajo de grado, para esto se utilizó la metodología de investigación propuesta por Serrano [4] . En la sección 2.1 se tratan de manera general las áreas del conocimiento relacionadas con este trabajo, que corresponden a "*Pruebas de software*", "*Calidad de Software*", "*Ingeniería basada en modelos*", "*Sistemas distribuidos*" y "*Agentes Software*". En la sección 2.2 se brinda una visión global a las "Pruebas dirigidas por modelos". La sección 2.3 contiene el estado del arte acerca de "Pruebas dirigidas por modelos".

2.1. BASES CONCEPTUALES

Esta sección presenta definiciones y características relevantes de las áreas del conocimiento necesarias para el presente trabajo y como lo influyen.

2.1.1. Calidad de software

Este trabajo se centra en el uso del enfoque MBT, con el fin de aportar mejoras en la calidad del proceso de pruebas, dando como resultados mejoras en el producto final. Actualmente se presentan varias definiciones de calidad, una de las más aceptadas es: "la calidad es la c*oncordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente."* [5].

Dada la competitividad del mundo en el cual vivimos, los esfuerzos de la industria del software, por generar productos de buena confianza en su funcionamiento, genera la necesidad de concebir productos de buena calidad [6]. Es por esta razón que la industria del software adapta técnicas y métodos de calidad de productos manufacturados y los lleva a los productos software.

En esta búsqueda de generar productos con calidad, la industria de software adapta el concepto de "control de calidad*",* el cual pretende por medio de la inspección de los procesos, procedimientos, aplicación de técnicas y actividades de carácter operativo dentro del desarrollo de software, satisfacer las necesidades relativas a la calidad.

Posteriormente, este proceso evolucionó hacia el "aseguramiento de la calidad", el cual pretende brindar confianza al producto a través de las acciones planificadas y sistemáticas identificando errores de manera temprana.

Pasado algún tiempo, el aseguramiento de la calidad se convierte en una sub etapa de la "gestión de calidad", la cual busca implementar técnicas y herramientas de inspección en diferentes niveles como son: a nivel del producto, nivel de proyecto y nivel de proceso.

Finalmente, el concepto de "calidad total" que abarca los anteriores, es la evolución de la continua mejora de estándares, estrategias y modelos de calidad, enfocada en la satisfacción del cliente por medio de las mejoras permanentes del aspecto organizacional y de los procesos de toda la empresa [3].

Los expertos en calidad, afirman que un producto de calidad compensa la inversión realizada por medio de la reducción del número de defectos del producto, disminuyendo también el trabajo o esfuerzo realizado en las correcciones, logrando mejorar la productividad de la empresa.

Este trabajo se enfoca en la gestión de la calidad, ver Figura 1, dentro de la cual se sitúa a nivel de gestión de producto y más específicamente en las pruebas funcionales, utilizando el enfoque guiado por modelos. Esto se realiza con el fin de encontrar defectos y problemas de manera temprana, evitando así, los costos asociados en la corrección de los mismos en un estado avanzado del proyecto de desarrollo. Así, la posible mejora a nivel de producto, a su vez conlleva la mejora en todos los niveles interrelacionados [7].

A continuación, se revisará algunos conceptos básicos en el área de pruebas software.

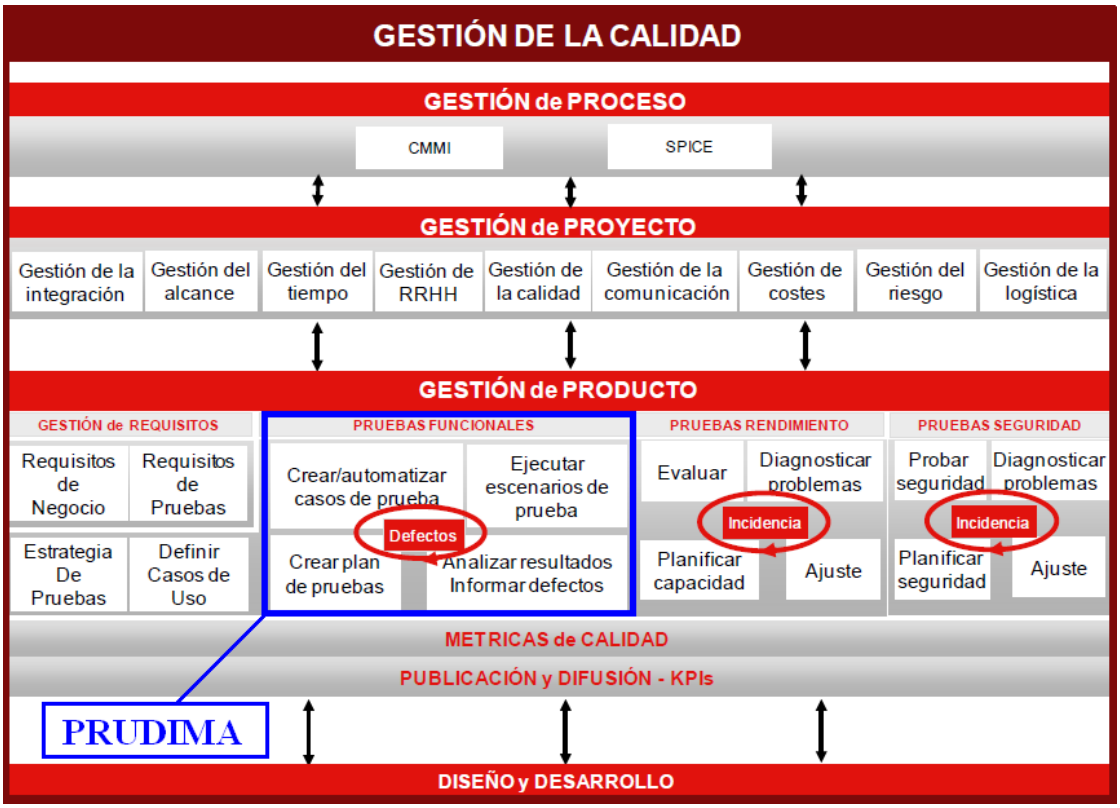

Figura 1- Foco de PRUDIMA dentro de la gestión de calidad – Original [7]

2.1.2. Pruebas Software

La evolución de las pruebas software, ver Figura 2, se han realizado tanto en su entorno industrial como en el entorno académico, partiendo del proceso de pruebas manuales (*Manual Testing Process*) y su posterior evolución a los procesos de automatización de pruebas por medio de Scripts y finalmente llegando al proceso de pruebas basado en modelos, ver Anexo A.

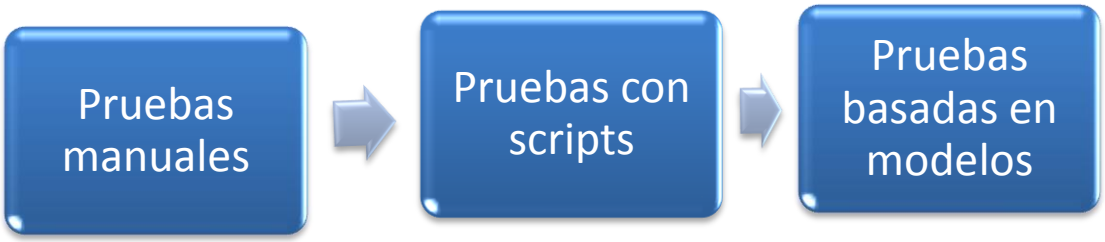

Figura 2 - *Evolución de pruebas software*

Existe una tendencia a creer que las pruebas realizadas al software son solo para encontrar los fallos que este posee. Lo que en realidad hacen las pruebas

software es encontrar defectos provocados por algún error. Luego, en el análisis de dicho error y al depurar el código fuente del software, es cuando se puede descubrir el fallo que lo desencadena y su posterior consecuencia. Por ello, las pruebas de software no garantizan la ausencia de errores, su único objetivo es poder detectar los defectos.

2.1.3. Ingeniería basada en modelos

La ingeniería dirigida por modelos (MDE) es un enfoque novedoso que busca facilitar el proceso de desarrollo de software a través de la reutilización del conocimiento de refinamiento. Aunque las tecnologías han evolucionado rápidamente en forma positiva, también lo han hecho, las necesidades de los usuarios y por tanto se requiere de software cada vez más complejo, mantenible, con una alta calidad de servicio (QoS) y con rápida salida al mercado. MDE aborda el problema con la combinación de tecnologías como: los "lenguajes de modelado específicos de dominio" (DSML) y los "*motores de transformación y generación*" [8][9]. Esta área presenta una gran proyección, pero aún se encuentra en desarrollo, puesto que no existe un estándar para el óptimo manejo de este enfoque.

Los DSML definen relaciones entre conceptos de un dominio específico [8][10] y basándose principalmente en la abstracción de la complejidad de un sistema, por medio de metamodelos, los cuales, construir un conjunto de modelos que permitan visualizar parcial o totalmente el sistema, permitiendo comprender de una manera relativamente más sencilla dicho sistema. Por otra parte, los motores de transformación y generación permiten generar distintos tipos de artefactos como: código fuente, ambientes de despliegue, datos de entrada para pruebas, entre otros; condensando ciertos aspectos de un modelo de entrada, por medio de reglas o mapeos específicos para cada concepto del sistema modelado.

Arquitectura dirigida por modelos

La necesidad de gestionar y controlar los procesos software es el principal factor que conduce a la necesidad de establecer mecanismos de desarrollo que acorten los ciclos de vida y facilita la portabilidad, la interoperabilidad y reusabilidad [11].

Hoy en día muchas de las metodologías de desarrollo de software se basan en la utilización de modelos[1] como la principal herramienta a la hora de describir y mantener el desarrollo del sistema. A esta tendencia se le conoce como *Model Driven Developmet* (MDD), el cual proporciona una estrategia general de realizar desarrollos por medio de la generación de modelos y trasformación de los mismos; pero esta no define técnicas ni fases a seguir. Es por esto, que a partir de MDD se

[1] Los *modelos* son representaciones formales del comportamiento o estructura del sistema que se está construyendo. Frecuentemente se puede presentar como una combinación de dibujos y texto.

aplican enfoques como la arquitectura dirigida por modelos (*Model Driven Architecture* - MDA) iniciativa del *Object Management Group* (OMG) [12], la cual está centrada en los diferentes modelos que el sistema genera y las trasformaciones que estos puedan tener siguiendo la filosofía de MDD, además se soporta en los estándares como *Unified Modeling Lenguage* (UML), *XML Metadata Interchange* (XMI), *Meta Object Facility* (MOF) [13] y *Common Warehouse MetaModel* (CWN).

MDA establece que se pueden conseguir beneficios importantes a nivel de: productividad, portabilidad, interoperabilidad, mantenimiento y documentación. También plantea una serie de trasformaciones a los modelos presentados en la Figura 3 que compone el sistema, estos modelos se describen a continuación:

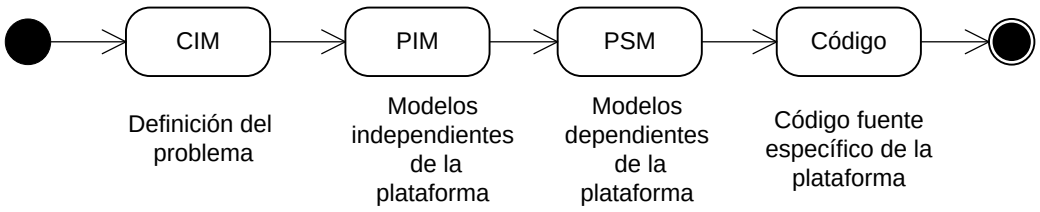

Figura 3 - *Trasformaciones de los modelos que componen el sistema. [12]*

Para generar los modelos *Computional Independient Models* (CIM) primero se debe puntualizar y escribir el problema, mediante la definición de las funciones que el sistema debe realizar sin mezclar la implementación de la solución. Para expresar las funcionalidades del sistema se pueden utilizar los diagramas de casos de uso, los cuales intentan aclarar los principales requisitos del sistema.

Una vez generados los modelos CIM, se puede realizar la trasformación a los modelos *Plataform Independent Models* (PIM). Los modelos PIM permiten mostrar una vista centrada en la operación del sistema sin los detalles necesarios para una determinada plataforma [12] .

Después de obtener los modelos PIM, se realiza una nueva transformación para generar los modelos *Plataform Specific Model* (PSM), los cuales, combinan las especificaciones expuestas en los PIM como los detalles particulares de la plataforma a desarrollar, agregando los complementos que hagan posible la implementación de código de la aplicación a partir de este.

Las anteriores trasformaciones entre modelos juegan un papel importante en el desarrollo dirigido por modelos, ya que permiten llegar hasta el código fuente de una aplicación software en una plataforma concreta desde su especificación independiente de la plataforma [14]. Existen varias formas de realizar la transformación de los modelos, entre ellos tenemos el estándar de *Query Views*

Transformtion (QVT) [13], *Visual Automated Model Transformation* (VIATRA) [15] o por *ATLAS Transformation Language* (ATL) los cuales se detallan posteriormente.

Para dar apoyo al enfoque MDA, se han desarrollado diferentes herramientas como: *ArcStyler*[2]*, OptimalJ*[3]*, AndroMDA*[4]*, Codagen Architect*[5]*, Together Architect*[6]*,* entre otras, que están en el ámbito de las herramientas comerciales y no comerciales. Sin embargo muchas de estas herramientas ,no contemplan en su totalidad las fases que se han definido para este enfoque y además cada herramienta tiene una manera diferente de interpretarlas [16].

Lenguaje de modelado unificado

El lenguaje de modelado (*Unified Modeling Lenguage* - UML) es una especificación de la OMG. Este lenguaje estándar se utiliza para escribir planos del software, además de proporcionar un conjunto de mecanismos para visualizar, especificar, construir y documentar los artefactos de un sistema a través de varios diagramas que expresan aspectos estáticos y dinámicos de una aplicación. Los aspectos estáticos están relacionados con la estructura del sistema, el propósito es describir entidades del sistema y cómo están relacionadas. Por otro lado, se encuentran los aspectos dinámicos los cuales se refieren a la interacción de la aplicación la creación y destrucción de objetos, sus conexiones con el tiempo y las transformaciones de los estados de estos objetos [17].

La versión actual 2.1 de la especificación de UML está compuesta por cuatro partes [18]:

- ***UML Superstructure*:** que especifica el lenguaje desde el punto de vista de los usuarios finales. Define seis (6) diagramas estructurales, tres (3) diagramas de comportamiento, cuatro (4) diagramas de interacción sí como los elementos que los componen.
- ***UML Infrastructure*:** Especifica la construcción de las bases de UML. Para esto define un núcleo de un metalenguaje que podrá ser utilizado para definir los metamodelos de otro lenguaje.
- ***UML Object Constraint Language*:** OCL es un lenguaje que permite definir una amplia semántica de modelos UML mediante la definición de precondiciones, post-condiciones, invariantes y otras condiciones.

[2] **ArcStyler** Información: http://www.omg.org/mda/mda_files/P2A_Tutorial.pdf
[3] **OptimalJ** Project: http://www.componentsource.com/beaws/products/compuware-optimalj-professional-eclipse-named-users/index.html
[4] **AndroMDA** Project: http://www.andromda.org/index.html
[5] **Codagen Architect** Información: http://www.omg.org/mda/mda_files/CodagenMDA-Final.pdf
[6] **Together Architect** Project : http://www.borland.com/products/together/

- ***UML Diagram Interchange*:** Extiende el metamodelo de UML a otro paquete adicional que modela información de carácter grafico asociada a los diagramas, permitiendo el intercambio de modelos conservando su representación original.

Transformación de modelos

La transformación de modelos hace referencia al proceso de convertir un modelo en otro del mismo sistema [14]. Es por esto, que dentro del desarrollo dirigido por modelos, las transformaciones juegan un papel clave a la hora de especificar y generar el sistema en una plataforma concreta.

Las transformaciones se pueden clasificar en dos tipos: de *modelo a modelo* (del inglés, *Model to Model - M2M*) las cuales parten de un modelo origen y pretenden llegar a un modelo de destino, aumentando o disminuyendo el nivel de abstracción según sea el caso. Por otro lado, tenemos las transformaciones de Modelo a Texto (Model to Text - M2T) las cuales parte de un modelo origen y llegan a generar un archivo con cierto formato o código fuente de un lenguaje en particular.

Dentro de las trasformaciones de modelos, se analizan las siguientes aproximaciones: *Query/Views/Transformation*(QVT), *ATLAS Transformation Lenguage* (ATL), *Acceleo*, *Visual Automated model Trasnformation* (VIATRA) , Motores de plantillas, Motor de Transformaciones de *BOA 2*. Las cuales se describen en el Anexo B.

2.1.4. Agentes software

Un agente software “es un sistema informático que está situado en algún ambiente, y que es capaz actuar con autonomía en este entorno con el fin de cumplir con sus objetivos de diseño”[19]. La arquitectura del prototipo PRUDIMA construido por los autores del presente trabajo, utilizan los agentes software para soportar el enfoque MBT, tomando como base una plataforma de agentes software que permite el despliegue del entorno de ejecución de pruebas, donde los agentes son actores reales del sistema bajo prueba.

Debido a la autonomía de entorno, que tienen los agentes software son capaces de emular los clientes del SD, teniendo la posibilidad de realizar invocaciones remotas al servidor que en sí mismo podría ser también un agente. De esta manera PRUDIMA pretende poder automatizar el proceso de pruebas en un entorno muy parecido al real.

2.1.5. Sistemas distribuidos

Un Sistema Distribuido (SD) es “aquel en el que los componentes situados en una red de computadores, comunican y coordinar sus acciones sólo por el paso de

mensajes. Esta definición conlleva a las siguientes características especialmente significativas de los SD: concurrencia de los componentes, la falta de un reloj global y fallos de los componentes independientes" [20].

Por lo anterior, se tiene que un SD consiste en un conjunto independiente de componentes corriendo de manera concurrente en diferentes maquinas e interactuando entre si, comunicados por medio de una red para logra un objetivo en común. Debido a esto se presentan una serie de problemas, haciendo la tarea de realizar las pruebas al SD más compleja, algunos de estos problemas son: escalabilidad de los criterios pruebas, generación de datos de prueba, pruebas redundantes, mecanismo de seguimiento y control en las pruebas, reproducibilidad de eventos, interbloqueos y condiciones de carrera, pruebas para la escalabilidad y el rendimiento del sistema además de la prueba de tolerancia a fallos [21]. Este conjunto de problemas no serán abordados en el presente trabajo porque no están contemplados en el alcance del mismo, pero se analizara la conformidad de las pruebas contra los requerimientos del sistema bajo prueba (SUT).

2.2. PRUEBAS DIRIGIDAS POR MODELOS

Las pruebas dirigidas por modelos se refieren a la derivación, ejecución y análisis de casos de prueba contra un sistema software, donde la derivación de los casos de prueba se realiza a partir de modelos formales o informales que especifican el sistema, y cuyo fin es medir la cantidad de errores o precisar la ausencia de estos.

2.2.1. Definiciones para pruebas dirigidas por modelos

Con el propósito de formalizar el concepto de pruebas dirigidas por modelos (*MDT, Model-Driven-Testing*) o MBT, en el cual, el presente trabajo no hace distinción entre las palabras "dirigidas" y "basadas", se presentan a continuación las definiciones obtenidas durante la investigación, yendo de las más simples hasta las más elaboradas:

- *Binder* [22] define como el "uso de un sistema software, que representa aspectos abstractos de un sistema bajo prueba para generar casos de pruebas".
- *Utting et al.* [23] definen las pruebas basadas en modelos como "la derivación automática de casos de pruebas concretas desde modelos formales abstractos y su ejecución".
- *Dias* [24] definen las pruebas basadas en modelos como "una técnica de pruebas destinada a la generación automática de pruebas con modelos extraídos de los artefactos software producidos a lo largo del proceso de desarrollo".
- *Tretmans et al.* [25] define que la especificación de pruebas dirigidas por modelos, es "probar la especificación del modelo del sistema, así como el sistema resultante automáticamente por medio de casos de prueba que son

derivados desde el modelo de especificación del sistema y los requerimientos del mismo".

2.2.2. Objetivo general de pruebas dirigidas por modelos

A nivel general el objetivo que persigue MDT es permitir la generación, ejecución y análisis automático de casos de prueba, por medio de algoritmos de generación que se basan en modelos estructurales o de comportamiento del sistema, o ambos, para mejorar la calidad y efectividad de las pruebas, además de la reducción de costos de las mismas y como consecuencia de lo anterior, mejorar la calidad del sistema en desarrollo.

Las pruebas dirigidas por modelos inician con un modelo verificado de un sistema, para luego intentar demostrar que la implementación real del sistema se comporta de acuerdo con el modelo [25].

2.2.3. Tipos de enfoques de pruebas dirigidos por modelos

Al aplicar MBT se debe definir su enfoque, ya que va de la mano con los requerimientos del proyecto de software a desarrollar y al cual se le aplicará esta técnica de pruebas. En la literatura [22], [26], [27] se ha caracterizado MBT realizando una revisión sistemática de los trabajos más representativos en esta área, y dentro de los cuales se ha definido el siguiente conjunto de campos que permiten definir el contexto de MBT: dominio del software, nivel de automatización, nivel de pruebas, herramientas de soporte, comportamiento del modelo, modelos usados para la generación de los casos de prueba, criterio de cubrimiento de pruebas, criterio de generación de casos de prueba, técnicas de pruebas. Ver Anexo C.

2.2.4. Ventajas y desventajas de las pruebas dirigidas por modelos

Ventajas

El objetivo final de cualquier tipo de pruebas software es encontrar los fallos en el software bajo prueba, así es como MBT según los estudios previos, es mejor que los enfoques de pruebas manuales, detectando el mismo o más número de fallos. La generación automática de casos de prueba es una de las bondades que realza a MBT por encima de los otros enfoques, pero también la detección temprana de defectos como lo muestra en [28] y poder ajustar requerimientos software que estén pobremente especificados por medio del modelo de pruebas [29].

La reducción de costos y tiempo, es otra característica de MBT, dado que permite generar un gran número de casos de prueba a partir de un modelo y su

ejecución y análisis automáticamente, ayudando a reducir el tiempo de construir baterías de pruebas y el costo requerido por cada *tester* que diseñe, ejecute y analice las pruebas manualmente [30].

En los enfoques manuales el proceso de razonamiento que conlleva el diseño de pruebas está ligado a la experiencia y capacidad de los ingenieros de pruebas, de su percepción del sistema y posiblemente de sus emociones, y es por esto que el proceso de pruebas manual es propenso a errores. En MBT el proceso de pensamiento queda plasmado en un modelo el cual puede ser validado por los *testers* y técnicas de chequeo de modelos como en [31][32], además que este proceso puede ser reproducido sistemáticamente.

Desventajas

Una de las desventajas por las cuales actualmente el enfoque MBT no está ampliamente aplicado en la industria es porque ésta lo ve costoso y complejo de adoptar. Además de la falta de herramientas software que permitan utilizar, de una manera relativamente sencilla, el proceso de MBT sin requerir demasiadas tareas manuales por parte de los diseñadores de las pruebas; herramientas que utilicen lenguajes de modelado conocidos como UML, lenguajes B y Z entre otros; para evitar la carga cognitiva de aprender nuevas o complejas notaciones a la hora de modelar [27]. A pesar de esto algunas compañías están impulsando su utilización con el desarrollo de sus propias herramientas como Microsoft con “Spec Explorer” [33] el cual lo utiliza hoy en día para probar su software “Skype” y “Lync”.

Otro de los limitantes de la adopción de MBT es la falta de metodologías o guías para la integración con los proceso de desarrollo de software. En [34] se presenta un estudio de cómo integrar el proceso de desarrollo dirigido por modelos (MDD) y MBT, y los beneficios que esto conlleva.

Las limitaciones propias de los modelos son otra cuestión a tener en cuenta a la hora de emplear MBT, es aquí donde se debe evaluar las características del lenguaje que se utiliza para describir el modelo, para saber si se tienen todos los recursos suficientes y necesarios para llevar a cabo la automatización. Por otro lado, usualmente los modelos no representan o abstraen requerimientos no funcionales como: usabilidad, seguridad, confiablidad, concurrencia, entre otros, y tampoco se los prueba. No obstante, algunos trabajos intentan cubrir estas brechas como en [35].

2.2.5. Modelos

En la ingeniería de software se usan cada vez más modelos para abordar la complejidad de los problemas que se requieren solucionar con sistemas software. Un modelo es “*una descripción o representación de un sistema software o su*

entorno para cierto propósito desarrollado usando un lenguaje de modelado" [36]. Es a partir de la *abstracción* que se pueden obtener como resultado modelos simplificados de un problema complejo. Las simplificaciones hechas a los modelos pueden ser de dos tipos: donde la falta de información puede ser insertada automáticamente y donde la información se excluye a propósito para mantener simple el modelo. En MBT existen dos modelos, principalmente el del SUT y el modelo de pruebas. Para obtener dichos modelos existen cuatro principios de abstracciones que se utilizan para MBT [37] :

- ***Abstracción funcional*:** se concentra en los requerimientos principales del SUT que deben ser verificados.
- ***Abstracción de datos*:** aquí se mapean los tipos de datos concretos a tipos lógicos o abstractos para lograr una representación compacta o una reducción de la complejidad a nivel abstracto.
- ***Abstracción de comunicación*:** aquí una interacción compleja a un nivel concreto se abstrae a una operación o un mensaje en el nivel más abstracto tratándolo atómicamente.
- ***Abstracción temporal*:** se da importancia únicamente al orden en el cual ocurren los eventos, sin tener en cuenta las posibles disparidades de tiempo entre componentes de un sistema.

Metamodelo (MM) es la manera formal de llamar a los lenguajes de modelado, los cuales describen la sintaxis abstracta de estos. *MOF* y *Ecore* son ejemplos de metamodelos, de los cuales se pueden derivar otros MM que permiten enfocarse en problemas más específicos. Los MM se clasifican en dos categorías [10]:

- ***Basados en conexión*:** poseen un conjunto de objetos interconectados donde la distribución o posición espacial de los objetos no es relevante.
- ***Basados en geometría*:** contienen un conjunto de objetos visuales organizados espacialmente en un plano cartesiano. En el cual, tanto su posición como dimensiones son importantes, donde las diferentes reglas de composiciones espaciales pueden ser utilizadas para formar instancias visuales como adyacencia, concatenación o exclusión.

Como analogía con los sistemas orientados a objetos, los modelos vienen a ser instancias de un MM, así como los objetos son instancias de las clases que los describen.

Los modelos generalmente pueden ser descritos por medio de textos o gráficos, los cuales brindan una visión abstracta, que puede ser completa o parcial del sistema modelado. Cuando son gráficos se pueden tener uno o varios diagramas como en el caso de UML. Los insumos para el proceso mental de modelar pueden obtenerse de diferentes fuentes tanto de especificaciones formales o informales, como de la propia experiencia en resolución de problemas similares.

En [37] se definen tres características que debe poseer un modelo: *mapeo* – los modelos son modelos de algo, *simplificación* – los modelos no capturan todos los atributos de lo que se modela y que sean *pragmáticos* – los modelos no están relacionados con los originales respectivos en un sentido único, pero cumplen ciertos objetivos. Además de las anteriores características, cuando se habla de construcción de software dirigido por modelos, existe un factor clave respecto a los modelos y es que deben poseer ciertos atributos de calidad para poder ser usados de manera satisfactoria. Según [36] existen seis atributos de calidad que debe cumplir un modelo:

1) **Correcto**: su definición es compuesta.
    a. "Incluir los elementos y relaciones correctas entre estos e incluir sentencias correctas acerca del dominio".
    b. "No violar las reglas y convenciones".
2) **Completitud**: "Tener toda la información necesaria que es relevante y ser lo suficientemente detallada de acuerdo a los propósitos del modelado".
3) **Consistencia**: "Se define como no tener contradicciones en el modelo, Esto cubre vistas o diagramas que pertenecen al mismo nivel de abstracción o fase de desarrollo (consistencia horizontal), y entre modelos o diagramas que representan el mismo aspecto, pero en diferentes niveles de abstracción o diferentes fases de desarrollo (consistencia vertical) y también cubre la semántica entre los modelos".
4) **Comprensibilidad**: "Es definida como se entendible por los usuarios previstos, ya sean humanos o herramientas".
5) **Confinamiento**: "Se define como el acuerdo con el propósito del modelado y el tipo de sistema, tales como incluir diagramas relevantes y estar en el nivel correcto de abstracción. Un modelo es una descripción de la cual se ha eliminado detalle intencionalmente. Un modelo confinado no tiene información innecesaria y no es más complejo o detallado que lo necesario".
6) **Mutabilidad**: "Se refiere al soporte de cambios o mejoras para que los modelos puedan ir cambiando o evolucionado rápidamente y continuamente".

Los usos dados a los modelos son variados, desde simplemente conceptualizar un problema, hasta construir enteramente un sistema software. Recurriendo de nuevo a UML, en este se tienen diferentes usos para los modelos, por ejemplo: "UML como esqueleto del sistema", utilizaría un diagrama de dominio para definir los componentes conceptuales y sus relaciones dentro del sistema. "UML como plano del sistema", podría usar los diagramas de clases y de componentes donde se tiene una vista más detallada y organizada del sistema. Finalmente "UML como lenguaje de programación" emplearía los diagramas de colaboración o secuencia,

en conjunto con los anteriores, los cuales contienen especificado completamente cada detalle para que permita la generación automática de código fuente [38].

2.2.6. Generación de casos de pruebas

El proceso de pruebas con el enfoque tradicional o manual es propenso a errores, por eso la generación de pruebas debería ser automática para ser efectiva y repetible. La generación de casos de prueba permite de manera automática crear un conjunto de pruebas ejecutables para el SUT, a partir del modelo del mismo, utilizando algoritmos de generación y criterios de selección de pruebas. Con un conjunto de pruebas obtenido, se intenta cubrir todos los posibles casos o estados de un sistema en ejecución. Sin embargo, dependiendo de la complejidad del SUT, algunas veces es imposible cubrir todos los caminos, ya sea porque tienden al infinito o porque pueden existir estados del SUT que quizás nunca ocurran.

Existen diferentes paradigmas para la generación de pruebas como el aleatorio, generación de test simbólicos y la generación dinámica [39], dentro de los cuales se utilizan diferentes técnicas entre las que se encuentran “pruebas basadas en búsquedas (*search-based testing*)”, “algoritmos genéticos”, “algoritmos evolutivos”, entre otros, para lograr la mayor cobertura posible.

Los criterios de cubrimientos son una técnica estándar para la generación automática de prueba, donde un criterio de cubrimiento representa un conjunto finito de objetivos de cubrimiento. El presente trabajo utiliza los criterios estructurales, basados en fallos, basados en requerimientos y el criterio explicito, además el cubrimiento puede orientarse hacia el modelo, en el cual el presente trabajo se enfoca, o al código del SUT, estos y los principales criterios de selección de casos de prueba utilizados en MBT son detallados en [30]:

- **Criterio basado en estructura**: trata de cubrir todos los controles de flujo que puedan existir en el modelo.

- ***Criterio basado en datos:*** cubre lo relacionado a los datos de entrada de cada transición en el modelo

- **Criterio basado en fallos**: intenta descubrir los fallos que a priori se pueden prever que contenga el sistema, para esto se generan un conjunto de casos de pruebas que prueben el modelo y por correspondencia el SUT tendría los mismo errores
- **Criterio basado en requerimientos**: genera un conjunto de casos de prueba que permita asegurar que todos los requerimientos informales son probados.

- **Criterios explícitos**: aquí un ingeniero de pruebas puede decidir sobre qué prueba o conjunto de pruebas deberían generarse desde el modelo, para cumplir con los objetivos de cubrimiento.

- **Criterio basado en estadísticas:** se generan casos de prueba aleatoriamente a partir de modelos que permitan el manejo de probabilidades dentro del mismo, como las cadenas de *Markov* (en inglés Markov chains).

2.2.7. Oráculos de pruebas

Los oráculos de pruebas son los árbitros que permiten saber, si un caso de prueba tuvo o no éxito. En un sentido más técnico son el componente que permite obtener las salidas de un caso de prueba del SUT y compararlas contra las esperadas o especificadas en el caso de prueba [40].

La definición del oráculo, como se menciona anteriormente es el encargado de evaluar y comparar las salidas reales del software con las salidas esperadas. El enfoque de pruebas manuales se basa en oráculos manuales, éstos se caracterizan principalmente por ser costos, además de poco fiables ya que la comparación está sujeta a la percepción del evaluador. Aunque se han realizado varias investigaciones para generar oráculos automáticos, ninguna de ellas, podría automatizar de manera completa todas las actividades del oráculo, en todas las circunstancias que pueda tener el software [41].

2.3. ESTADO DEL ARTE DE PRUEBAS DIRIGIDAS POR MODELOS

Para abordar el presente proyecto se realizó el estado del arte de los trabajos realizados en un rango de tiempo, desde la escritura de esta monografía hasta 10 años atrás, para conocer las principales tendencias y formas de aplicación de MBT dentro del marco de desarrollo de SD tanto en el entorno académico como industrial.

Para lo anterior, se realizaron búsquedas de proyectos de investigación utilizando bibliotecas virtuales especializadas como: *IEEEXplorer*, *ACM Portal*, *Science Direct, Springer Link,* entre otras. También se realizó la búsqueda en los sitios web de conferencias de MBT.

A continuación se realiza una sub-clasificación de los trabajos según el paradigma de comunicación utilizado por el SD que fue objeto de estudio y al cual le fue aplicado MBT [20], las categorías son las siguientes:

- Comunicación interproceso
- Invocación remota
- Comunicación indirecta

SUB-CLASIFICACIÓN SEGÚN PARADIGMA DE COMUNICACIÓN DEL SD

Comunicación interproceso

En [42], [43] se ha aplicado el enfoque MBT para probar SD que utilizan una comunicación entre los procesos a un nivel primitivo, es decir sin el uso de middlewares, en algunas implementaciones llegando a establecer el nivel de transporte dentro de la pila TCP/IP, e incluso a probar componentes hardware que se comunican con un protocolo específico.

A pesar de que MBT es aplicable a este tipo de software distribuido y que hay bastante trabajo por realizar en este campo, el presente trabajo se centrará en un nivel más alto de abstracción de software, para apoyar el uso de MBT en los procesos de construcción de software actuales.

Invocación remota

Como se pudo evidenciar en los casos de estudio de los trabajos [28], [44], [45], [46], [47], [48], se aplica MBT para software usado ampliamente en la red de redes (internet) y sobre los cuales se han construido muchos tipos de software utilizando el paradigma de invocación remota. Entre los tipos de software distribuido que se prueban se encuentran por ejemplo: en infraestructuras de comunicación vía correo electrónico, despliegue de servicios web, telemedicina, entre otros. Cabe resaltar que este tipo de software distribuido es utilizado cada vez más y se ha convertido en una parte importante de la vida cotidiana de las personas.

Dado lo anterior el presente trabajo centra su caso de estudio, en el uso de esta sub-categoría de paradigma de comunicación, ya que contribuye en un sector importante de la construcción de software, y más específicamente en el ámbito académico de la asignatura de SD de la Universidad del Cauca en el cual, se enseña este tipo de paradigma, además de aportar al proyecto de mejoramiento de los procesos de construcción de software.

Comunicación indirecta

En los siguientes trabajos [49], [50], [51], [52], [53], se muestra cómo el uso de MBT es válido para este enfoque de SD, aunque se debe considerar que el proceso de pruebas es mucho más complejo, ya que se deben tener en cuenta múltiples factores, entre los cuales se tienen la cantidad de nodos que pueden interactuar en un solo caso de prueba y que deben ser manejados individualmente de manera que se pueda hacer un reporte adecuado de los posibles defectos encontrados en cada uno. Otro factor sería, la concurrencia que deben soportar los componentes del sistema, ya que el entorno de pruebas debería desplegarse

adecuadamente para el soporte dicha concurrencia y no sea él mismo quien genere los errores.

Se razona que este tipo de software está en constante evolución, pero debido a las complejidades involucradas en el desarrollo de un prototipo que permita verificar la aplicación de MBT, no es viable para este proyecto centrarse en este tipo de software distribuido.

Una vez agrupados los 12 trabajos de investigación, se procede con una revisión más detallada de los mismos, analizando para cada uno de ellos la hipótesis, metodología, los conceptos y tecnologías utilizadas para lograr los resultados del estudio, como también las brechas de cada uno de estos trabajos. Esto permitió identificar características comunes, por ejemplo, que en todos se necesitan lenguajes de transformación para convertir un modelo origen a otro modelo destino. De esta manera, se fue obteniendo retroalimentación de un trabajo a otro y se pudo establecer los siguientes criterios de comparación de trabajos:

- **Lenguaje de transformación**: clasifica con que lenguaje de transformaciones se concretizan las pruebas software.
- **Uso de agentes para interactuar con el SUT**: especifica si el trabajo utiliza agentes software para interactuar con el SUT.
- **Herramientas soporte**: se discriminan los proyectos que han realizado sus propias implementaciones de MBT contra los que utilizan herramientas de terceros para sus proyectos, con el fin de obtener un panorama global en el uso de herramientas para la realización del caso de estudio del presente trabajo. Los posibles valores para este criterio es dado por el nombre del proyecto seguido de las siglas que correspondan según quienes hayan desarrollado la herramienta (desarrollo a medida (DAM), herramienta de terceros (HDT)).
- **Implementación de la propuesta:** esta columna indica si la herramienta fue desarrollada por los autores de la propuesta.
- **Lenguajes de especificación de modelos**: especifica qué lenguaje se utiliza para especificar el modelo del SUT o del modelo de pruebas, para conocer los lenguajes de especificación de modelos con mayor uso.
- **Modelos o diagramas**: describe los modelos utilizados para realizar las pruebas por medio del enfoque MBT.
- **Tecnologías de SD aplicadas**: determina cuales son las tecnologías asociadas a los SD que son aplicadas en las herramientas de MBT. Los posibles valores son dados según el paradigma de comunicación de los SD.

En la Tabla 1, se comparan los trabajos de investigación con los anteriores criterios. Se establece la siguiente nomenclatura para algunos valores de celdas:

- **HDT:** Herramientas desarrollada por terceros
- **HDM:** Herramientas desarrollada a medida.
- **CIP:** Comunicación interproceso
- **CI:** Comunicación indirecta
- **IR:** Invocación remota

Como se observa en la Tabla 1, en los trabajos de investigación relacionados, se ve una tendencia a utilizar lenguajes orientados a objetos como Java y C++, al ser lenguajes de alto nivel que permiten tener un amplio conjunto de librerías que pueden ser utilizadas para propósitos específicos de las transformaciones.

En cuanto al uso de agentes para interactuar con el SUT, la tendencia es muy baja, sin embargo se utilizan otros mecanismos como lo son procesos independientes en las maquinas que componen los sistemas distribuidos bajo prueba.

Dentro de las herramientas de soporte de MBT se ve una tendencia a realizar un desarrollo a la medida que se ajusten a la metodología o propuesta realizada. Esto facilita la implementación de los casos de estudio aunque al mismo tiempo genera una variedad de herramientas.

Dentro de las tecnologías de SD aplicadas se encuentra una tendencia a utilizar la tecnología de invocaciones remotas, ya que esta es altamente utilizada dentro de la industria como mecanismo de comunicación entre los componentes de un SD.

Dado el anterior análisis de los criterios evaluados se decidió realizar esta investigación enmarcada en la búsqueda de un mecanismo de pruebas bajo el paradigma MBT en SD. Se seleccionó un estudio de caso, donde se pueda aplicar dentro de las etapas tempranas del desarrollo del software, comunicarse por medio de interfaces y utilizar la invocación de métodos remotos como tecnología de SD aplicada, para aportar conocimientos en el campo con mayor utilización.

Trabajo de investigación	Lenguaje de transformación	Uso de agentes para interactuar con el SUT	Herramientas de soporte a MBT	Implementación de la propuesta	Lenguajes de especificación de modelos	Modelos o diagramas	Tecnologías de SD aplicadas
1 - UML Based Statistical Testing Acceleration of Distributed Safety-Critical Software[43]	Pseudo-código	No	No utiliza	Ninguna	UML	Diag. de estados, Diag. de secuencia	CIP
2 – Model Based Testing of Internet Email Protocols[47]	Lenguaje formal	No	JavaTESK	HDT	Final State Machine	Métodos formales	IR
3 - Model-based runtime analysis of distributed reactive systems[42]	C++	No	Runtime-Reflection Framework	HAM	Linear time temporal logic	Métodos formales	CI (Sockets - API C++)
4 - Early Fault Detection with Model based Testing[28]	QUVIQ	No	QuickCheck	HDT	Erlang	Diagrama de actividades	IR
5 - Model Based Testing for Agent Systems[50]	Java	Sí	Testing framework using JACK I.S.	HAM	Agent UML	Modelo Prometheus	CI
6 - Testing the Reliability of Web Services Transactions in Cooperative Applications[53]	Java	No	Prototipo sin nombre	HAM	UML	Diagramas de estado	IR
7 - Conformance Testing of Distributed Concurrent Systems with Executable Designs[51]	Concolic	No	Creol Tools	HAM	Creol	Modelo Creol	CI
8 - Model-Based Testing of Thin-Client Web Applications and Navigation Input[46]	G∀ST	No	G∀ST	HAM	Clean	Modelo Clean	IR
9 - Model-Based Testing of Service Infrastructure Components[44]	LTSA	No	Groove	HAM	UML	Diag. De clases, diag. de estados	IR
10 - Model-Based Testing of Service-Oriented Applications via State Models[45]	Java	No	JStateModelTest	HAM	GT Rules, EFSMs	Diagramas de estado	IR
11 - Towards a TTCN-3 Test System for Runtime Testing of Adaptable and SD [48]	Java	No	TT4RT	HDT	TTCN-3	Diag. De clases, diag. de estados	IR
12 - “The AGEDIS Tools for Model Based Testing”[52]	Java	No	AGEDIS testing framework	HDT	AGEDIS Modelling Language	Diag. De clases, diag. de estados	IR
PRUDIMA - Pruebas Dirigidas por Modelos para Sistemas Distribuidos	**Acceleo**	**Sí**	**Prototipo PRUDIMA**	**HAM**	**UML - UML Profile**	**Diag. De clases, diag. de secuencia**	**IR**

Tabla 1 - *Comparación entre trabajos relacionados*

Capítulo 3

3. CARACTERIZACIÓN DE PRUDIMA

Este capítulo inicia describiendo los metamodelos a partir de los cuales se creó el perfil UML de PRUDIMA, para abstraer tanto la complejidad de los SD como la de sus pruebas. Seguido, se presentan las transformaciones M2T realizadas para los modelos de pruebas instanciados a partir del lenguaje UML y el perfil UML de PRUDIMA. En la sección final, se define un proceso en el cual, se integran los anteriores artefactos, además de la utilización de tecnologías, técnicas y herramientas dentro de los pasos, para aplicar el enfoque MBT para las pruebas de SD en un ámbito académico.

3.1. METAMODELOS Y PERFIL UML DE PRUDIMA

Para modelar las pruebas de SD se construyeron dos metamodelos, los cuales, permiten abstraer separadamente los conceptos involucrados en el modelamiento tanto de un SD como de sus pruebas, enmarcados en el paradigma de comunicación de invocación remota [20]. Sin embargo, para lograr utilizar los metamodelos construidos de manera conjunta con UML, se hizo necesario integrarlos a través de un perfil UML que se nombró "Perfil UML de PRUDIMA", el cual, permitió extender la semántica de algunos conceptos de UML y así, utilizar este lenguaje de modelamiento ampliamente usado en la definición de los modelos de pruebas. Los metamodelos construidos fueron:

- Metamodelo para SD
- Metamodelo para el diseño de las pruebas a SD (PSD)

En la construcción de los anteriores metamodelos se usó la guía del trabajo realizado por Garcia, Fuentes y Gomez [54], que define cuatro actividades para la creación de metamodelos de tipo Entidad-Relación: *determinar las características estructurales del lenguaje de modelamiento*, *escoger entre representación homogénea o heterogénea*, *escoger entre representación redundante y no redundante* y, por último, *definir el metamodelo.* En la Figura 4 se muestra el flujo de dichas actividades.

La decisión de utilizar metamodelos basados en conexión y de tipo entidad-relación se debe a que la especificación actual de UML se basa en dichos

enfoques [18][54]. Se utilizó, además, el meta-metamodelo de *ECore* como lenguaje de modelamiento [55] para la construcción de los metamodelos.

En las secciones siguientes se detallan cada una de las actividades llevadas a cabo en la construcción de los metamodelos.

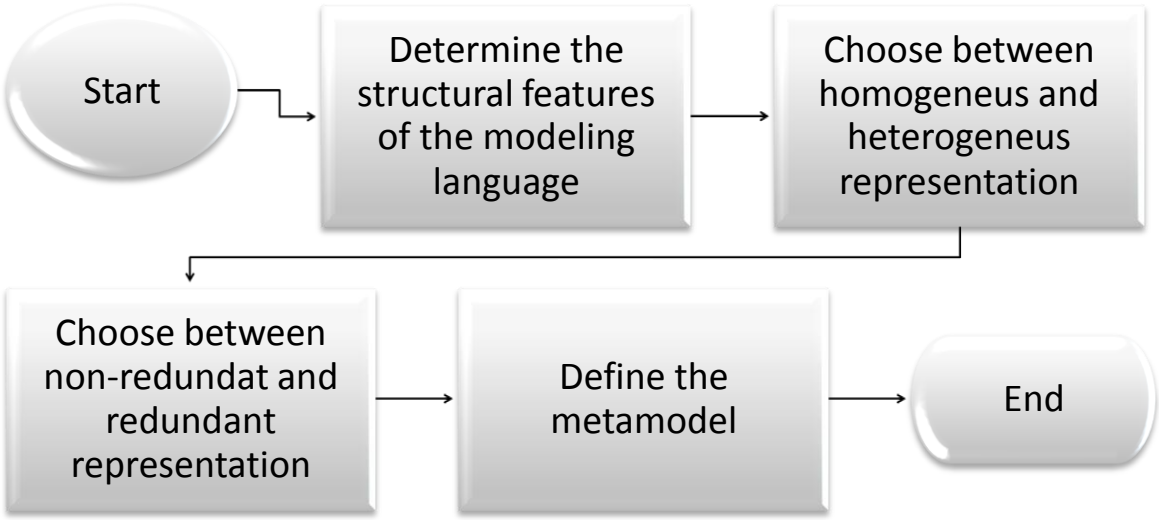

Figura 4 - *Actividades de creación de un metamodelo [51]*

3.1.1. Metamodelo para SD (MSD)

El metamodelo para SD construido, se enfocó en los SD del tipo cliente/servidor (paradigma de comunicación de invocación remota), ya que abarcan las principales tecnologías utilizadas en las prácticas académicas descritas dentro del enfoque de este trabajo. Este contiene las entidades y relaciones necesarias que permiten modelar la estructura estática de los SD basándose en los conceptos del "lenguaje de definición de interfaces" (siglas en inglés IDL) [56], [57] y en el conocimiento de expertos, para abstraer los diferentes conceptos del dominio de los SD.

A continuación, se describen las actividades llevadas a cabo para la definición de este metamodelo:

1. **Determinar las características estructurales del lenguaje de modelado**: al ser enfocado para integrarse con la especificación de UML se heredan las características estructurales. Las entidades, las relaciones y atributos del dominio de los SD tipo cliente/servidor son definidos por elementos del meta-metamodelo *Ecore*, como son: *EClasses* para las entidades,

EReferences y *EAttributes* para las relaciones finales y atributos respectivamente, estos conceptos se amplían con mayor detalle en [10], para ampliar los detalles de estos conceptos ver Anexo D.

2. **Escoger entre representación homogénea y heterogénea**: debido a que la infraestructura de UML utiliza una representación heterogénea se sigue esta misma línea de representación [18].

3. **Escoger entre representación redundante y no redundante**: para este caso es adoptada la representación redundante, ya que permite tener una mejor navegabilidad desde las distintas entidades del metamodelo y su eficiente procesamiento.

4. **Definición del metamodelo**: en la Figura 5, se presentan las entidades de dominio, relaciones y atributos utilizados por el metamodelo. Los tipos de relación utilizados en el metamodelo son obtenidas a partir de las relaciones utilizadas en el lenguaje UML, como las asociaciones por agregación y composición.

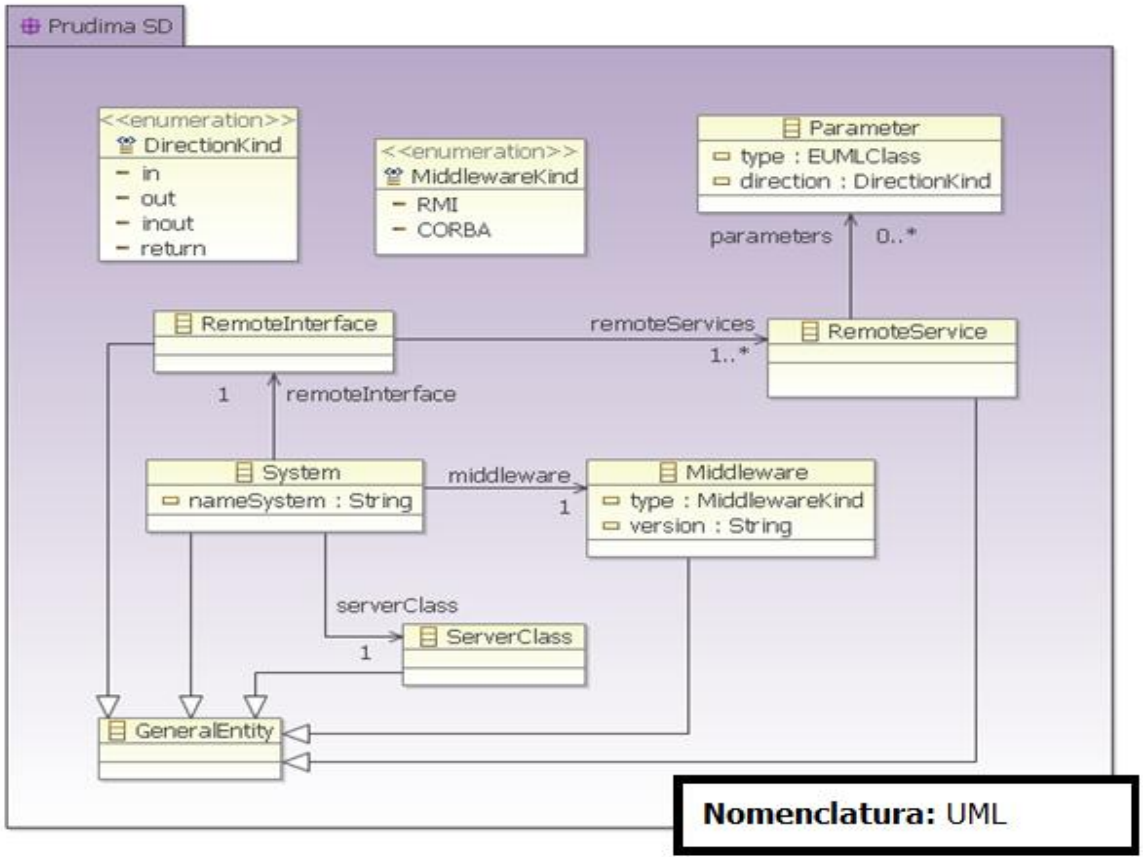

Figura 5 - *Vista de entidades metamodelo DS*

Especificación del metamodelo de SD

A continuación, entre las tablas 2 y 7 se especifica la semántica de las entidades que componen el metamodelo de SD:

Entidad	System	Identificador	MSD-E1
Descripción	Esta entidad contiene la configuración del SD, por medio de la definición de los parámetros y relaciones que establecen la configuración.		
Atributos			
Nombre	Descripción		Tipo
nameSystem	Contiene el nombre del SD		String
Relaciones			
Nombre	Descripción	Multiplicidad	Tipo
remoteInterface	Define una interfaz remota del sistema.	Uno a uno	Composición
middleware	Este atributo define el middleware sobre el cual, el SD se soporta.	Uno a uno	Composición
serverClass	Representa el elemento que desplegará el sistema SD	Uno a uno	Composición
Semántica			
Esta entidad caracteriza el SD y define su contexto tecnológico sobre el cual se soporta, este contexto puede ser RMI ó CORBA. Además de asociar por composición una interface remota, la cual es el punto de entrada para que los clientes consuman los servicios del sistema.			

Tabla 2 - *Descripción de la entidad System*

Entidad	Remote Interface	Identificador	MSD-E2
Descripción	Esta entidad representa la interfaz remota del SD		
Atributos			

Ninguno			
Relaciones			
Nombre	Descripción	Multiplicidad	Tipo
remoteServices	Lista de servicios remotos asociados a la interfaz remota	Uno a muchos	Composición
system	Representa la configuración del SD	Uno a uno	Composición
Semántica			
Es una entidad que define las funcionalidades del componente servidor, por medio de un conjunto de servicios remotos, los cuales pueden ser invocados por un cliente.			

Tabla 3 - *Descripción de la entidad Remote Interface*

Entidad	Remote Service	Identificador	MSD-E3
Descripción	Esta entidad representa uno de los servicios presentes en la interfaz remota del SD		
Atributos			
Ninguno			
Relaciones			
Nombre	Descripción	Multiplicidad	Tipo
parameters	Lista de elementos del tipo Parameter, que son utilizados en las invocaciones a los métodos, como argumentos de entrada o salida.	Uno a muchos	Agregación
serverClass	Representa el elemento que desplegará el SD.	Uno a uno	Composición
Semántica			
Esta entidad declara un servicio remoto, con sus parámetros de entrada y salida. El cual, realizará una acción en el componente servidor y si es requerido, retornará el resultado al cliente que lo haya invocado.			

Tabla 4 - *Descripción de la entidad Remote Service*

Entidad	Parameter	Identificador	MSD-E4
Descripción	Esta entidad representa un parámetro de un Remote Service		
Atributos			
Nombre	Descripción		Tipo
type	Define el tipo asociado al cual pertenece el parámetro		EUMLClass
direction	Establece la dirección del parámetro, este puede ser de entrada o salida (“in”, “out”, “in/out”, “return”)		DirectionKind
Relaciones			
remoteServices	Lista de servicios remotos asociados a la interfaz remota		Uno a muchos
Semántica			
Esta entidad declara un parámetro de un servicio remoto permitiendo conocer el tipo de elemento que representa y la dirección en la que viaja.			

Tabla 5 - *Descripción de la semántica de la entidad Parameter*

Entidad	Middleware	Identificador	MSD-E5
Descripción	Esta entidad permite establecer el contexto tecnológico del SD.		
Atributos			
Nombre	Descripción		Tipo
middlewareType	Establece el tipo de tecnología sobre el cual se soportará el SD.		MiddlewareKind
version	Especifica la versión del middleware.		String
Relaciones			
system	Representa la configuración del SD		Uno a uno
Semántica			

Esta entidad permite especificar a través de los atributos middlewareType a qué tipo de plataforma tecnología estará enfocado el SD y version la versión de la plataforma tecnológica.

Tabla 6 - *Descripción de la entidad Middleware*

Entidad	ServerClass	Identificador	MSD-E6
Descripción	Esta entidad representa el elemento que desplegará los servicios remotos del SD.		
Atributos			
Ninguno			
Relaciones			
system	Representa la configuración del SD	Uno a uno	
Semántica			
Esta entidad representa la clase servidor remoto, la cual tiene como funcionalidad desplegar los servicios del SD.			

Tabla 7 - *Descripción de la entidad ServerClass*

3.1.2. Metamodelo para PSD (MPSD)

Se crea un metamodelo para las pruebas a los SD basándose en los conceptos de UML-TP [58]. La decisión de construir un metamodelo propio para las pruebas a los SD se debe a que UML-TP está concebido con conceptos genéricos para varios dominios de aplicación [59], sin embargo, para el enfoque práctico que se le quiso dar a este trabajo, resultaba muy complejo de implementar.

No obstante, se analizaron los conceptos como: *arquitectura de pruebas*, *datos de pruebas*, *comportamiento de pruebas*, *tiempo de pruebas*. A partir de estos se definieron los elementos necesarios para poder tener un metamodelo de pruebas que permitiera modelar tanto pruebas unitarias como funcionales de una manera práctica y que dichos modelos pudiesen ser integrados con las instancias del metamodelo de SD.

Para la construcción de este metamodelo se utilizó la misma guía que para el metamodelo de SD y se tomaron las mismas decisiones respecto a las tres primeras actividades: “Determinar las características estructurales del lenguaje de modelado”, “Escoger entre representación homogénea y heterogénea”, “Escoger

entre representación redundante y no redundante", debido a que se siguen los lineamientos de UML.

Lo que diferencia a este metamodelo del metamodelo de SD, se especifica en cuarta actividad, que se trata de la definición del mismo:

- **Definición del meta-modelo**: en la Figura 6, se definen las entidades de dominio que corresponde a: *Test, Parameter, MockClient, TestData, Step* y *Assertion*. Los tipos de relaciones utilizadas por el metamodelo son tomados a partir de las relaciones definidas en UML. Luego, en la Figura 8 se presenta otra vista del metamodelo mostrando las relaciones entre las distintas entidades del dominio.

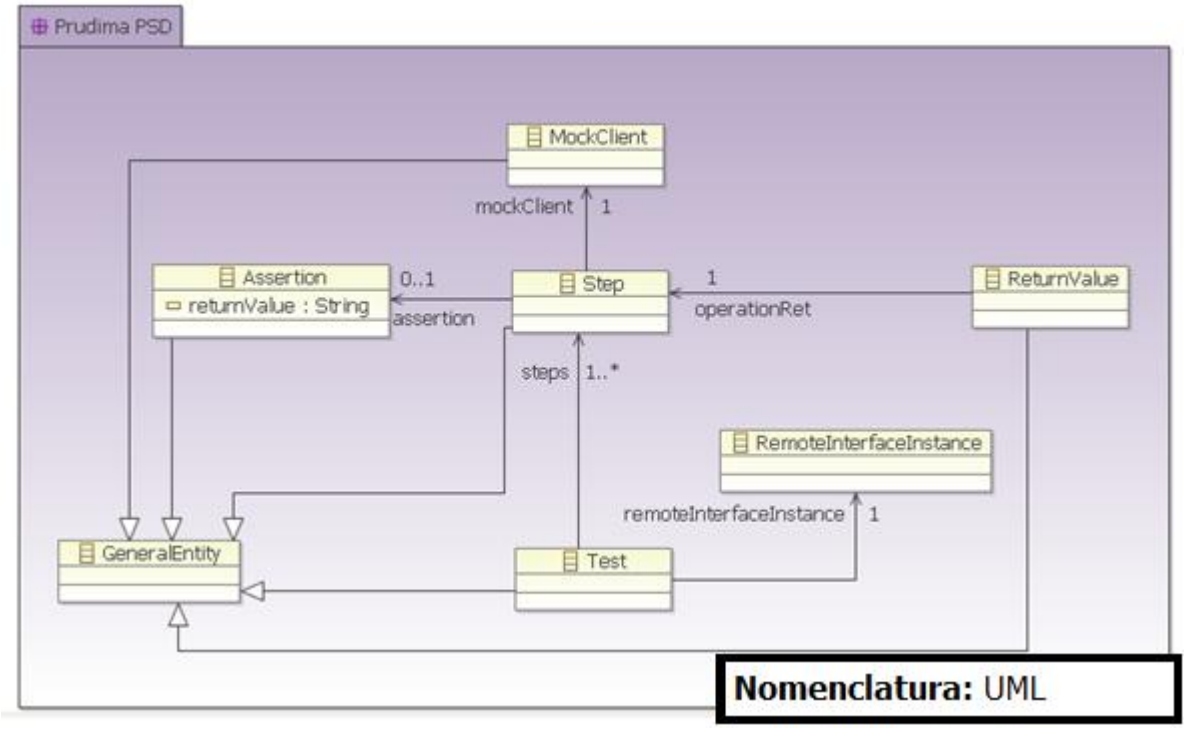

Figura 6 - *Vista de entidades metamodelo PDS*

3.1.3. Especificación del metamodelo de pruebas a SD (PSD)

A continuación se presentan entre las Tabla 8 y la Tabla 13 se especifica la semántica de las entidades que componen el metamodelo de SD, por medio de las descripciones de sus atributos y relaciones:

Entidad	Test	Identificador	MPSD-E1
Descripción	Representa una prueba unitaria o funcional dentro de las pruebas al sistema.		
Atributos			
Ninguno			
Relaciones			
Nombre	Descripción	Multiplicidad	Tipo
steps	Lista de pasos ordenadas que componen la prueba	Uno a muchos	Composición
remoteInterfaceInstance	Representa una instancia de una interfaz remota en ejecución.	Uno a uno	Composición
Semántica			
Esta entidad permite representar pruebas unitarias o funcionales dentro de las pruebas al sistema, por medio de una lista de pasos ordenados.			

Tabla 8 *- Descripción de la semántica de la entidad Test*

Entidad	Step	Identificador	MPSD-E2
Descripción	Esta entidad representa un paso, el cual invoca una operación del sistema bajo prueba.		
Atributos			
Ninguno			
Relaciones			
Nombre	Descripción	Multiplicidad	Tipo
mockClient	Es un atributo que relaciona el cliente desde el cual se invoca el paso, su multiplicidad es de uno a uno, y el tipo de asociación es de composición.	Uno a uno	Composición
remoteService	Representa un servicio remoto contenido en el	Uno a uno	Composición

	metamodelo de SD.		
test	Representa una prueba unitaria o funcional	Muchos a uno	Composición
assertion	Representa una condición para el retorno de una operación.	Uno a uno	Composición
Semántica			
Esta entidad especifica un paso, el cual será el encargado de hacer el llamado a una operación del SUT utilizando para ello el atributo asociado con la entidad RemoteService.			

Tabla 9 - *Descripción de la semántica de la entidad Step*

Entidad	MockClient	Identificador	MPSD-E3
Descripción	Esta entidad representa un actor cliente del sistema bajo prueba.		
Atributos			
Ninguno			
Relaciones			
step	Paso de la prueba	Uno a muchos	
Semántica			
Esta entidad define a un actor cliente el cual es el encargado de ejecutar pasos dentro de una prueba del sistema.			

Tabla 10 - *Descripción de la semántica de la entidad MockClient*

Entidad	Assertion	Identificador	MPSD-E4
Descripción	Esta entidad representa una condición para el retorno de una operación del SUT.		
Atributos			
Nombre	Descripción		Tipo
returnValue	Permite establecer el valor esperado de la función.		Literal
Relaciones			

Ninguna
Semántica
Esta entidad define una condición que se espera se cumpla o no, y cuya comparación se define por el valor retornado de los servicios remotos del SD.

Tabla 11 - *Descripción de la semántica de la entidad Assertion*

Entidad	Return Value	Identificador	MPSD- E5
Descripción	Esta entidad representa el valor de retorno de la invocación a un metodo.		
Atributos			
Ninguno			
Relaciones			
Nombre	**Descripción**	**Multiplicidad**	**Tipo**
step	Este atributo de tipo Step almacena el identificador de un paso asociado a esta entidad	Uno a uno	Composición
Semántica			
Esta entidad representa un valor de retorno que será devuelto por un paso de una prueba en ejecución.			

Tabla 12 - *Descripción de la semántica de la entidad Return Value*

Entidad	RemoteInterfaceInstance	Identificador	MSD-E8
Descripción	Esta entidad representa una instancia de una interfaz remota en ejecución		
Atributos			
Ninguno			
Relaciones			
test	Representa una prueba unitaria o funcional	Muchos a uno	

Semántica
Esta entidad permite representar una instancia en ejecución de una interfaz remota que puede recibir solicitudes a través de sus servicios remotos.

Tabla 13 - *Descripción de la entidad RemoteInterfaceInstance*

3.1.4. Integración de meta-modelos en un perfil UML

Un perfil UML permite extender la semántica de UML, la definición es "un perfil UML es definido como un paquete estereotipado *profile*, que puede extender un metamodelo u otro Perfil. Los perfiles UML son definidos en términos de tres mecanismos básicos: estereotipos, restricciones y valores etiquetados" [60].

Para lograr utilizar en conjunto los metamodelos previamente construidos, se hizo necesario definir un perfil UML que permitiera integrar las entidades definidas en estos y a su vez poder utilizar las entidades de UML. Debido a lo anterior, se sigue los pasos que se encuentran en la guía [60]:

- **Paso 1:** se debe definir el conjunto de elementos que comprenden el sistema y las relaciones entre estos. Aquí se utiliza los metamodelos previamente construidos, de manera que las entidades definidas se conviertan en **Estereotipos** que permiten generalizar y extender la semántica de algunas **metaclases** de UML.

- **Paso 2:** se define el Perfil UML, donde se establecen los **Stereotipos** para cada elemento relevante del metamodelo.

- **Paso 3:** definir a qué **Metaclase** de UML será aplicado cada estereotipo.

- **Paso 4:** establecer los atributos que aparecen en el metamodelo como **Valores Etiquetados**, incluyendo sus tipos y valores iniciales.

- **Paso 5:** definir las **Restricciones Del Dominio**, como por ejemplo la multiplicidad de las relaciones.

La Tabla 14, contiene los requisitos para los pasos del segundo al quinto, haciendo referencia a los identificadores de los elementos ya descritos en las secciones 3.1.1 y 3.1.2 que definen los metamodelos SD y PSD respectivamente. Donde se definieron los atributos y relaciones cumpliendo el paso dos. Para el tercer paso, en la columna descripción encontramos además las metaclases de UML que son extendidas por los estereotipos. Para las restricciones de relaciones del cuarto paso y finalmente las restricciones de multiplicidad del paso quinto, se hace referencia nuevamente a las secciones de los metamodelos.

Se utilizó la herramienta *Papyrus* dentro del entorno *Eclipse Model Framework* (EMF) [55] como herramienta de modelado, y el resultado del modelamiento se puede ver en la *Figura 7*, donde se observa el modelo completo del perfil UML.

Estereotipo	Descripción	Metaclase	Referencia al elemento
RemoteInterface	Permite indicarle a un elemento del tipo Interface de UML que se comporte como un elemento del tipo RemoteInterface.	Interface	MSD-E2
RemoteService	Relaciona un elemento del tipo Operation de UML con una entidad del metamodelo SD llamada RemoteService.	Operation	MSD-E3
System	Extiende un elemento Package de UML para que se comporte como un System del metamodelo SD.	Package	MSD-E1
Middleware	Estereotipo que se relaciona con la entidad Middleware del metamodelo SD.	*Ninguna*	MSD-E5
ServerClass	Extiende un elemento Class de UML para que se comporte como una entidad ServerClass del metamodelo SD.	Class	MSD-E6
Parameter	Extiende un elemento Parameter de UML para que se comporte como una entidad Parameter del metamodelo SD.	Parameter	MSD-E4
ReturnValue	Estereotipo que se relaciona el conjunto de elementos de UML especificados en el ítem Metaclases, al metamodelo PSD.	LiteralBoolean LiteralInteger LiteralString LiteralReal LiteralNull	MPSD-E5
Step	Extiende un elemento Message de UML para que se comporte como una entidad Step del metamodelo PSD.	Message	MPSD-E2
MockClient	Extiende un elemento Lifeline de UML para que se comporte como una entidad MockClient del metamodelo PSD.	Lifeline	MPSD-E3
Test	Extiende un elemento Interaction de UML para que se comporte como una entidad Test del metamodelo PSD.	Interaction	MPSD-E1
Assertion	Extiende un elemento Message de UML para que se comporte como una entidad Assertion del metamodelo PSD.	Message	MPSD-E4
RemoteInterfaceInstance	Extiende un elemento Lifeline de UML para que se comporte como una entidad RemoteInterfaceInstance del metamodelo SD.	Lifeline	MPSD-E4

Tabla 14 - *Definición de Perfil UML de PRUDIMA*

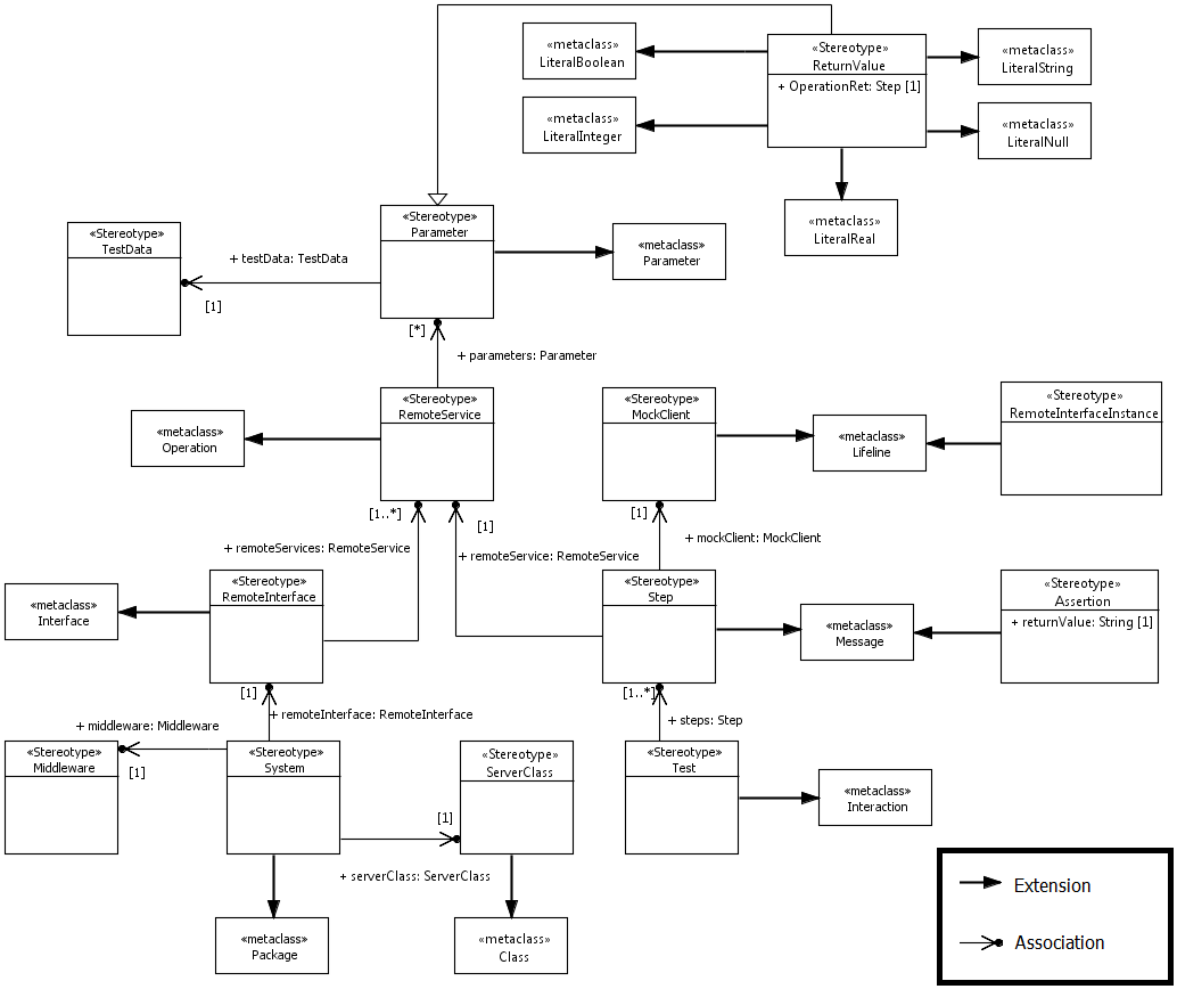

Figura 7 - *Perfil UML PRUDIMA para pruebas a SD*

3.2. TRANSFORMACIONES DE MODELOS EN PRUDIMA

Dentro de MBT existe una actividad llamada *concretización,* en la cual, se transforman los modelos abstractos de las pruebas del sistema a un formato ejecutable. A este tipo de transformaciones se las llama *transformaciones de modelo a texto* (M2T).

Por lo anterior, se desarrollaron una serie de transformaciones de manera iterativa e incrementalmente que permitieron generar un sistema software que cumple con las funcionalidades necesarias para el uso del enfoque MBT de este trabajo. El sistema está compuesto por dos componentes principales, de estructura dinámica y estática. Los detalles del sistema se pueden ver en el Capítulo 4, el cual fue desarrollado en paralelo con las transformaciones.

Como el sistema debía ser ejecutable, se escogió el lenguaje Java como el formato objetivo de la implementación del sistema, y el formato XML (del inglés *eXtensible Markup Language*) para interconectar el sistema dinámico con el estático. Para Java se utilizó la implementación de JDK debido a que está catalogado dentro de la licencia GPL de GNU. En las siguientes secciones se realizará la caracterización de las transformaciones de PRUDIMA como también el proceso utilizado para llegar a estas.

3.2.1. Método para crear las transformaciones de modelo a texto de PRUDIMA

Las transformaciones de modelo a texto de PRUDIMA, se crearon utilizando el método de ingeniería o paradigma evolutivo el cual consta de las siguientes fases:

1) *Observar soluciones existentes*.
2) *Proponer mejoras.*
3) *Construir*, *medir y analizar la solución propuesta.*

Estas fases se realizan de manera iterativa hasta lograr el objetivo de generar una transformación de modelo a texto, de manera eficiente pero sobre todo que se adapte a las necesidades expuestas en este proyecto de investigación.

Observar soluciones

Las transformaciones de modelo a texto constituyen una tecnología fundamental en el desarrollo de software dirigidos por modelos [61], en la cual, por medio de algún mecanismo permite generar automáticamente una salida de texto a partir de modelos, los cuales poseen representación implícita o explícita de un metamodelo dado. En esta fase se realizó una evaluación, basándose en el trabajo de V. Maximiliano [62], de los lenguajes y herramientas principales de transformación de modelo a texto que se encuentran en el mercado y que pueden ser utilizadas dentro del contexto del presente trabajo.

Esta evaluación se pueden encontrar en el Anexo E, dando como resultado el cuadro comparativo presentado en la Tabla 15, donde se encuentra las siguientes lenguages: *JET*[7], MOFScript, *Xpand*[8] y *Acceleo*[9].

[7] **JET**: Java Emiter Template
Project: http://www.alice-sl.net/siegfried.nolte/forum/mda/m2tsamples/m2t_JET_UML2Java.zip
[8] **Xpand**: es un lenguaje especializado en generación de código basado sobre modelos de EMF
Project: http://www.alice-dsl.net/siegfried.nolte/forum/mda/m2tsamples/m2t_OAW_UML2Java.zip
[9] **Acceleo**: http://www.acceleo.org/pages/home/en
Project: http://www.obeonetwork.com/group/uml-to-java-generator

Características	Acceleo	JET	MOFScript	Xpand
Tipo de implementación	Textual	Textual	Textual	Textual
Tipo de transformaciones soportadas	M2T	M2T	M2T	M2T
Estilo	Declarativo y operacional	Declarativo y operacional	Declarativo y operacional	Declarativo y operacional
Modular	SI	No	No	SI
Se puede realizar trazabilidad	SI	SI	SI	SI
Reutilización de las reglas implementadas	SI	SI	SI	Si
Tipo de licencia	Licencia EPL	Licencia EPL	Licencia EPL	Licencia EPL
Documentación disponible	Buena	Regular	Regular	Regular
Se basa en los estándares de la OMG	SI	NO	SI	NO
Código Java	Por medio de servicios	Directamente en las plantillas	No utiliza	No utiliza

Tabla 15 - *Comparación entre herramientas de M2T*

Tras haber analizado los lenguajes por medio de los criterios presentados en el la Tabla 15, se optó por la herramienta de transformación Acceleo, debido a que es el único que cumple con el estándar "*MOF Model-To-Text Transformation Language, MOFM2T*" definido por la OMG en 2008 [63], por su enfoque basado en plantillas que permite una separación y reutilización de las funcionalidades y por la documentación disponible.

Proponer mejoras a las soluciones

En esta fase se inicia el diseño de las transformaciones de modelo a texto, basándose en los conceptos *UML2Java* y realizando una adaptación que permita soportar los metamodelos de SD y P-SD, integrados con el perfil UML de PRUDIMA extendido a partir de UML.

Como ya se mencionó, el objetivo de las transformaciones es llegar a la generación del código fuente del sistema ejecutable con soporte para MBT. Por este motivo las transformaciones se irán generando a medida que las funcionalidades de los componentes dinámicos del sistema, sean generados. Por

ende, se realizó un proceso de ingeniería inversa, para lograr definir las transformaciones necesarias.

Para realizar las trasformaciones se propone utilizar la potencialidad que nos brinda la utilización de plantillas por medio de la arquitectura representada en la *Figura 8*, compuesta por paquetes que deberán contener plantillas (archivos con extensión .mtl) donde se encuentra el código fuente de transformación de modelo a texto en lenguaje de *Acceleo* y *servicios* que están implementados en lenguaje *Java* los cuales prestan funcionalidades a las plantillas. Estos servicios son utilizados en los casos en que se requiere una lógica más elaborada y que sea difícil de implementar directamente en las plantillas.

Los paquetes definidos en la arquitectura agrupan conceptualmente las diferentes transformaciones que se realizan a los elementos de UML y estereotipos aplicados a los modelos, representados por medio de diagramas de clases y de secuencia.

En este trabajo, únicamente se soportan los elementos de UML expuestos en la descripción del perfil UML de PRUDIMA, ver Sección 3.1.4.

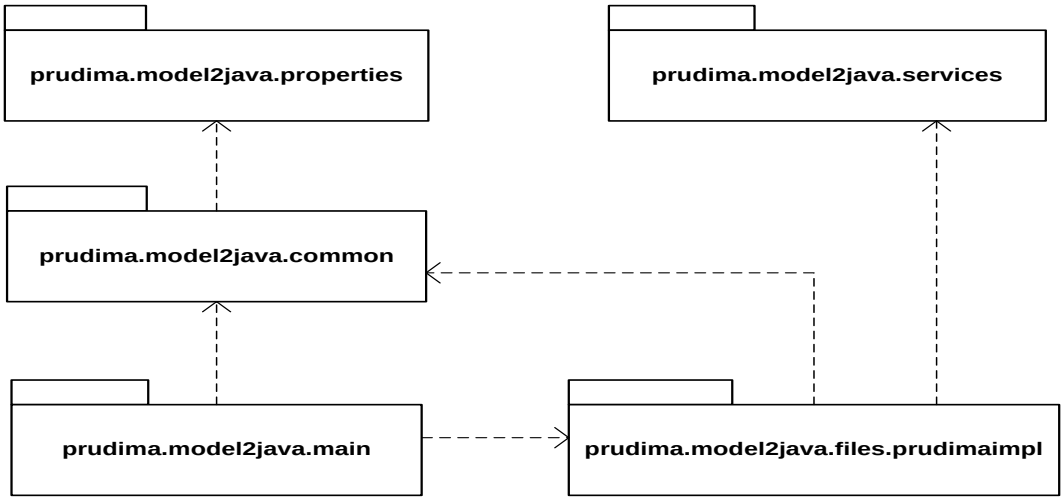

Figura 8 - *Arquitectura de transformaciones de PRUDIMA*

Paquetes

En esta sección, se describen cada uno de los paquetes presentes en la arquitectura de transformaciones:

- ***prudima.model2java.main:*** contiene la plantilla de la cual parte el proceso de transformación de modelo a texto, a partir de esta plantilla se realizan invocaciones a otras plantillas definidas en los paquetes asociados.
- ***prudima.model2java.files.prudima:*** contiene las plantillas específicas para los elementos que tengan aplicados el perfil UML de PRUDIMA.

- ***prudima.model2java.common:*** contiene plantillas que son transversales y reusables por el resto de platillas.
- ***prudima.model2java.properties:*** contiene plantillas con valores constantes que son utilizados para configuraciones estáticas de las transformaciones.
- ***prudima.model2java.services:*** contiene archivos con código fuente en lenguaje java que permiten tener servicios que pueden ser invocados por el resto de plantillas. Los servicios se refieren a métodos en código java que pueden contener lógica compleja.

Construir, medir y analizar la solución propuesta

En esta fase, se implementaron las transformaciones con el lenguaje de trasformación *Acceleo*, teniendo como base el código fuente de los componentes dinámicos del prototipo PRUDIMA, se implementó la arquitectura expuesta en la Figura 8.

Como plataforma de desarrollo, se seleccionó el entorno de desarrollo integrado (IDE) de *Eclipse Modeling Framework*, ya que la herramienta *Acceleo* está bajo su marco de trabajo. En la Figura 9, se puede observar la sintaxis de código de una de transformación en Acceleo, cuyo fin es extraer la información desde el modelo de entrada, para transformarlo a un archivo con código fuente en lenguaje Java. El modelo está formado en bajo nivel, como un archivo en formato XMI, el cual es utilizado por el EMF.

Como se mostró en la Figura 8, se construyeron cinco paquetes de transformaciones. A continuación se describen los archivos que contienen la lógica para realizar las transformaciones de modelo a texto.

Plantillas de transformación

A continuación se presenta una plantilla de descripción de alto nivel de los requisitos que debe cumplir cada plantilla de transformación, para más información de los requerimientos de las plantillas, vea el Anexo F.

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.

Posteriormente implementación de cada uno de los módulos y plantillas generadas, se ha creó una clase encargada de prestar los servicios u operaciones complejas a las transformaciones.

Una vez realizado el proceso de desarrollo de las transformaciones expuestas en los puntos anteriores, debemos proceder a realizar la transformación de modelo a texto, para esto debemos tener el esquema de trabajo de *Acceleo* expuesto en la Figura 10, donde, a partir de un modelo debemos seleccionar el modulo a generar y como resultado *Acceleo* entregará unos archivos de salida como se presenta en la Figura 11. En esta se encuentra un ejemplo del código fuente en lenguaje Java resultante tras la aplicación de las transformaciones a un modelo.

```
[comment encoding = UTF-8 /]
[**
 * Prudima Project
 **/]
[module generate('http:///prudimaSd.ecore', 'http://www.eclipse.org/uml2/4.0.0/UML')]
[import co::edu::unicauca::prudima::model2java::files::prudima::UtilitiesRMIJavaFile    /]
[import co::edu::unicauca::prudima::model2java::files::prudima::TestManagerJavaFile     /]
[import co::edu::unicauca::prudima::model2java::files::prudima::TestControllerJavaFile  /]
[import co::edu::unicauca::prudima::model2java::files::prudima::PrudimaMainJavaFile /]

[import co::edu::unicauca::prudima::model2java::files::prudima::agentCaseTestJavaFile   /]
[import co::edu::unicauca::prudima::model2java::files::prudima::agentJadeJavaFile       /]
[import co::edu::unicauca::prudima::model2java::files::prudima::agentManagerJavaFile    /]
[import co::edu::unicauca::prudima::model2java::files::prudima::ExcBatFile /]

[template public generateElement(aModel : Model)]
[comment @main/]
[for (aInterface : Interface | aModel.eAllContents(Interface))]
    [aInterface.genUtilitiesRMIJavaFile() /]
[/for]
[aModel.genManagerJavaFile()        /]
[aModel.TestControllerJavaFile()    /]
[aModel.genTestManagerJavaFile()    /]
[aModel.ExcBatFile()                /]
[aModel.genPrudimaMainJavaFile() /]
[for (aInteraction : Interaction | aModel.eAllContents(Interaction) )]
    [aInteraction.genAgentCaseTestJavaFile() /]
    [for (l : Lifeline | aInteraction.eAllContents(Lifeline))]
        [for (s : Stereotype | l.getAppliedStereotypes())]
            [if (s.name.equalsIgnoreCase('MockClient'))]
                [l.genAgentJadeJavaFile() /]
            [/if]
        [/for]
    [/for]
[/for]
[/template]
```

Figura 9 - *Ejemplo código fuente de transformaciones en Acceleo*

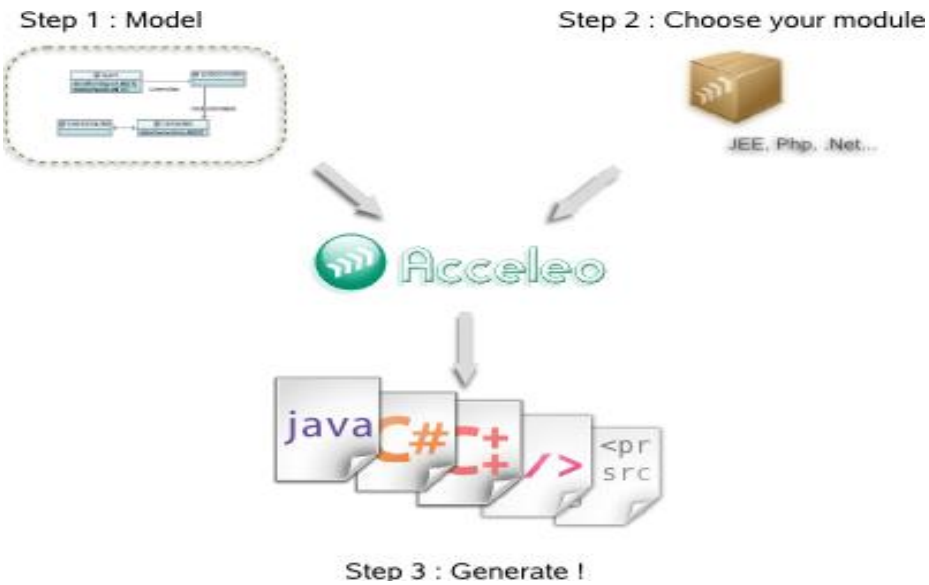

Figura 10 - *Esquema de generación Acceleo*

CallsManager.java

```
package agents.behaviours;

import java.net.MalformedURLException;

public class CallsManager extends CyclicBehaviour {

    private static final long serialVersionUID = 1L;
    private ICareersSystem _iCareersSystem;
    private Agent _owner;

    public CallsManager(Agent a, ICareersSystem careersSystem) {

    public void action() {

    private void Initialize(ACLMessage reply) throws Exception {
    ViewUsersOnline _viewUsersOnline;
    Authenticate _authenticate;
    DelCareer _delCareer;
    ViewListCareers _viewListCareers;
    AddCareer _addCareer;
    DelUser _delUser;
    AddUser _addUser;
    AssignRole _assignRole;

    // Support classes
    private class ViewUsersOnline {
        private Integer roleType;
        public Integer getRoleType() {
    }
```

Figura 11 - *Ejemplo de código java tras ejecución de transformaciones PRUDIMA*

3.3. PROCESO PARA PRUEBAS DE SD

De acuerdo con el análisis de la literatura, se encontraron varios trabajos donde se presentas diferentes procesos [23], [28], [30], [37], [41], [64], [65], [66], [67], [68] para aplicar MBT y la descripción de sus pasos en contextos determinados. A partir de ellos, se realizó un análisis de los pasos que mejor se ajustaban para lograr mejorar la calidad a las pruebas de SD. Con lo cual, se definió el proceso práctico de la aplicación de MBT para SD (P-MBT-SD), ver Figura 12, el cual

define los pasos que se deben llevar con el orden especifico que el flujo indica. Este proceso consta tanto de pasos manuales, las cuales son efectuadas por el *tester*, como también por tareas automáticas realizadas por el sistema software que apoya el proceso, que es un prototipo de una herramienta CASE para MBT construida por los autores de este trabajo, que se denominó *PRUDIMA*.

Se hallaron similitudes en los procesos estudiados y se tomó como base el proceso más genérico, que consta de los siguientes pasos: *Modelar, Generar, Concretizar, Ejecutar y Analizar,* los cuales están presentes en todos los trabajos. Algunos de los pasos anteriores se llevan a cabo de varias maneras utilizando diferentes enfoques, dependiendo del contexto donde se esté aplicando MBT.

P-MBT-SD es un proceso que puede ser aplicado de una manera tanto iterativa e incremental como aplicarlo en la última fase de pruebas, como se maneja en algunas metodologías de desarrollo[10]. Sin embargo, es recomendable que sea aplicado desde un principio, ya que permite aprovechar los beneficios del enfoque MBT, como encontrar errores en las etapas tempranas del desarrollo del SUT, corrección de requerimientos pobremente definidos, entre otros [26][28].

3.3.1. Descripción de los pasos de P- MBT- SD

A continuación se realiza una descripción detallada de cada uno de los pasos que compone el P-MBT-SD.

1 *Modelar el SUT:* el primer paso corresponde a diseñar el SD que será probado (SUT). Esto lo realizará el *diseñador*[11] del sistema, quien utilizará los requerimientos del sistema previamente definidos para modelar tanto la parte estática como la parte dinámica del SD. En este caso, PRUDIMA soporta el modelamiento con el estándar UML [18].

***Diagrama de clases*:** es un artefacto el cual permite tener una vista estática del sistema. Para este proceso, dentro del diagrama se define los contratos necesarios a través de interfaces y clases relacionadas que permiten conocer los servicios remotos del SD que van a ser el objeto de pruebas.

***Diagramas de interacción*:** este artefacto permite tener una vista dinámica del sistema, es decir, se pueden observar comportamientos por medio del paso de mensajes entre objetos del sistema. Dentro de la especificación de UML [18] existen dos tipos de estos diagramas: *diagramas de colaboración* y *secuencia*. Se decide utilizar los diagramas de secuencia porque enfatizan en el ordenamiento temporal del paso de mensajes entre los objetos.

[10] **El modelo de desarrollo en cascada**: es un ejemplo de modelo, en el cual se tiene una fase de pruebas que se realiza al final del desarrollo del sistema.

[11] **Diseñador**: persona o grupo de personas encargadas de modelar el sistema

2 *Modelar Casos de Prueba*: en este paso se requiere de los entregables realizados en el paso previo: diagrama de clases y diagramas de interacción.

A partir de estos insumos, se aplica el *perfil UML de PRUDIMA* (ver Sección 3.1.4) a los diagramas de clases e interacción, con lo cual, se realiza el proceso de modelamiento de casos de pruebas, convirtiendo en escenarios de pruebas abstractos los diagramas de secuencia. Este paso sigue un enfoque de especificación explicita de casos de pruebas [30], debido a la ventaja de tener completo control sobre los casos de pruebas que se están generando.

Cabe resaltar, que al utilizar los diagramas de UML como base para el modelamiento, se fomenta la reutilización de los entregables producidos en fases previas de algunas metodologías de desarrollo de software como RUP, UP Ágil, entre otras. Por otro lado, al modelar los casos de pruebas en las fases de diseño, se pueden encontrar requerimientos pobremente definidos y corregirlos de manera temprana.

Contrastando P-MBT-SD con el de otros trabajos, el primero utiliza el lenguaje UML que es el lenguaje de facto para modelar en la industria del software, mientras que los otros utilizan lenguajes de modelamiento no estándares como *Markov Chains*, *Erlang*, diagramas de objetos, máquinas de estados, UML BPEL *profile*, OCL. No obstante, también se encontraron buenas prácticas como definir y modelar los requerimientos en lenguaje SysML, para que a partir de este modelo, derivar los modelos del sistema con UML, permitiendo tener una trazabilidad entre modelos del sistema y el modelo de requerimientos.

3 *Validar modelos:* es el paso a seguir una vez estén listos los modelos de pruebas, se analizan los modelos de pruebas buscando posibles errores de aplicación del perfil UML de PRUDIMA, los cuales no permitan transformarlos a un formato ejecutable. Este paso es automáticamente soportado por PRUDIMA. Lo cual, ayuda a reducir costos, ya que permite encontrar errores de modelamiento en etapas tempranas. Para esto PRUDIMA utiliza unas reglas definidas así:

- ✓ Se deben encontrar aplicados los estereotipos del perfil UML de PRUDIMA a los elementos necesarios para concretizar los casos de pruebas que se encuentren en los diagramas de clases e interacción.
- ✓ Las aserciones dentro del modelo deben contener valores válidos para el sistema.

4 *¿Es correcto el modelo?:* en este paso el modelador verifica el resultado de la validación anterior, si todo es correcto sigue al paso 5, sino se dirige al paso 4.1.

4.1 *Corregir modelos de pruebas:* el modelador corrige los defectos encontrados en el modelo. Seguido vuelve a realizar al paso 3 que valida el modelo.

5 *Concretizar sistema de pruebas*: este paso, hace referencia a la concretización de las pruebas abstractas siguiendo un enfoque de *transformación* [30], es decir, se transforman a código fuente (M2T) el conjunto de pruebas abstractas que se construyeron a partir de los modelos abstractos del SUT en el primer paso. Existen tres enfoques para el proceso de concretización: *adaptación*, *transformación* y *mixto* [30][40][66][69], se decidió utilizar el segundo, porque, no requiere construir *adaptadores*[12] para la comunicación con el SUT, como es el caso del enfoque por adaptación, lo que permite disminuir la carga cognitiva para el *tester*.

Junto al código ejecutable de las pruebas, PRUDIMA genera también toda la arquitectura necesaria para tener un sistema de ejecución de pruebas completo. Este sistema incluye el soporte para inclusión, despliegue y estimulación del SUT, métodos del oráculo de pruebas, motor de ejecución de pruebas y reporte de resultados de pruebas.

La generación de los métodos de oráculo de pruebas, son para determinar el veredicto de los casos de pruebas [41], que para este trabajo pueden ser *aprobó* o *falló.* Estos siguen un enfoque manual ya que son alimentados por los modelos y por el repositorio de datos de casos de pruebas que también siguieron el mismo enfoque.

La generación automática de los oráculos de prueba se sale del alcance del presente trabajo, sin embargo, en la literatura existen trabajos que lo hacen por medio de diversas formas como: el uso de *máquinas de estado de UML* [40], *redes neuronales artificiales* [70]*, "N-Version Diverse Systems and M-Model Program Testing", "Decision Table", "IFN Regression Tester", "AI Planner Test Oracle", "ANN Based Test Oracle", "Input/output Analysis Based Automatic Expected Output Generator"* [41].

6 *Generar casos de pruebas*: luego de tener los escenarios de pruebas se requiere de una tarea manual por parte del *tester,* el cual crea los casos de prueba, es decir, crea una instancia de un escenario de prueba; para luego insertar la información requerida por cada uno de los casos de prueba. Dicha información es almacenada en un repositorio de datos de pruebas, para luego ser consumida en ejecución.

[12] **Adaptador**: es código fuente que se debe escribir para interactuar con la API del SUT [69].

Dado que la generación automática de datos no es el foco de este trabajo, aquí se genera manualmente, en concordancia con el criterio de selección de casos de prueba escogido en el segundo paso. Sin embargo, se realizó la exploración de trabajos relacionados con la generación automática y se encontraron técnicas como: particiones de equivalencia, análisis de valores, generación de datos aleatorios, modelos de datos, entre otros [30][66][71].

7 *Incluir el SD a probar*: el *tester* deberá indicarle al sistema previamente generado cual va a ser el SD a probar (SUT), estableciendo los canales de comunicación adecuados para una correcta estimulación del SUT; estos canales se derivan a partir de los modelos generados en el primer paso. Para realizar este paso es necesario apoyarse de la GUI del sistema PRUDIMA.

8 *Ejecutar casos de pruebas*: el *tester* procede a ejecutar a través del prototipo PRUDIMA el conjunto de pruebas, que se encuentran en el sistema de pruebas, generado los respectivos reportes de los veredictos hechos por los oráculos. El enfoque de ejecución utilizado es el *off-line,* ya que para ejecutar las pruebas, estas deben ser generadas previamente. Al contrario del enfoque de ejecución *on-line,* las cuales, se van ejecutando a medida que se modelan [30]. Existen algunas ventajas que corresponden al enfoque escogido, como son el administrar y ejecutar las pruebas con una herramienta software, poder repetir un conjunto de pruebas, la veces que sean necesarias incluso en distintas maquinas, lo cual conlleva a hacer factible las pruebas de regresión [23].

9 *Analizar resultados de los casos de pruebas*: este paso, el cual es común en todos los trabajos incluso en pruebas con enfoques diferentes como manuales, scripting, automáticas, entre otros [30][72]. El *tester* debe verificar el resultado de las pruebas, para reportar las incidencias y detallando los errores que causaron el fallo. Los errores encontrados pueden ser en el SUT, en el modelo de pruebas o en los requerimientos definidos para el sistema. Brindando retroalimentación al área correspondiente, dependiendo del tipo de errores encontrados, las áreas pueden ser diseño, desarrollo o pruebas para que se realicen las correcciones adecuadas.

10 *¿Son todos los casos de pruebas exitosos*? : aquí el *tester* verifica si hay algún resultado fallido, es decir, sí su veredicto es *fail.* Si es así, se debe seguir al paso 10.1, sino, el proceso termina satisfactoriamente.

10.1 *Corregir SUT*: se debe corregir el SUT por parte de los desarrolladores del mismo, después se vuelve al paso 7 y se itera hasta que todos los resultados de los casos de pruebas sean satisfactorios.

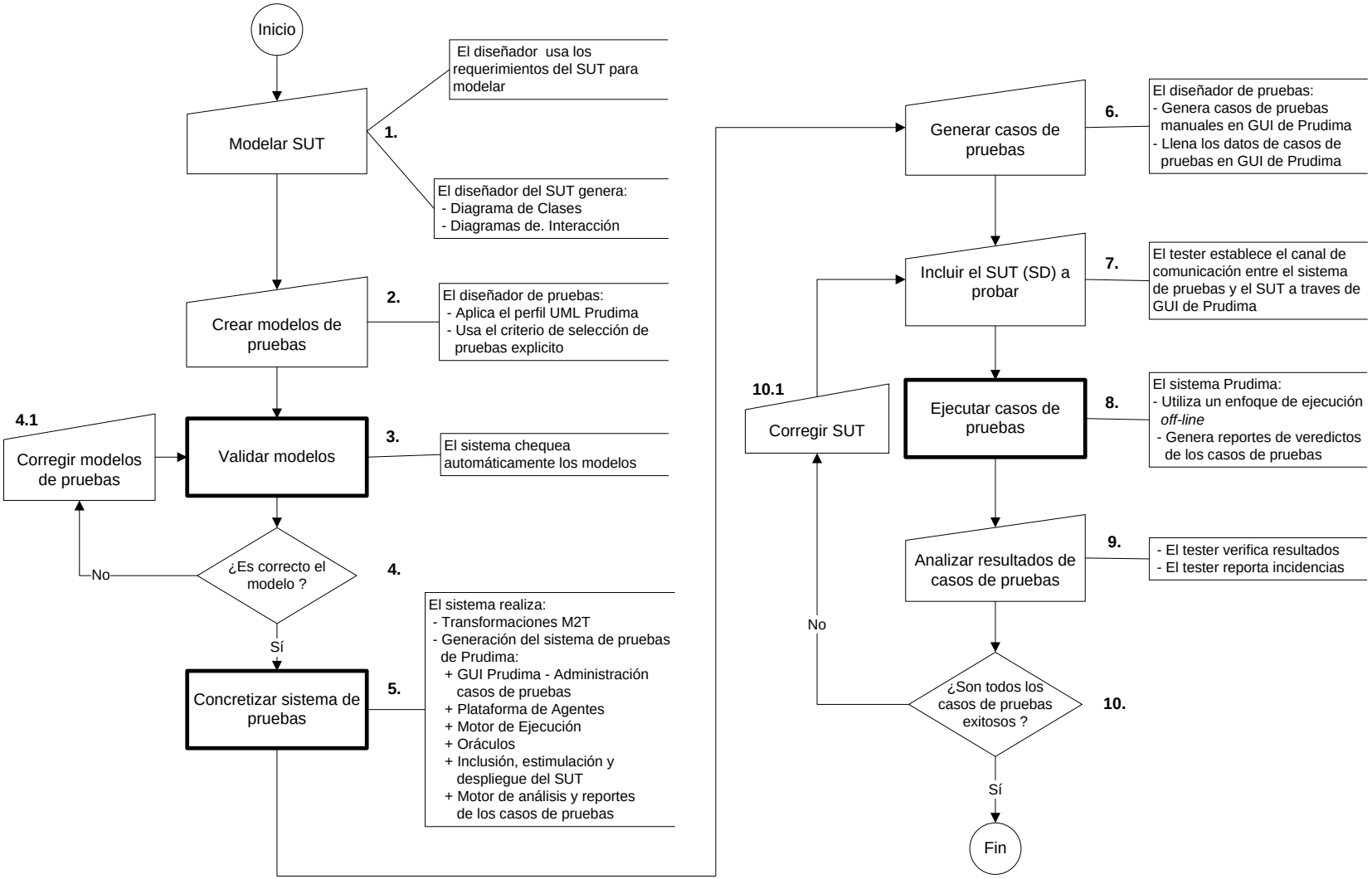

Figura 12 - *Diagrama de flujo del proceso de aplicación de MBT de PRUDIMA*

Capítulo 4

4. PROTOTIPO PRUDIMA

Para utilizar el proceso práctico de pruebas a SD guiado por el enfoque MBT, se requiere de una herramienta software que permita la reutilización de los modelos del sistema y la automatización de tareas manuales, las cuales son propensas a errores y costosas de realizar. Debido a esto se hace necesario construir un prototipo que sirva de apoyo al *tester* para seguir los pasos del proceso de pruebas definido en este trabajo.

En este capítulo se presenta las fases de la metodología de desarrollo de software UP Ágil que se llevaron a cabo para la construcción de la solución software, la cual, soporta el proceso de MBT para pruebas a los SD definido en este trabajo. Se decidió utilizar AUP, ya que esta permite tener rápidamente versiones del prototipo para y realizar mejoras en cada iteración, entre otras ventajas [73]. En la Figura 13, se presenta el calendario del desarrollo del proyecto.

	Nombre	Pr...	Duración	Inicio	Terminado
1	**Fase de Inicio**		5 days	**3/04/13 08:00 AM**	**9/04/13 05:00 PM**
2	Defenición del proyecto		1 day	3/04/13 08:00 AM	3/04/13 05:00 PM
3	Especificación de requerimientos software	2	4 days	4/04/13 08:00 AM	9/04/13 05:00 PM
4	**Fase de Elaboración**	**3**	18 days	**10/04/13 08:00 AM**	**3/05/13 05:00 PM**
5	**Diseño**		18 days	**10/04/13 08:00 AM**	**3/05/13 05:00 PM**
6	Diseño de Casos de Uso		3 days	10/04/13 08:00 AM	12/04/13 05:00 PM
7	Diseño de la Arquitectura	6	10 days	15/04/13 08:00 AM	26/04/13 05:00 PM
8	Diseño de la capa de persistencia	7	5 days	29/04/13 08:00 AM	3/05/13 05:00 PM
9	**Fase de Construcción**	**4**	58 days	**6/05/13 08:00 AM**	**24/07/13 05:00 PM**
10	**Iteración I**		38 days	**6/05/13 08:00 AM**	**26/06/13 05:00 PM**
11	Implementación de la parte Estática		14 days	6/05/13 08:00 AM	23/05/13 05:00 PM
12	Integración plataforma de Agentes JADE	11	8 days	24/05/13 08:00 AM	4/06/13 05:00 PM
13	Implementación de la capa de persistencia	12	5 days	5/06/13 08:00 AM	11/06/13 05:00 PM
14	Implementación de las plantillas	13	5 days	12/06/13 08:00 AM	18/06/13 05:00 PM
15	Implementación de la interfaz gráfica	14	3 days	19/06/13 08:00 AM	21/06/13 05:00 PM
16	Ejecución de pruebas	15	1 day	24/06/13 08:00 AM	24/06/13 05:00 PM
17	Pruebas Alfa	16	2 days	25/06/13 08:00 AM	26/06/13 05:00 PM
18	**Iteración II**		18 days	**1/07/13 08:00 AM**	**24/07/13 05:00 PM**
19	Depuración de las plantillas de Acceleo	25	7 days	1/07/13 08:00 AM	9/07/13 05:00 PM
20	Depuración de los Agentes JADE	19	5 days	10/07/13 08:00 AM	16/07/13 05:00 PM
21	Depuración de la interfaz gráfica	20	2 days	17/07/13 08:00 AM	18/07/13 05:00 PM
22	Generación y depuración de plugins	21	2 days	19/07/13 08:00 AM	22/07/13 05:00 PM
23	Pruebas Alfa	22	2 days	23/07/13 08:00 AM	24/07/13 05:00 PM
24	**Fase de Transición**		34 days	**27/06/13 08:00 AM**	**13/08/13 05:00 PM**
25	Pruebas Beta Privadas	10	2 days	27/06/13 08:00 AM	28/06/13 05:00 PM
26	**Documentación**	**18**	9 days	**25/07/13 08:00 AM**	**6/08/13 05:00 PM**
27	Elaboración del manual de instalación		4 days	25/07/13 08:00 AM	30/07/13 05:00 PM
28	Elaboración del manual de Usuario	27	5 days	31/07/13 08:00 AM	6/08/13 05:00 PM
29	Despliegue de la aplicación	26	5 days	7/08/13 08:00 AM	13/08/13 05:00 PM

Figura 13 - *Línea de tiempo del proyecto de desarrollo*

4.1. FASE DE INICIO

El objetivo de esta fase es identificar el alcance inicial del proyecto de desarrollo, revisar una posible arquitectura del sistema y obtener los recursos iniciales para iniciar el proyecto. Para lo anterior, se realizaron reuniones donde se expresan las necesidades y alcances que la solución debe tener, a partir del conocimiento de los *stakeholders* del proyecto, que para este trabajo son los autores del mismo y los tutores.

4.1.1. Modelado de requisitos

Después de realizar el proceso de definición del enfoque MBT a seguir, en la Sección 3.3, se identificaron los requerimientos iniciales y los resultantes de la recolección de las expectativas del usuario frente a las funcionalidades del prototipo.

El prototipo tiene como objetivo brindarle al usuario una herramienta de ayuda para implementar el enfoque MBT. Para esto es necesario que la herramienta realice las transformaciones a los modelos creados por el usuario. El resultado de estas transformaciones, será una aplicación software encargada de gestionar los casos de prueba, los datos de las pruebas, la ejecución de las pruebas y los reportes de las mismas.

Lo anterior permitió iniciar con la especificación de los requisitos del prototipo, su clasificación y priorización.

Descripción de los requisitos

A continuación se presenta una plantilla de los requerimientos recolectados para el prototipo de la herramienta CASE (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Computadora) con soporte al proceso de MBT definido en el presente trabajo, para ver el resto de requerimientos, vea el Anexo G.

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 [18], 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 16 - *Requerimiento Generar prueba*

4.1.2. Definición de requisitos funcionales y no funcionales

A partir de la lista de requisitos expresadas anteriormente, se realizó el procesamiento de la información capturada en el proceso de levantamiento de requisitos realizado en las entrevistas entre los *stakeholders* del proyecto, dando como resultado, la generación de un documento de especificación de requerimientos software (en sus siglas en ingles *SRS*), en el cual, se formaliza la actividad de levantamiento de requerimientos. Este documento se puede encontrar en el Anexo G.

Lista de 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.

Lista de 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.

4.2. FASE DE ELABORACIÓN

El objetivo de esta fase es definir la arquitectura de la herramienta a desarrollar. En este punto el grupo de trabajo verifica que realmente se puede desarrollar un sistema que satisfaga los requisitos, a través de un esqueleto del sistema llamado "Prototipo de la arquitectura" [74]. Se realizó entonces, el modelado de los casos de uso en formatos de alto nivel, ver Anexo G.

4.2.1. Definición de actores

Entre las Tabla 17 y la Tabla 19, se presentan una breve descripción de los actores que interactúan con el sistema, así como las restricciones y sus responsabilidades.

Actor	Modelador
Fuente	Documento de especificación de requerimientos (SRS). Anexo G.
Descripción	Un usuario 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 definidos en el documento de especificación de requerimientos (*SRS*).

Tabla 17 - *Descripción Modelador*

Actor	*Tester*
Fuente	Documento de especificación de requerimientos (SRS). Anexo G.
Descripción	Una persona con este rol, podrá ejecutar las pruebas generadas por el modelador y realizar la carga de los datos de los diferentes casos de pruebas, con los valores que el considere necesarios para lograr el cubrimiento de los casos de pruebas contra el SUT.

Tabla 18 - *Descripción Tester*

Actor	Analista
Fuente	Documento de especificación de requerimientos (SRS). Anexo G.
Descripción	La persona con este rol, es el encargado de analizar el informe

	entregado por el *Tester* de las pruebas ejecutadas contra el SUT. Cuyo análisis, comprende el cubrimiento de las pruebas y la correlación con los requerimientos del SUT.

Tabla 19 - *Descripción Analista*

4.2.2. Casos de uso

En esta sección del documento se presentará el comportamiento del sistema con base en las relaciones entre los componentes del sistema y los actores. A continuación en la Figura 14, se presenta el diagrama de casos de uso, en la cual, se ilustra una vista conceptual que permitirá conocer los actores del sistema y los casos de uso con los cuales interactúan.

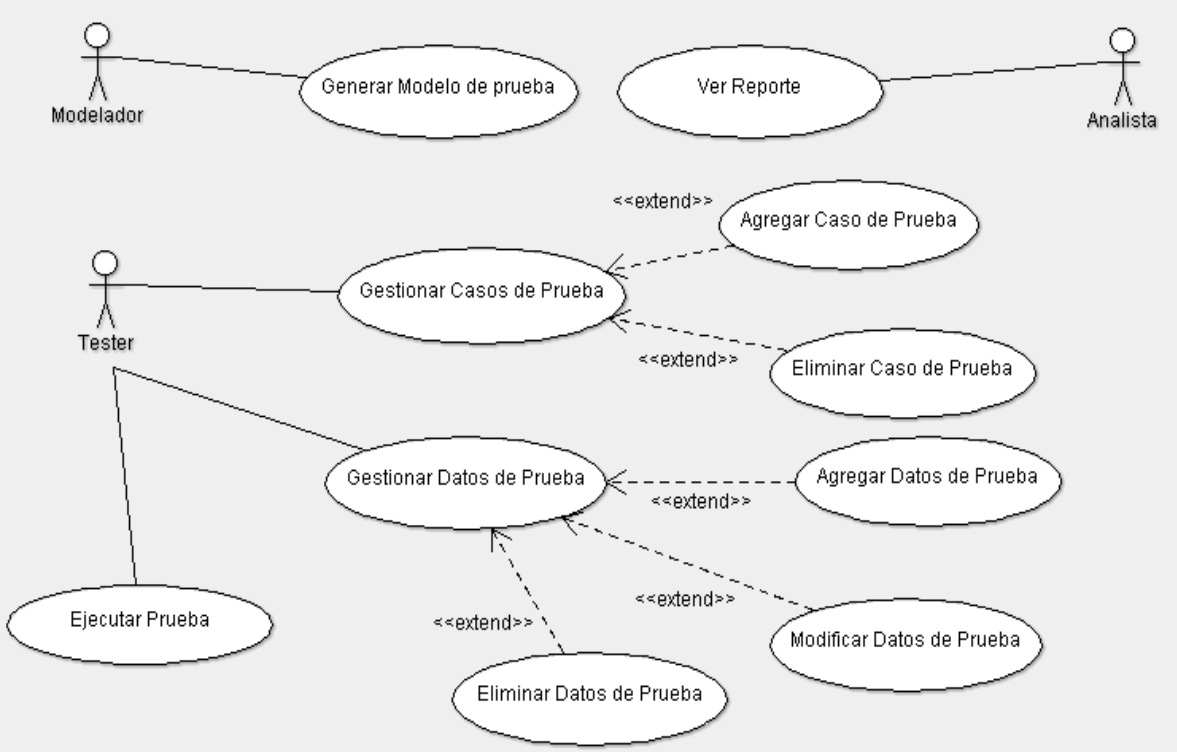

Figura 14 - *Diagrama de casos de uso de PRUDIMA*

4.2.3. Arquitectura del prototipo software

La arquitectura inicial fue planteada de tal manera que todo el código fuente de la herramienta fuese generado dinámicamente a partir de los modelos de pruebas del SUT. Sin embargo, analizando rigurosamente, se percibió que existían componentes, como la UI del usuario, que deberían ser independientes a los modelos de pruebas, concibiendo así una arquitectura estática que provea los servicios necesarios para ser consumidos por la parte dinámica. Por lo tanto, la

dinámica quedó dependiente de los modelos de pruebas creados por el actor Modelador y la parte estática se definió como se puede apreciar en el diagrama de componentes de la Figura 15.

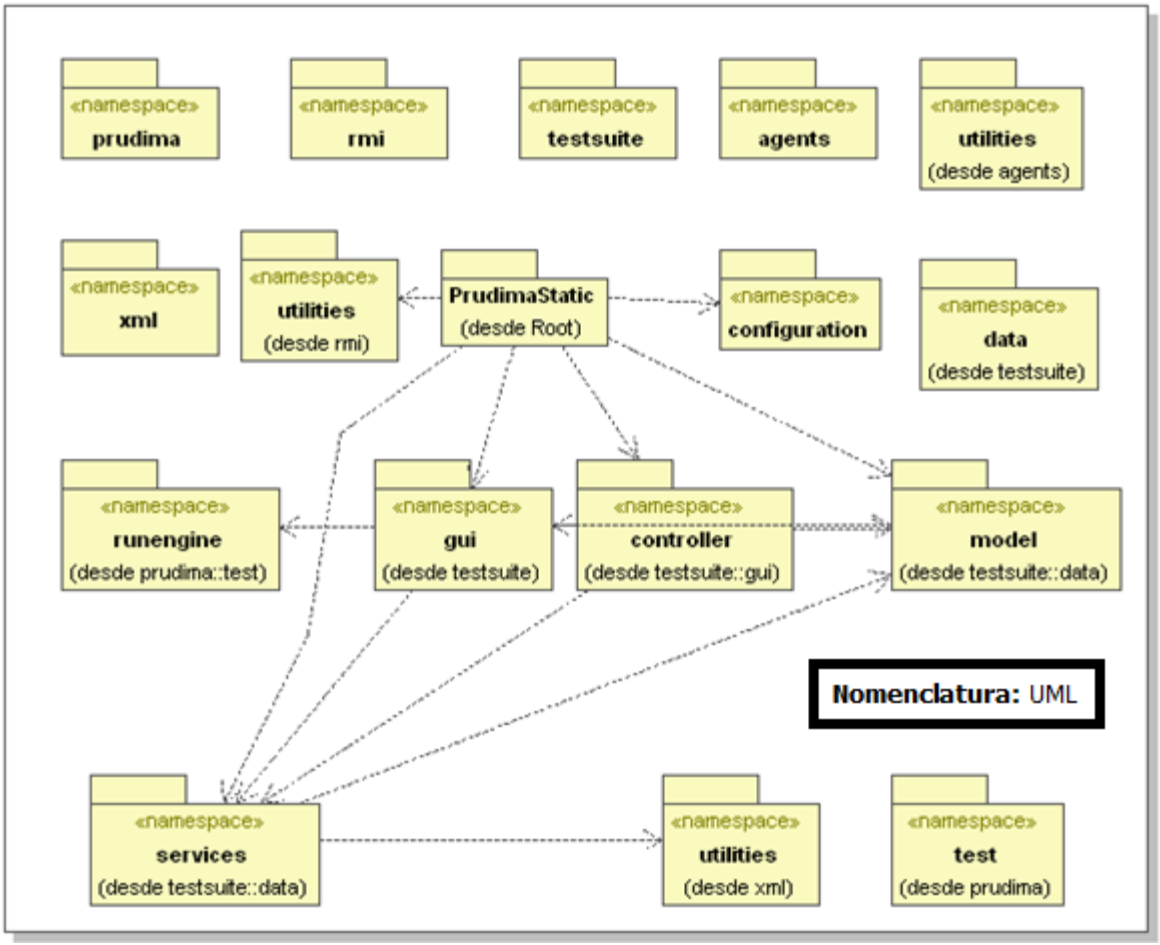

Figura 15 - *Diagrama de paquetes de la implementación de PE*

4.2.4. Capas lógicas de la solución

En esta sección se presentará la estructura de la arquitectura dividida en capas; en las diferentes secciones se presentan las capas y los componentes más relevantes de la aplicación encargada de realizar las pruebas al SUT.

PRUDIMA está constituido por tres grandes capas: la **capa de interfaz de usuario**, la **capa de servicios de aplicación** y la **plataforma de agentes**, como se muestran en la Figura 16. Donde en nivel más bajo es la plataforma de agentes y de ahí en adelante cada una de estas brinda servicios a la capa contigua superior.

Capa interfaz de usuario

La capa de interfaz de usuario es la primera en orden descendente, ya que el usuario interactúa directamente con ésta, permitiéndole modelar gráficamente los casos de pruebas, incluir configuraciones para los datos de pruebas, la ejecución de las pruebas y la visualización de los reportes. Para esta capa se utiliza el entorno de desarrollo Eclipse junto al entorno de modelamiento *Papyrus*, el primero por ser una plataforma gratuita para desarrollo en Java y el segundo además de ser gratuita implementa la especificación *OMG* para UML en su versión 2 y también a su completa integración con el IDE de *Eclipse*.

El modelamiento de los casos de prueba se realiza bajo el entorno Papyrus el cual, provee editores de diagramas para lenguajes de modelamiento basados en EMF entre los cuales está UML. Los diagramas utilizados por PRUDIMA son el de clases y los de secuencia, ya que estos permiten tener tanto una visión estática como dinámica del sistema. En los diagramas de secuencia es donde se plasman los diseños de las pruebas realizadas por parte del *tester*. Papyrus permite utilizar el Perfil UML de PRUDIMA, diseñado para las pruebas a SD.

La configuración para los datos de pruebas, es otra actividad que debe realizar el *tester*, donde se definen los conjuntos de valores de los parámetros utilizados en los métodos del SUT, que podrán ser consumidos por parte del oráculo de pruebas.

Capa de servicios de aplicación

En esta capa se tienen los servicios de transformaciones, gestión de casos y datos de pruebas, servicios de despliegue y ejecución, servicios de análisis y generación de reportes. Tiene como insumo los diagramas realizados previamente y las configuraciones dadas a través de la UI. Aquí se generan varios archivos que permiten mantener un estado para el sistema de pruebas.

Capa de plataforma de agentes

La base conceptual para la realización de las pruebas en un SD dentro de PRUDIMA, es la utilización de procesos individuales que son ejecutados en diferentes nodos, configurados por el usuario. Dichos procesos se comunican entre ellos y son controlados por un proceso controlador que por medio del envío de mensajes, indican al proceso receptor de dicho mensaje las acciones que se debe realizar. Esto con el fin, de ejecutar los casos de pruebas estimulando al SUT para encontrar posibles errores en el mismo [75].

En el proceso de búsqueda de tecnologías que permitieran un proceso de desarrollo ágil del concepto anterior, se encontraron algunos trabajos [76], que

presentaron ventajas de soportarse sobre una arquitectura basada en agentes, como por ejemplo la monitorización de errores en nodos específicos a través del paso de mensajes.

Por lo tanto, en este trabajo se escogió a *Java Agent DEvelopment Framework* (JADE) como la plataforma sobre la cual se realiza la distribución de los actores de PRUDIMA, ya que es una plataforma de agentes de código abierto y es ampliamente usada [77][78][79]. Esta plataforma nos permite desplegar de una manera ágil agentes en cada nodo del SD y comunicar los resultados a un agente controlador especial que luego los centralizará y generará un reporte para el *tester*.

Figura 16 - *Capas de la solución PRUDIMA*

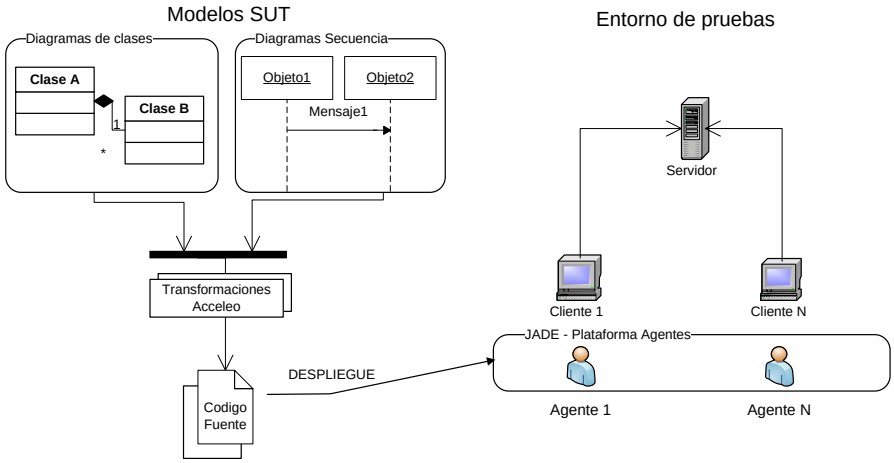

Figura 17 - *Proceso de transformación y ejecución*

4.3. FASE DE CONSTRUCCIÓN

En esta fase el objetivo es construir el software ejecutable de una manera iterativa e incremental que satisfaga los requerimientos de mayor prioridad para los *stakeholders* del proyecto.

Previo al desarrollo del prototipo se realizó una inspección de las tecnologías disponibles para el desarrollo de la herramienta.
A continuación presentamos una lista de tecnologías que permitió realizar el proceso de construcción durante todas las iteraciones.

Lenguaje de MOFScript*:* Se realizó una comparación de las principales lenguajes de *Scripting* (ver Anexo E) que se encontraron durante este trabajo de investigación. De esta comparación se seleccionó el lenguaje *Acceleo*, debido a su facilidad de uso, la documentación disponible y la posibilidad de separar los módulos generados necesarios en diferentes plantillas.

Lenguaje de Modelado: para el lenguaje de modelado se resolvió utilizar UML en su versión 2, ya que dispone de elementos que nos ayuda a expresar las diferentes vistas del sistema a probar, vista estática (diagramas de clases) y dinámica (diagramas de secuencia), como también, a modelar las pruebas para dicho sistema. Otra razón de la elección fue que permite extender los anteriores modelos a partir de los perfiles UML, lo cual permite definir una semántica adicional a los elementos de los modelos.

Herramientas de desarrollo: para el desarrollo del prototipo, se seleccionó Eclipse en su versión 4.3 nombrada Juno, ya que posee una licencia EPL [80] la cual permite utilizar, modificar, copiar y distribuir el trabajo. Este IDE presenta una interfaz con un alto grado de madurez, además de contar con la funcionalidad de resaltar la sintaxis, compilación en tiempo real y gran cantidad de *plugins*

disponibles elaborados por la comunidad. También se tuvo en cuenta la experiencia de manejo de este IDE por parte de los participantes del proyecto, tanto de los desarrolladores como usuarios finales del prototipo. Otro punto importante en el momento de la selección fue la documentación disponible en línea a través de manuales y foros.

Herramienta de modelado: se realizó una comparación entre las principales herramientas de modelado, se puede encontrar más detalles en el Anexo H. En esta comparación, se buscó una herramienta que se integre con el IDE de trabajo seleccionado esto con el fin de generar una herramienta integral, para la ejecución de la pruebas. Además se tuvieron en cuenta los criterios de licenciamiento y estado de madurez del proyecto. Como consecuencia se obtuvo que Papyrus como la mejor candidata.

Lenguaje de programación: se seleccionó como lenguaje de programación *Java*, debido este soporta la tecnología RMI, tecnología en la cual están realizadas las prácticas de la asignatura “Laboratorio de SD”. *Java* también soporta el uso de agentes por medio de la plataforma JADE, y la posibilidad de utilizar *Wrappers* para realizar operaciones complejas de implementar en las plantillas de *Acceleo*.

Librerías gráficas: se seleccionó Java Swing para realizar la interfaz de usuario, encargado de generar las cajas de textos, botones y demás elementos gráficos que permiten interactuar con el sistema. Se utilizara la herramienta *WindowsBuilder*, la cual permite realizar de una manera fácil la generación de los diferentes formularios ya que nos brinda un diseñador gráfico con la funcionalidad de *drag-and-drop*, entre muchas otras características.

Para la construcción del prototipo con soporte al procedimiento de MBT definido en el presente trabajo, se realizaron dos iteraciones:

- **Primera iteración**: aquí se codificaron los componentes necesarios para el funcionamiento básico del prototipo, cumpliendo con todos los requerimientos definidos en la fase de elaboración. Luego, en la primera iteración de la fase de transición, se realizaron algunas pruebas las cuales, dieron las pautas necesarias para el inicio de una nueva iteración.
- **Segunda iteración**: se enfocó en realizar mejoras tanto en facilidad de uso como en los motores de ejecución de pruebas.

4.3.1. Primera iteración

En esta iteración, se realizó en primera instancia la implementación de la parte estática del prototipo, para lo cual se procedió a implementar las diferentes funcionalidades que permitiesen tener un conjunto de servicios consumibles. Se

presenta en la Figura 18, una vista en resumen del diagrama de clases de dicha parte.

Luego, se empaquetó el componente estático en un archivo *.jar*, el cual permite ser agregado a los proyectos generados dinámicamente por el prototipo. Cuya finalidad es prestar los servicios transversales a cada uno de estos proyectos.
Seguido, se realizó el desarrollo de los servicios para la plataforma de agentes encargada de generar los clientes, quienes invocaran remotamente los servicios prestados por el SUT. Para esto, se siguió el diagrama de clases de la Figura 19, donde se aprecian las principales clases y sus relaciones, de manera que brindan un soporte para los agentes desplegados en el momento de la ejecución de las pruebas contra el SUT

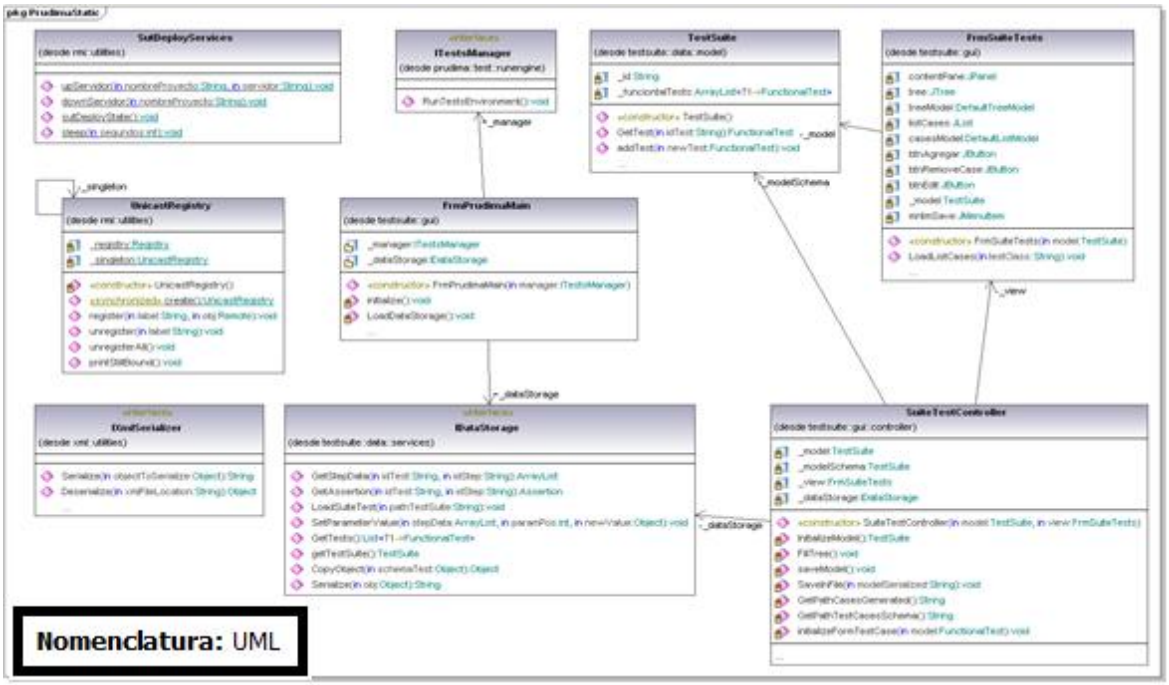

Figura 18 - *Diagrama de clases de persistencia y despliegue de pruebas de PRUDIMA*

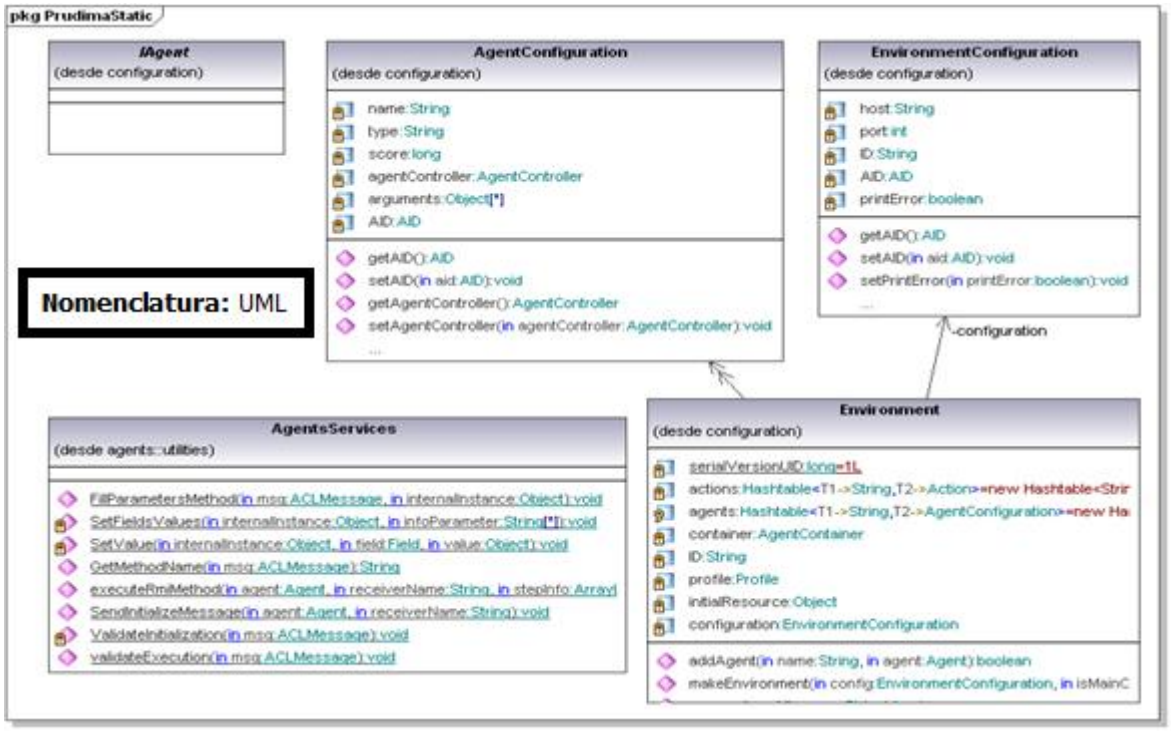

Figura 19 - *Vista del diagrama de clases de Agentes*

A continuación al desarrollo de la plataforma de agentes, se implementó toda la parte gráfica del prototipo, encargada de la gestión de los casos y datos de pruebas, de la ejecución de las pruebas y de la visualización de los reportes de los veredictos realizados por el prototipo. Para esto se generaron los formularios, presentados en la Figura 20 y Figura 21.

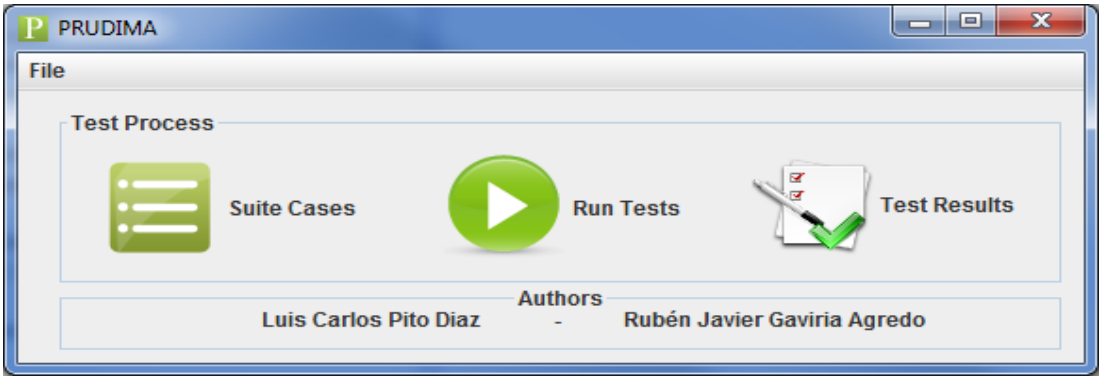

Figura 20 - *Menú Principal PRUDIMA*

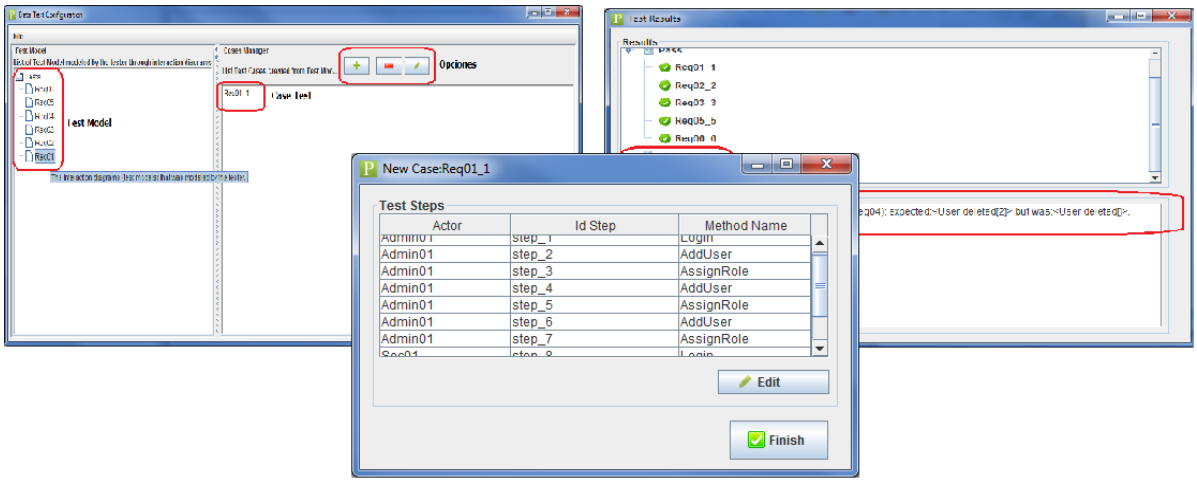

Figura 21 - *Formularios del prototipo PRUDIMA*

En seguida, se realizó la implementación de la capa de persistencia de datos, encargada de almacenar tanto los datos de los casos de pruebas en archivos en formato XML como los resultados de la ejecución de las pruebas en archivos con formato TXT.

Como se mencionó en la Sección 2.2.1, se implementaron en paralelo las plantillas en *Acceleo* encargadas transformar los modelos de pruebas diseñados en código fuente *Java* del componente dinámico del prototipo. Para esto, se siguió la arquitectura y la descripción de las plantillas expuestas en la Sección 3.2. Finalmente, se realizó el componente encargado de visualizar los reportes de las pruebas, la función de este, es tomar los reportes generados y presentárselos al usuario por medio de la UI.

4.3.2. Segunda iteración

Para realizar la segunda iteración se tuvieron en cuenta las recomendaciones realizadas por parte de los participantes de las pruebas beta, además de las mejoras a las funcionalidades de la herramienta.

Primero se realizó, la depuración de las funcionalidades de las plantillas de *Acceleo*, en busca de partes de código que presentaran problemas de rendimiento, procediendo a realizar las respectivas correcciones para mejorar los tiempos de generación del código de la herramienta. Además se realizó un módulo encargado de verificar que los estereotipos necesarios estén aplicados a los elementos del modelo de pruebas.

También se realizó una mejora en la implementación interna del componente de agentes logrando simplificar la arquitectura de dicha funcionalidad y la reutilización de código. Asimismo, se trabajó en la implementación de las mejoras solicitadas en la interfaz gráfica, agregando múltiples iconos que trataran de guiar al usuario en la acción que realiza cada componente gráfico.

Se implementaron luego los *plugins* "***co.edu.unicauca.prudima.acceleo.ui***" y "***co.edu.unicauca.prudima.xml.ui***", con lo anterior se busca obtener un mecanismo que facilite el uso de las transformaciones para el usuario final, por medio de la arquitectura de *plugins* de Eclipse, véase Figura 22, los cuales, permiten la funcionalidad de agregar una opción al menú emergente, que aparece cada vez que damos clic derecho sobre un archivo con extensión UML y que permite invocar las transformaciones de PRUDIMA construidas en *Acceleo*.

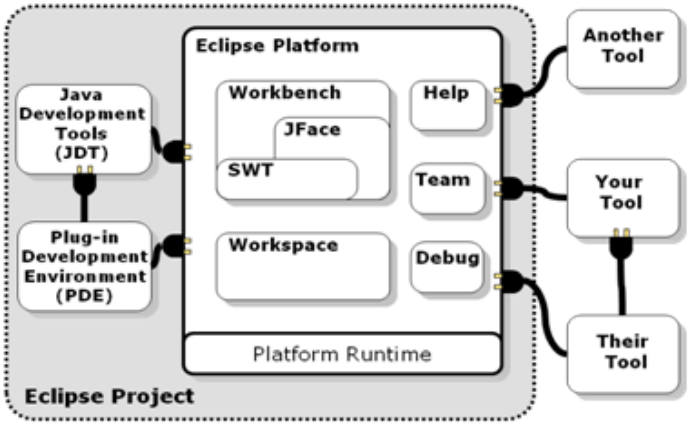

Figura 22 - *Arquitectura de plugins de Eclipse*

4.4. FASE DE TRANSICIÓN

El objetivo de esta fase es validar y desplegar la solución construida en un entorno de producción.

4.4.1. Pruebas de software

Todo software conlleva un proceso de pruebas los cuales parten desde la etapa de desarrollo y finalizan con las pruebas de aceptación de los clientes del producto. Esta fase consumió bastante tiempo, alrededor de 25 días del grupo de desarrollo, para poder tener una versión que pudiese ser puesta en producción. A continuación se describen las distintas iteraciones que se llevaron a cabo para validar la solución.

Al finalizar la primera fase de construcción, se procedió a realizar las pruebas *Alfa*, las cuales, se enfocaron en las pruebas de sistema y que permitieron validar los requerimientos definidos en las fases anteriores. Sin embargo, se encontraron algunos defectos a nivel de interfaz de usuario y fallos menores respecto a la capa de almacenamiento de datos, los cuales se pudieron corregir enseguida. Una vez superada esta etapa, se generó una versión *Beta* y al mismo tiempo la documentación requerida para los usuarios finales, entre los cuales, están el manual de usuario e instalación, estos manuales se encuentran en forma de videos publicados en el sistema de videos *Youtube* llamados: Como configurar proyecto Prudima[13] y Modelando con Papyrus[14].

Posteriormente, la versión *Beta* fue probada por personas externas al proyecto con conocimiento en SD y pruebas manuales de los mismos, lo que permitió abordar el proceso de pruebas fácilmente. No obstante, fue necesario realizar una inducción del enfoque MBT para contextualizar a las personas en el uso de la herramienta y lo que se esperaba de esta.

Las pruebas se enfocaron en: funcionalidad, *performance* y facilidad de uso. El proceso se logró culminar a partir de la utilización de una guía que abarcaba todas las funcionalidades del sistema PRUDIMA y que permitía tener una percepción completa del mismo utilizando una encuesta. Sin embargo, para la evaluación de los conceptos de *performance* y facilidad de uso no se siguió ningún estándar de pruebas, sino, que fueron dejados a criterio de cada una de las personas que evaluaba la herramienta.

Terminado este ciclo de pruebas, se obtuvo retroalimentación con las siguientes incidencias:

- Mejorar los textos de la interfaz gráfica donde se creaban los casos de pruebas
- Mejorar el rendimiento de la tarea automática que desplegaba el SD
- Colocar iconos que fuesen más dicientes en toda la aplicación
- Validar que los estereotipos necesarios se encuentren aplicados al modelo de pruebas

En la segunda iteración se ajustó el sistema corrigiendo los defectos encontrados, siendo liberada la versión final para ponerla en producción. Luego se procedió a realizar las pruebas de aceptación por parte de los *stakeholders* del proyecto,

[13] Cómo configurar proyecto Prudima
Video tutorial: http://www.youtube.com/watch?v=qXZaBLMGsBM
[14] Modelando con Papyrus
Video tutorial: http://www.youtube.com/watch?v=RZXu8KaV080

quienes dieron su visto bueno y finalmente, la versión quedó lista para el proceso de despliegue.

4.4.2. Despliegue del prototipo de la herramienta PRUDIMA

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, ver Figura 23.

Para esto, se definieron los siguientes pasos, para más detalle ver Anexo K:

1. Copiar los *plugins* de PRUDIMA dentro de la carpeta *plugins* de la instalación de Eclipse:
    - Wizard
    - Metamodelos
    - Transformaciónes M2T

2. Copiar la carpeta *libraries* de PRUDIMA dentro de la carpeta plugins de Eclipse. 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

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

Se encontraron algunos problemas en los despliegues con versiones de SO de 32 bits por lo que se tuvo que realizar los ajustes necesarios en las maquinas objetivo. Lo cual llevo a realizar un análisis de identificación de los requerimientos mínimos respecto al *hardware y software* base, sobre el cual la aplicación tendría su óptimo desempeño, resultando en la siguiente lista de requisitos:

Requisitos mínimos de *hardware:*

- Procesador de 1 GHz
- Memoria RAM de 512 Mg
- Disco duro de 20 Gg (Incluido el SO de Windows 7)
- NIC de 100 Mb

Requisitos mínimos de *software*:

- Sistema Operativo Windows XP, Windows 7.
- Plataforma Java en su versión 1.7.0_25

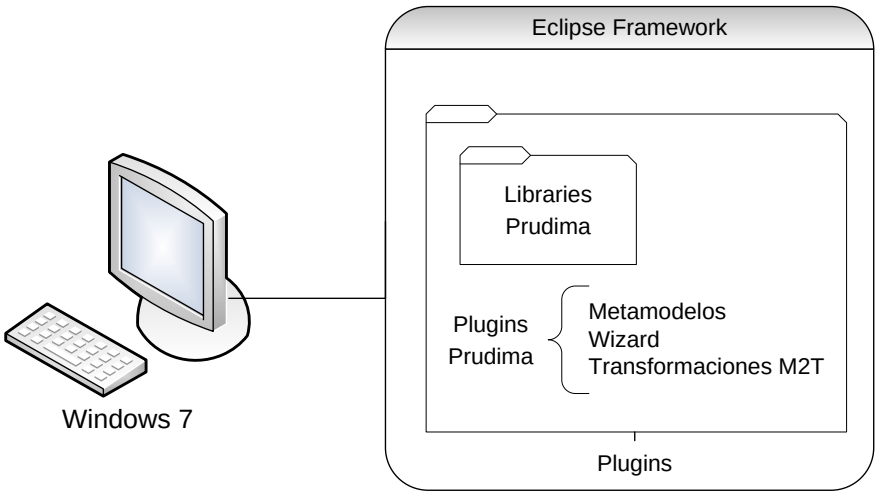

Figura 23 - *Vista de despliegue de PRUDIMA*

Se utilizó una sala de computo de la Universidad del Cauca para el proceso de despliegues, donde se verificaron los requisitos de diez computadores, los cuales no todos cumplían con lo mínimo necesario, por ejemplo se encontraron versiones de Java antiguas, por lo cual, se debieron hacer las instalaciones correspondientes de lo que faltaba. Con esto se finalizó tanto la fase de transición como también el proceso de desarrollo del prototipo de la herramienta PRUDIMA.

Capítulo 5

5. ESTUDIO DE CASO – PRUDIMA

En el presente capítulo se expone un estudio empírico, realizado con la finalidad de validar los elementos construidos a lo largo de la investigación, a saber, el proceso definido de aplicación del enfoque MBT a pruebas de SD en un entorno académico, el uso del perfil UML que agrupa los metamodelos tanto para SD como para PSD y el prototipo PRUDIMA.

5.1. METODOLOGÍA

Para llevar a cabo este estudio, se utilizaron los lineamientos definidos en [81], ya que es una metodología de investigación adecuada en el campo de ingeniería de software, la cual, permite observar y medir las unidades de análisis en un contexto real, manteniendo la integridad y las características significativas de los eventos. A continuación, se describen los pasos llevados a cabo.

Diseño: el estudio de caso fue definido para validar el proceso de MBT definido en este trabajo, el cual utiliza un perfil UML que agrupa los metamodelos de SD y PSD, a través del prototipo PRUDIMA, que ayuda a guiar el proceso paso a paso. Para esto, se escogió el tipo de estudio de caso, se definieron métricas e indicadores, que permitieran analizar posteriormente los datos obtenidos durante la ejecución del estudio y se determinaron las formas de recolección de los datos.

La forma de validar el proceso, es comparando los enfoques MBT y manual. Esto porque actualmente la evaluación de las prácticas de laboratorio de SD sigue un proceso de pruebas manual. Es por esto, que este estudio quiere comprobar si el uso de un enfoque dirigido por modelos para dicho proceso, permite disminuir los costos asociados al tiempo de evaluación.

Preparación: se definieron los instrumentos necesarios tanto para el desarrollo del caso como para la recolección de los datos durante su ejecución, que fueron:

- Guías para realizar los procesos de pruebas manuales y MBT, este último con la ayuda del prototipo PRUDIMA, ver Anexo I
- Reportes entregados por el prototipo PRUDIMA
- El formato de encuesta

Ejecución y recolección de los datos: la ejecución fué llevada a cabo en dos sesiones, en las cuales participaron diferentes grupos de personas, que fueron capacitadas para realizar el proceso de pruebas MBT y en el uso del prototipo PRUDIMA. En dichas sesiones se entregaron las guías para el desarrollo de la práctica. En la primera sesión debían llevar a cabo el proceso completo de MBT y en la segunda sesión únicamente se dedicaron a probar un SD utilizando dos enfoques de pruebas, el manual y MBT. Los datos recolectados fueron a través de los reportes generados por el prototipo, como también por plantillas en Excel que las personas debían llenar. Todo lo anterior bajo el monitoreo por parte de los miembros del equipo de investigación.

Análisis: para este paso se realizó un análisis cuantitativo y cualitativo de los datos recolectados durante la ejecución del caso.

Reporte: se reportan los hallazgos encontrados con los datos obtenidos en el paso previo.

5.2. PREGUNTA DE INVESTIGACIÓN

El prototipo PRUDIMA es una herramienta CASE, que soporta el proceso P-MBT-SD definido en este trabajo. Esta herramienta permite definir modelos de pruebas utilizando UML, haciendo uso de los diagramas de clases y de secuencia. A estos diagramas, se les aplica el perfil UML de PRUDIMA definido para las pruebas a SD, para luego transformar automáticamente dichos modelos en un formato ejecutable, obteniendo así el sistema de pruebas que permite desplegar y ejecutar las pruebas obteniendo sus resultados.

En busca de mitigar los problemas de calidad que conlleva el uso del proceso de pruebas manuales, por ser propenso a errores y por el alto consumo de recursos tanto en tiempo como en recurso humano, se plantea la siguiente pregunta de investigación: *¿Se puede mejorar el proceso de pruebas aplicado a las prácticas de LSD siguiendo un enfoque MBT reduciendo el consumo del recurso tiempo*?

5.3. OBJETIVO DEL ESTUDIO DE CASO

El objetivo de este caso de estudio es validar la utilización de los metamodelos de SD y PSD, el perfil UML que los agrupa y el proceso de pruebas con enfoque MBT, a través del uso del prototipo PRUDIMA, para modelar, ejecutar y analizar pruebas a un SD, donde todos los elementos mencionados anteriormente fueron desarrollados por los autores del presente trabajo. Para esto, se realizarán dos sesiones prácticas, en las cuales, se evaluará el proceso de MBT y se compararán los resultados obtenidos de este enfoque con los del trabajo de López [82], que tiene los reportes de los tiempos de pruebas manuales aplicados a la evaluación

de prácticas en LSD, dentro del contexto académico del curso de SD de la Universidad del Cauca.

5.4. SELECCIÓN DEL ESTUDIO DE CASO

Ya que la metodología requiere tener interpretaciones integrales de un caso y sus relaciones con otras variables, en este trabajo se analiza un solo caso con el propósito de obtener un análisis bien detallado. Por lo tanto, se selecciona un estudio de caso del tipo holístico, siendo crítico y revelador.

La unidad de análisis para este caso, es la aplicación del proceso MBT para SD en un contexto académico [83]. El proceso de pruebas a SD es llevado a cabo por ingenieros con conocimientos en el área, por lo tanto, son ellos los que pueden evaluar el proceso para MBT propuesto y su selección corresponde a los criterios de disponibilidad [84] de ingenieros o estudiantes de ingeniera que tuviesen conocimientos en pruebas manuales a SD, quienes serán las principales fuentes de información.

5.5. CONTEXTO DEL ESTUDIO DE CASO

5.5.1. La organización

El LSD es una asignatura dictada en séptimo semestre de la carrera de Ingeniería de Sistemas, la cual aplica los conceptos de la asignatura de Sistemas Distribuidos, por medio de desarrollo de prácticas de ejercicios de desarrollo, de aplicaciones distribuidas, en los lenguajes y herramientas más conocida como: *RPC, RMI, CORBA y DCOM*.

5.5.2. El caso y sus sujetos de investigación

El presente estudio de caso es de tipo holístico [85], teniendo en cuenta los modelos de pruebas y la fuente primaria de información, en este caso dos grupos de estudiantes de Ingeniería de Sistemas, quienes han aprobado satisfactoriamente la asignatura de LSD, en semestres anteriores. Cabe resaltar que estos estudiantes poseen la experiencia en el desarrollo y aplicación de pruebas de las aplicaciones que se realizan en LSD. Este estudio de caso, tiene un propósito positivista [86], ya que busca probar que existe una mejora en la aplicación de un enfoque de pruebas por medio de MBT, en comparación con el enfoque manual que se viene realizando. Con el objetivo de aplicar el proceso definido en el presente trabajo, se realizan dos sesiones de trabajo, las cuales serán dirigidas por medio de una guía, en las que realizara una especificación el sistema a probar y los pasos a seguir para practicar el procedimiento MBT y el procedimiento de pruebas manuales.

5.6. INDICADORES Y MEDICIONES

Para evaluar objetivamente este estudio de caso, se definieron un conjunto de métricas e indicadores que permitieran responder la pregunta de investigación, en la Tabla 20 se presentan en resumen.

Pregunta de investigación	Indicador	Mediciones	Instrumentos
¿Se puede mejorar el proceso de pruebas aplicado a las prácticas de LSD siguiendo un enfoque MBT reduciendo el consumo del recurso tiempo?	Eficacia	Número de pruebas exitosas frente a la cantidad de pruebas ejecutadas	Reportes prototipo. Protocolo de observación
	Cobertura de las pruebas definidas	Cantidad de casos de pruebas ejecutados sobre los casos de pruebas definidos	Reportes prototipo. Protocolo de observación
	Eficiencia	Cantidad de pruebas realizadas en un determinado periodo de tiempo	Reportes prototipo. Protocolo de observación
	Facilidad de uso	Facilidad de uso de la aplicación del proceso con enfoque MBT definido.	Encuesta de percepción del proceso MBT

Tabla 20 - *Resumen de indicadores y mediciones identificados*

A continuación se detallan cada una de los indicadores identificados y su mediciones asociadas a este estudio de caso, basados en [87]:

Eficacia: se define como el análisis del cumplimento de los requisitos definidos.

$$Ef = \frac{Ce}{Cx}$$

Donde *Ef* es la eficiencia, *Ce* es el número total de casos de pruebas exitosos, *Cx* es el número total de casos de prueba ejecutados.

Cobertura de las pruebas definidas: se define como, el grado de cubrimiento de las pruebas definidas.

$$Cx = \frac{Px}{Pd}$$

Donde Cx es la cobertura de las pruebas definidas, Px es el número de pruebas ejecutadas, Pd es el número de pruebas definidas.

Eficiencia: se define como el grado en que los objetivos se cumplen con el menor costo posible.

$$E = \frac{Px}{t}$$

Donde E es la eficiencia, Px es el número de pruebas ejecutadas, t es el tiempo total de las pruebas.

Facilidad de uso: la facilidad de uso se ha definido como el grado en que un producto puede ser usado por determinados usuarios para lograr sus propósitos con eficacia, eficiencia y satisfacción, en un contexto de uso específico, siendo en este caso medido el uso del proceso MBT, definido en este trabajo para las pruebas a SD. La fórmula que se definió para hacer el cálculo de la facilidad de uso percibida es:

$$F = \frac{\sum_{i=1}^{n} \frac{RE_i}{RD_i}}{n}$$

Donde F es la facilidad de uso, REi es el resultado evaluado para la pregunta i, el cual puede tomar valores de uno a cinco, RDi es el resultado deseado en la encuesta a la pregunta i, cuyo resultado esperado es 5, n es el número de preguntas evaluadas en la encuesta.

5.7. EJECUCIÓN DEL ESTUDIO DE CASO

Como ya se había mencionado anteriormente se realizaron dos sesiones que se describirán a continuación:

Primera sesión

Inicialmente se realizó a los participantes presentados en la Figura 24, una introducción del enfoque MBT por medio de una presentación, como también la

explicación de la práctica a realizar. Luego, se procedió a realizar una capacitación sobre la utilización del entorno del prototipo PRUDIMA implementado en este trabajo. Una vez finalizada la capacitación se realizaron los pasos indicados en la guía para esta sesión, donde se establece que primero se debe realizar el modelado del SUT y de las pruebas a ejecutar, teniendo en cuenta los requerimientos especificados dentro de la guía. En seguida, se debe realizar las transformaciones de modelo a texto para la generación de la aplicación que ejecutara la prueba y posteriormente, se solicitara realizar la carga de los datos a través del prototipo PRUDIMA. Finalmente deben realizar la ejecución de las pruebas para seis SUT.

Figura 24 - *Participantes de la primera sesión*

Segunda sesión

En la primera parte de esta sesión, se realizó una introducción de los enfoques MBT y manual, además de mostrar la guía a realizar, como se observa en la Figura 25. Posteriormente se realiza una corta capacitación del prototipo PRUDIMA implementado en este trabajo y del enfoque de pruebas manuales. Luego, se procederá a realizar una distribución al azar de los estudiantes, dentro de los grupos presentados en la Tabla 21. El participante deberá aplicar el trabajo en el orden designado para su grupo. Por ejemplo: un estudiante que pertenezca al grupo 3, en la primera hora deberá practicar el enfoque manual a los laboratorios terminados en B y para la segunda hora deberá aplicar el enfoque automático para los laboratorios terminados en A.

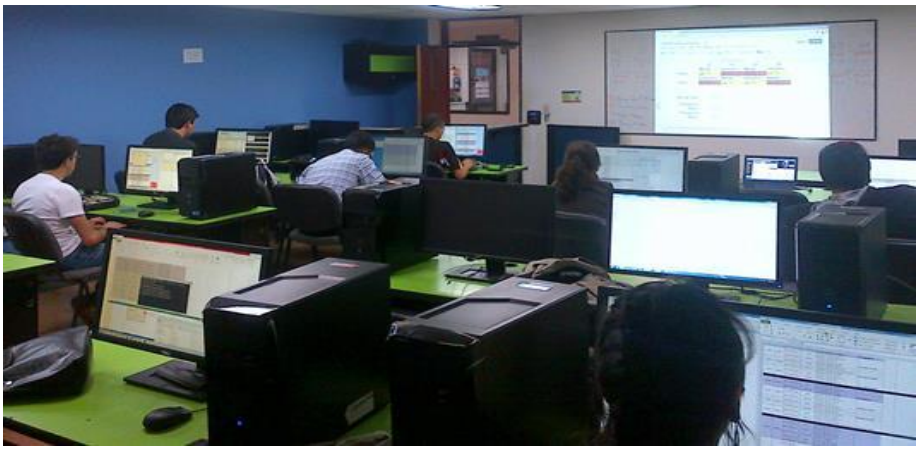

Figura 25 - *Participantes de la segunda sesión*

	Grupo 1	Grupo 2	Grupo 3	Grupo 4
1ª Hora	***Manual***	**MBT**	***Manual***	**MBT**
	Lab. Terminados en A	Lab. Terminados en B	Lab. Terminados en B	Lab. Terminados en A
2ª Hora	**MBT**	***Manual***	**MBT**	***Manual***
	Lab. Terminados en B	Lab. Terminados en A	Lab. Terminados en A	Lab. Terminados en B

Tabla 21 - *Tipos de grupos segunda sesión*

Dentro de la guía entregada a los estudiantes se encuentra cómo practicar los enfoques a los laboratorios designados.

Para realizar el enfoque MBT, el estudiante cuenta con el modelo y las pruebas previamente definidas. Con lo cual, solo basta con realizar la transformación de modelo a texto, para obtener la herramienta encargada de probar. Luego, debe cargar los datos, ejecutar el sistema para que realice las pruebas automáticas. Esperar para obtener los resultados, los cuales debe registrarlos dentro de un archivo de reporte final.

Para realizar el enfoque manual, el estudiante debe ejecutar cada uno de los laboratorios a probar luego ingresar los datos entregados, esperar y observar las respuestas de los SUT. Finalmente, anotar los resultados dentro del archivo de reporte final.

5.8. RESULTADOS CUANTITATIVOS

Mediciones directas:

A continuación se presentan los registros de datos tomados durante la aplicación del estudio de caso. Es importante tener en cuenta que las sesiones contaban con una restricción en tiempo de aplicación, de tres horas para la primera sesión y de dos horas para la segunda sesión. En la Tabla 22, se presentan los tiempos tomados, en las actividades ejecutas para la primera sesión.

Actividad	Tiempo en promedio (min)	Participante 1	Participante 2	Participante 3	Participante 4
Modelar el SUT	15	18	11	12	19
Modelar Casos de Prueba	25	28	24	22	26
Validar modelos	0.016	0.016	0.016	0.016	0.016
Concretizar sistema de pruebas	0.016	0.016	0.016	0.016	0.016
Generar casos de pruebas	10	13	7	8	12
Incluir el SD a probar	12	16	8	10	14
Ejecutar casos de pruebas	24	26	22	24	24

Tabla 22 - *Tiempos de las actividades MBT en la primera sesión*

Para la segunda sesión se los tiempos de las actividades explicadas en la guía para el enfoque MBT y de las pruebas manuales, presentadas en la Tabla 23 y Tabla 24 respectivamente.

Actividad	Tiempo en promedio (min)	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Generar casos de pruebas	22	20	24	22	24	24	19	18	25	21	23
Incluir el SD a probar	2	3	1	2	2	2	2	2	3	1	2

Ejecutar casos de pruebas	3.3	4	3	3	4	3	3	3	3	4	3
Realizar el reporte	10	10	9	10	12	10	9	9	12	10	8

Tabla 23 - *Tiempo en las actividades MBT en la segunda sesión.*

Actividad	Tiempo en promedio (min)	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Iniciar la aplicación	15	14	17	14	16	14	18	13	14	15	15
Ingresar los datos de prueba	20	24	21	17	19	22	17	17	22	23	19
Realizar el reporte	10	10	9	10	12	10	9	9	12	10	8

Tabla 24 - *Tiempo en las actividades Manuales en la primera sesión.*

En las Tabla 25 y Tabla 26, se presentan el cálculo de los diferentes indicadores presentados en la sesión 4.6 Indicadores y Mediciones, por cada participante de la segunda sesión. A continuación se describirán las columnas que componen las tablas:

- **Participante:** en esta columna se describe el número de participante de la sesión.
- **Casos de pruebas exitosas:** número de casos de pruebas bien formulados o bien ejecutados y que detectaron los fallos dentro de la aplicación.
- **Casos de pruebas incorrectos:** número de casos de prueba mal formulados o mal generados y que no detectaron los fallos dentro de la aplicación.
- **Casos de pruebas ejecutadas:** número de casos de pruebas ejecutados por enfoque, para la aplicación.
- **Casos de pruebas definidas:** número de casos de pruebas propuestos para verificar los requerimientos en la guía.
- **Eficacia:** indicador de eficacia por participante, se haya por medio de división de la columna Casos de pruebas exitosas sobre Casos de pruebas ejecutadas.
- **Cobertura de las pruebas definidas:** indicador de cobertura por participante, se haya por medio de división de las columnas Pruebas ejecutadas sobre Pruebas definidas.
- **Eficiencia:** indicador de eficiencia por participante, se haya por medio de división de las columnas Pruebas ejecutadas sobre el tiempo.

Participante	Casos de pruebas exitosas	Casos de prueba Incorrectos	Casos de pruebas ejecutadas	Pruebas definidas	Eficacia	Cobertura de las pruebas definidas	Eficiencia (cp/min)
Participante 1	7	3	10	20	0,700	0,500	0,208
Participante 2	12	0	12	20	1,000	0,600	0,255
Participante 3	15	3	18	20	0,833	0,900	0,439
Participante 4	9	2	11	20	0,818	0,550	0,234
Participante 5	8	6	14	20	0,571	0,700	0,304
Participante 6	7	0	7	20	1,000	0,350	0,159
Participante 7	10	3	13	20	0,769	0,650	0,333
Participante 8	11	4	15	20	0,733	0,750	0,313
Participante 9	13	1	14	20	0,929	0,700	0,292
Participante 10	12	0	12	20	1,000	0,600	0,286
Promedio	**10,4**	**2,2**	**12,6**	**20**	**0,835**	**0,630**	**0,282**

Tabla 25 - *Indicadores por participantes enfoque Manual*

Participante	Casos de pruebas exitosa	Casos de prueba Incorrectos	Casos de pruebas ejecutadas	Pruebas definidas	Eficacia	Cobertura de las pruebas definidas	Eficiencia (cp/min)
Participante 1	19	1	20	20	0,950	1	0,541
Participante 2	20	0	20	20	1,000	1	0,541
Participante 3	17	3	20	20	0,850	1	0,541
Participante 4	20	0	20	20	1,000	1	0,476
Participante 5	19	1	20	20	0,950	1	0,513
Participante 6	18	2	20	20	0,900	1	0,606
Participante 7	20	0	20	20	1,000	1	0,625
Participante 8	18	2	20	20	0,900	1	0,465
Participante 9	19	1	20	20	0,950	1	0,556
Participante 10	18	2	20	20	0,900	1	0,556
Promedio	**18,8**	**1,2**	**20**	**20**	**0,940**	**1**	**0,542**

Tabla 26 - *Indicadores por participantes enfoque MBT*

En la Tabla 27, se realiza la comparación los indicadores en promedio por enfoque de la segunda sesión.

Enfoque	Casos de pruebas exitosa	Casos de prueba incorrectos	Casos de pruebas ejecutadas	Pruebas definidas	Eficacia	Cobertura de las pruebas definidas	Eficiencia (cp/min)
Enfoque Manual	10,4	2,2	12,6	20	0,835	0,63	0,282
Enfoque MBT	18,8	1,2	20	20	0,940	1	0,542

Tabla 27 - *Comparación de indicadores entre los enfoques manual y MBT de la segunda sesión*

A continuación se presenta la lista del cálculo de indicadores calculados a partir de los valores promediados en las Tabla 25 y Tabla *26*:

Eficacia

Para el cálculo de este indicador, se utilizan los valores promediados, obtenidos de la segunda sesión de ejecución, ver Tabla 33:

- **Eficacia en MBT:** el valor obtenido es del 94 por ciento de eficacia.

$$Ef = \frac{Ce}{Cx} = \frac{18{,}8}{20} = 0.94 * 100 = 94$$

- **Eficacia en pruebas manuales:** el valor obtenido de eficacia fue del 83,5 por ciento.

$$Ef = \frac{Ce}{Cx} = \frac{10{,}4}{12{,}6} = 0.835 * 100 = 83{,}5$$

Cobertura de las pruebas definidas

Al igual que para el indicador de eficacia, fueron utilizados los datos de la segunda sesión.

- **Cobertura de las pruebas definidas en MBT:** el valor obtenido fue del 100 por ciento.

$$Cx = \frac{Px}{Pd} = \frac{20}{20} = 1 * 100 = 100$$

- **Cobertura de las pruebas definidas en el enfoque manual:** el valor obtenido fue del 63 por ciento.

$$Cx = \frac{Px}{Pd} = \frac{12{,}6}{20} = 0{,}63 * 100 = 63$$

Eficiencia

Para los resultados de este indicador se debe tener en cuenta que sus valores son dependientes de la complejidad de las pruebas a ejecutar. Debido a esto, se midió este indicador en dos conjuntos de casos de pruebas diferentes, los cuales se tomaron a partir de las dos sesiones ejecutadas.

***Primera sesión*:** para los valores de eficiencia en las pruebas manuales, se utilizaron los datos del trabajo de López [82], donde se realiza una medición del tiempo necesario para la ejecución de 207 pruebas, para ver el detalle de los valores utilizados ver Anexo J.

- **Eficiencia MBT:** aquí se obtuvo un valor de 2,37 pruebas ejecutadas por minuto

$$E = \frac{Px}{t} = \frac{207}{86{,}032} = 2{,}37$$

- **Eficiencia pruebas manuales:** se obtuvo un valor de 1,83 pruebas ejecutadas por minuto.

$$E = \frac{Px}{t} = \frac{207}{111} = 1{,}83$$

***Segunda sesión*:** aquí, el conjunto de pruebas a ejecutar fue de 20, sin embargo la cantidad de pasos requeridos en las pruebas fue mayor, por lo cual el valor del indicador es menor respecto a los resultados de la primera sesión.

- **Eficiencia MBT:** se obtuvo un valor de 0,54 pruebas ejecutadas por minuto.

$$E = \frac{Px}{t} = \frac{20}{45} = 0{,}54$$

- **Eficiencia pruebas manuales:** se obtuvo un valor promedio de 0,28 pruebas por minuto.

$$E = \frac{Px}{t} = \frac{12{,}6}{45} = 0{,}28$$

Facilidad de uso

A partir de la encuesta diligenciada por los participantes de la primera sesión, se analizaron los datos recolectados, con el fin obtener un valor cuantitativo de la facilidad de uso del proceso de MBT, descrito en el presente trabajo. A continuación en la Tabla 28, se presentan el indicador de facilidad de uso por participante:

	Participante			
	Participante 1	**Participante 2**	**Participante 3**	**Participante 4**
Nivel de facilidad de uso	0,84	0,82	0,84	0,9

Tabla 28 – *Indicador de facilidad de uso primera sesión*

Teniendo en cuenta, los resultados obtenidos de los participantes en la primera sesión se puede observar, un sobresaliente nivel de satisfacción; en la utilización del enfoque MBT para las pruebas a un SD, por medio de una herramienta CASE.

Resultados de la encuesta a nivel general

De acuerdo a los resultados presentados en la Tabla 29, se puede observar que los participantes de la primera sesión, consideran que es *Excelente* de manera unánime que el perfil UML de PRUDIMA junto con UML, permite describir de manera adecuado las pruebas a SD de forma adecuada. Por otro lado se tiene una percepción *Sobresaliente* en la sencillez de la tarea de modelar las pruebas para SD, en la facilidad para el usuario en realizar las transformaciones de modelo a código y de la forma ágil de probar múltiples implementaciones de un mismo sistema.

Id	Pregunta	Respuesta				
		Participante 1	**Participante 2**	**Participante 3**	**Participante 4**	**Promedio por pregunta**
1	¿Los pasos del proceso de MBT de PRUDIMA están bien definidos?	5	3	4	5	**4,25**
2	¿La tarea de modelar las pruebas para el sistema distribuido es sencilla?	4	5	5	4	**4,5**
3	¿El proceso de validación del modelo se realiza de manera práctica?	4	3	4	4	**3,75**
4	¿El perfil UML de PRUDIMA junto a UML, permiten describir las pruebas a sistemas distribuidos de forma adecuada?	5	5	5	5	**5**
5	¿Es fácil para el usuario realizar la tarea de transformación de modelos a código fuente?	4	4	5	5	**4,5**
6	¿Es práctica la manera de ejecutar las pruebas contra el sistema a probar?	4	4	4	4	**4**
7	¿Los reportes entregados de los veredictos de las pruebas son entendibles?	4	4	3	4	**3,75**
8	¿El proceso de probar múltiples implementaciones de un mismo sistema se realiza de forma ágil?	4	5	4	5	**4,5**

Id	Pregunta	Respuesta				
		Participante 1	Participante 2	Participante 3	Participante 4	Promedio por pregunta
9	¿Los formularios de la interfaz gráfica de la herramienta PRUDIMA son intuitivos?	4	4	3	5	**4**
10	¿En general considera que el prototipo construido en este trabajo es usable?	4	4	5	4	**4,25**
	Promedio por Participante	**4,2**	**4,1**	**4,2**	**4,5**	**4,25**

Tabla 29 - *Resultados de la encuesta aplicada en la primera sesión*

Donde los valores permitidos para evaluar la herramienta son: 5 = Excelente, 4 = Sobresaliente, 3 =Aceptable, 2 = Deficiente, 1 = Insuficiente.

5.9. ANÁLISIS DE LOS RESULTADOS

A continuación se presentan las gráficas de los resultados obtenidos a partir de la aplicación del estudio de caso y su respectivo análisis de comparación por cada indicador entre los enfoques de pruebas MBT y manuales. manuales.

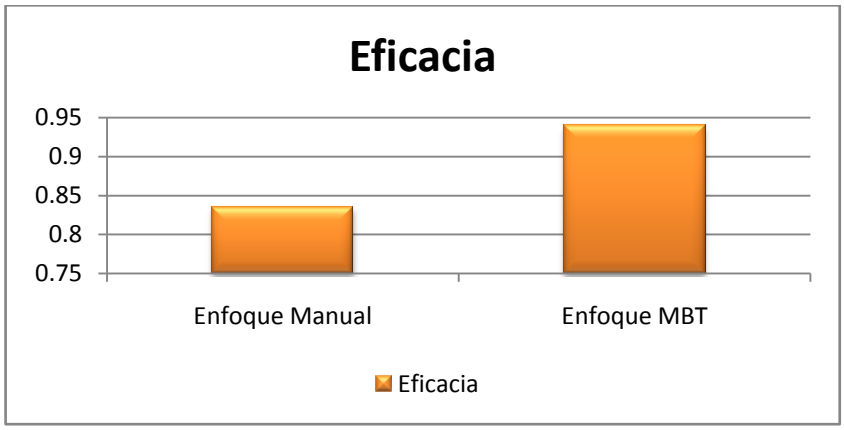

Figura 26 - *Comparación de la eficacia entre los enfoques de pruebas manual y MBT*

Como se aprecia en la Figura 26, se ve como el enfoque MBT es más eficiente que el manual, dado que el 94 por ciento de los casos de pruebas ejecutados en MBT fueron exitosos, contrastándolo con el 84 por ciento del enfoque manual. Sin embargo, se puede apreciar cómo MBT tampoco cumple el cien por ciento de eficacia, ya que existe una dependencia del factor humano, que en este trabajo se ve reflejado en la parte de modelamiento de las pruebas, carga de los datos e inclusión del SUT a probar.

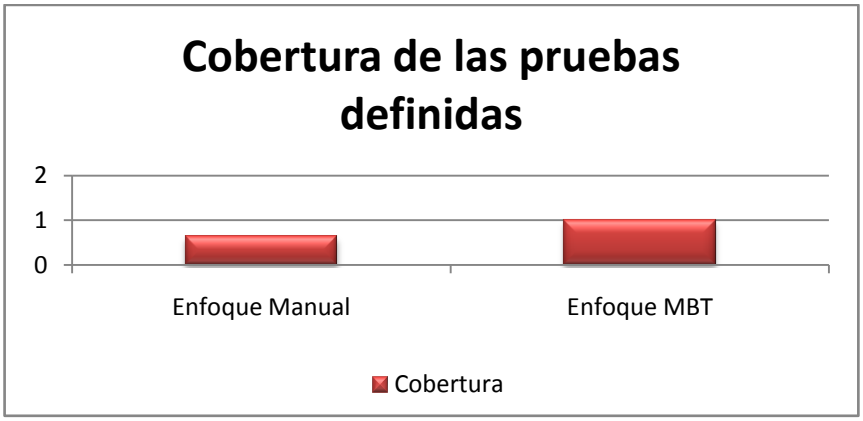

Figura 27 - *Comparación de la cobertura de las pruebas definidas entre los enfoques Manual y MBT*

La Figura 27, permite observar como MBT cumple con el cien por ciento de la cobertura de las pruebas definidas, es decir se ejecutaron todas las pruebas que se definieron inicialmente, a partir de los requerimientos, dentro de un periodo de tiempo limitado. Por otra parte, el enfoque manual alcanzó un sesenta por ciento de cobertura de las pruebas definidas dentro del mismo periodo de tiempo.

Sin embargo, si el tiempo para realizar la fase de pruebas, hubiese sido mayor, es posible que ambos enfoques alcanzaran el mismo nivel de cobertura las pruebas definidas, pero con MBT se realizaría en menor tiempo.

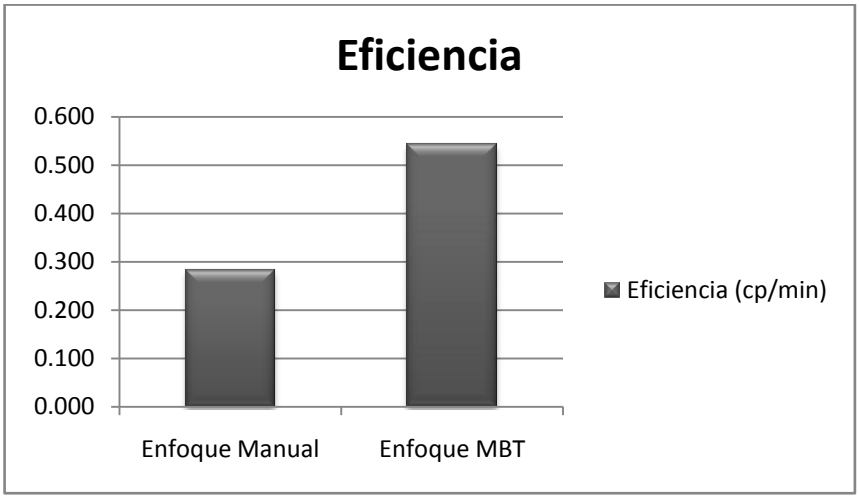

Figura 28 - *Comparación de eficiencia entre los enfoques MBT y Manual*

Este indicador nos permite saber la cantidad de pruebas realizadas en un intervalo de tiempo definido, de la fase de pruebas en ambos enfoques. En la Figura 28, se muestra una mayor eficiencia en MBT con un valor del 0,57 pruebas por minuto en comparación con los 0,29 pruebas por minuto del enfoque manual.

Dado lo anterior, se responde la pregunta de investigación del estudio de caso, concluyendo en base a los resultados, que hay una mejora en la calidad de las pruebas a los SD al usar el enfoque MBT para el proceso de pruebas a SD, con la respectiva reducción del recurso tiempo.

5.9.1. OBSERVACIONES

Por último se presentan algunas observaciones dadas por los autores del presente trabajo que fueron dándose a medida que se realizaba el estudio de caso.

En la Figura 29, se presenta la comparación realizada en el momento de la ejecución de los enfoques manuales y MBT. En esta se presenta como la ejecución de las pruebas manuales es más rápido si se realiza con un número menor a *y1* de pruebas, pero si la pruebas tiene un número mayor, el enfoque MBT es más rápido en comparación con el enfoque manual. Se observa también que existe un tiempo inicial *t1* sin aplicar pruebas en el enfoque MBT, el cual representa el tiempo dedicado al modelamiento de las pruebas en el enfoque MBT, una vez terminado el modelamiento se ejecuta la aplicación y se obtiene una ejecución de pruebas automáticas a una velocidad mayor a la manual.

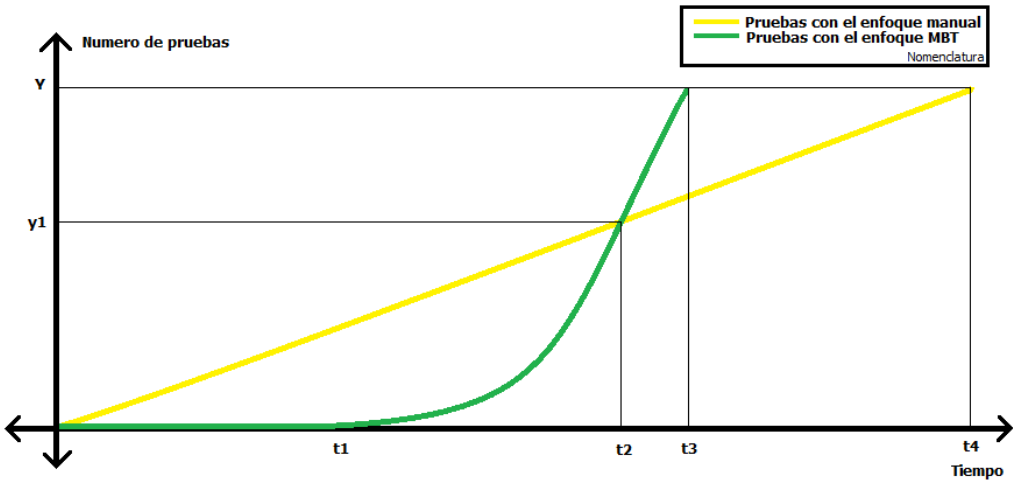

Figura 29 *- Observación del enfoque MBT vs Manual durante el estudio de caso*

La Figura 29, permite observar que la mejora en aplicación de las pruebas con un enfoque basado en modelos frente a las pruebas manuales, depende del número

de casos de pruebas que se deban ejecutar. Si el número de pruebas manuales a realizar conlleva un tiempo menor al modelamiento de las pruebas basadas en modelos es recomendable tomar el enfoque manual. En cambio si el número de pruebas es alto, como en la mayoría de los casos del software industrial, se podría obtener mayores beneficios con el enfoque MBT. Sin embargo, el estudio de caso no permitió encontrar una fórmula o el número de casos de prueba que permita saber con certeza el enfoque a utilizar, la selección del proceso se deja a cargo de la experiencia del *tester*.

Capítulo 6

6. ANÁLISIS DE CUMPLIMIENTO DE OBJETIVOS

En este capítulo se presenta un modelo de indicadores [88] para evaluar el cumplimiento de los objetivos planteados en el anteproyecto, a través de la correlación de los resultados obtenidos en este trabajo y dichos objetivos. A continuación se exponen los lineamientos de conformación e interpretación de los indicadores propuestos.

6.1. LINEAMIENTOS DE CONFORMACIÓN E INTERPRETACIÓN DE LOS INDICADORES

Con el fin de expresar los resultados finales de cada uno de los objetivos se describen a continuación los tipos de indicadores utilizados en la evaluación de los resultados y la forma correcta de interpretarlos.

Los indicadores de desempeño que se evalúan, básicamente adoptan la forma de un cociente, en el cual, el denominador es un valor numérico que ayuda a efectuar la comparación con el logro obtenido así:

$$\text{Indicador} = \left(\frac{Numerador}{Denominador}\right) * FactorEscala$$

De esta forma se definen los siguientes modelos de indicadores que se deben personalizar y aplicar a los actores, productos, funciones, etc, dependiendo del contexto del objetivo evaluado:

Indicador de Cobertura de las pruebas definidas (IC). Permite determinar la cantidad de elementos cubiertos por un producto o estrategia.

$$\text{Cobertura} = \left(\frac{\text{Número de nodos beneficiados con el servicio}}{\text{Número de nodos que se esperaba servir}}\right) * 100$$

Indicador de Eficacia (IE*)*. Analiza el cumplimiento de los requisitos definidos.

$$\text{Eficacia} = \left(\frac{\text{Recursos Ejercidos}}{\text{Recursos Asignados}}\right) * 100$$

Indicador de Eficiencia (IF). Identifica la relación existente entre las metas alcanzadas, el tiempo y los recursos consumidos con respecto a un estándar. Mide el buen uso de los recursos.

$$\text{Eficiencia} = \left(\frac{\textit{Metas alcanzadas}}{\textit{Recursos Consumidos}}\right) * 100$$

Indicador de Calidad (IQ). Está orientado a medir la satisfacción de los beneficiarios.

$$\text{Eficiencia} = \textit{Calificación entre (1:Mala (0\%), 2:Regular (50\%), 3:Buena(75\%), 4:Excelente(100\%))}$$

A continuación, se presenta la evaluación realizada al implementar el anterior modelo, lo cual medir el nivel de cumplimiento de cada uno de los objetivos. Sin embargo, solo se utilizaron los indicadores de eficiencia IF y calidad IQ, ya que los otros no son aplicables a este trabajo de investigación.

6.2. DESCRIPCIÓN Y ALCANCE DEL CUMPLIMIENTO DE LOS OBJETIVOS

Se construyó una tabla por cada objetivo específico de la propuesta de este proyecto, llamado así: "*Prudima-Sd, Pruebas Dirigidas Por Modelos Para Sistemas Distribuidos: Caso De Estudio Curso De Sistemas Distribuidos Universidad Del Cauca*", donde se muestran los productos esperados derivados de cada objetivo, los resultados obtenidos, los indicadores que evalúan el objetivo, los medios de verificación de los resultados y las observaciones que permiten aclarar los resultados en cada objetivo.

No. Objetivo	1
Descripción del objetivo	**Obtener un conjunto de metamodelos que definan la sintaxis y semántica abstracta tanto para el modelado de sistemas distribuidos como para el modelado de los casos de pruebas de dichos sistemas.**
Productos esperados	1. Metamodelo para SD. 2. Metamodelo para pruebas a SD.
Resultados obtenidos	1. Especificación del metamodelo de SD, donde se describe la sintaxis y semántica de cada elemento perteneciente a este. 2. Especificación del metamodelo de PSD, donde se describe la sintaxis y semántica de cada elemento perteneciente a este. 3. Especificación de un perfil UML que integra los anteriores metamodelos.
Indicadores (Escala * 100)	**Eficacia** $IE1 = \frac{NoProductosObtenidos}{NoProductosAObtener} = \frac{3}{2} * 100 = 150\%$ **Calidad** *IQ1 = ¿Los metamodelos creados permiten expresar adecuadamente las pruebas a SD?*

	R = Los metamodelos abstraen la complejidad de diseñar las pruebas para SD a través de la utilización del lenguaje de modelado UML y el perfil UML de PRUDIMA utilizando diagramas de clase y secuencia. . $IQ1 = 4 = 100\%$ **Total Cumplimiento del Objetivo (promedio eficacia)** $Objetivo\ 1 = \frac{100}{1} = 100\%$
Medios de verificación	En las secciones 2.1.2 y 2.1.3 se encuentran la definición de los metamodelos de SD y PSD respectivamente. Además, en la sección 2.1.4 se encuentra la definición del perfil UML que los agrupa.
Estrategias, problemas y/o observaciones	Se construyeron dos metamodelos, uno para modelar SD y otro para el modelamiento de sus pruebas, para utilizarlos dentro de un enfoque MBT cuyo soporte se presenta en los dos primeros capítulos. En el primero se tienen las bases conceptuales tanto de metamodelos como del enfoque MBT, referenciando investigaciones orientadas en propuestas similares. En el segundo se presenta el desarrollo de estos metamodelos utilizando una guía de construcción de metamodelos basados en el enfoque Entidad-Relación. Adicionalmente, durante la revisión de la literatura, se encontraron trabajos donde se habla de perfiles UML y de su integración con UML. Por esto, se profundizó en el tema, dando como resultado el diseño y la construcción de un perfil UML que integra los metamodelos previamente construidos, el cual, permitió a su vez la integración con el lenguaje de modelamiento UML, para utilizar las entidades de los metamodelos de manera práctica. Un inconveniente que se presentó, fue tratar inicialmente de abstraer en el metamodelo, la complejidad inherente de los SD. Lo que conllevo a invertir una gran cantidad de tiempo en la actividad de análisis y diseño. Finalmente se superó este inconveniente, definiendo únicamente las entidades necesarias para modelar los SD y sus pruebas pero basados en el paradigma de comunicación cliente-servidor, bajo la tecnología RMI.

Tabla 30 - *Cumplimiento del primer objetivo específico*

No. Objetivo	2
Descripción del objetivo	**Definir un proceso práctico para las pruebas de sistemas distribuidos basándose en el uso de modelos construidos a partir de los metamodelos previamente obtenidos.**
Productos esperados	1. Documento de definición del proceso práctico para las pruebas de SD.
Resultados obtenidos	1. Documento donde se definió el proceso práctico para las pruebas de SD, incluyendo una diagrama de flujo del mismo.
Indicadores (Escala * 100)	**Eficacia** $IE1 = \frac{NoProductosObtenidos}{NoProductosAObtener} = \frac{1}{1} * 100 = 100\%$ **Calidad**

	IQ1 = ¿El proceso permite probar un SD con un enfoque MBT? *R* = Los pasos definidos en el proceso permiten abstraer la complejidad de la realización de las pruebas a SD, a través de la definición de un enfoque MBT práctico, el cual se soporta sobre una herramienta CASE desarrollada en este trabajo. . $IQ1 = 4 = 100\%$ **Total Cumplimiento del Objetivo (promedio eficacia)** $Objetivo\ 2 = \frac{100}{1} = 100\%$
Medios de verificación	En la sección 2.3 del presente trabajo, se encuentra la definición detallada del proceso para pruebas a SD, junto a un diagrama que resume el flujo del mismo.
Estrategias, problemas y/o observaciones	Se ha construido un proceso que permite implementar un enfoque MBT para las pruebas a SD, descrito en las secciones de los dos primeros capítulos. El primero contiene las bases conceptuales para la creación y el segundo presenta la definición del proceso. Se encuentran muchas maneras de aplicar el enfoque MBT, en los cuales se definen distintos pasos utilizando distintas técnicas. Por lo cual, fue trabajo extenso decidir cada uno de los pasos que mejor se adaptaba al contexto del presente trabajo.

Tabla 31 - *Cumplimiento del segundo objetivo específico*

No. Objetivo	3
Descripción del objetivo	**Determinar la validez del proceso de pruebas dirigidos por modelos, a través de un prototipo de herramienta CASE y de modelos de casos de prueba previamente diseñados, en un entorno de ejecución soportado por agentes software, donde la herramienta realizará análisis y reportes de las pruebas, para medir el número de fallos presentados en el software en construcción.**
Productos esperados	1. Prototipo de herramienta CASE que soporte el enfoque MBT definido en este trabajo. 2. Código fuente del prototipo de la herramienta CASE. 3. Documentación del prototipo de la herramienta CASE. 4. Documento donde se evidencie el diseño, la ejecución y el análisis de resultados de un estudio de caso.
Resultados obtenidos	1. Se realizó el proceso de desarrollo de un prototipo de la herramienta CASE asignándole el nombre de PRUDIMA, la cual, soporta el proceso de MBT definido en el presente trabajo. 2. Documento de evidencia de diseño, ejecución y análisis de resultados del estudio de caso. 3. Código fuente en lenguaje *Java* y *Acceleo*. 4. Documentos donde se muestra tanto el proceso de instalación del prototipo de la herramienta como su uso.
Indicadores (Escala * 100)	**Eficacia** $IE1 = \frac{NoProductosObtenidos}{NoProductosAObtener} = \frac{4}{4} * 100 = 100\%$

	Calidad *IQ1 = ¿La herramienta CASE PRUDIMA desarrollada brinda un soporte adecuado al proceso de MBT definido en este trabajo?* *R* = La herramienta permite el modelado de pruebas, la transformación de los modelos a código fuente ejecutable (Java), la ejecución de las pruebas y análisis de los resultados de las mismas. Todo esto integrado dentro del IDE de EMF. $$IQ1 = 4 = 100\%$$ *IQ2 = ¿El estudio de caso presenta adecuadamente los resultados obtenidos de la ejecución del estudio de caso?* *R* = Se muestran los resultados de forma individual, tabulada y gráfica. Además de presentar un análisis detallado del significado de los mismos. $$IQ2 = 4 = 100\%$$ **Total Cumplimiento del Objetivo (promedio eficacia)** $Objetivo\ 3 = \frac{100}{1} = 100\%$
Medios de verificación	1. En capítulo 3 del presente trabajo, se describe todo el proceso de desarrollo de software que se utilizó para construir la herramienta CASE llamada PRUDIMA. 2. El código fuente del prototipo y la documentación del prototipo de la herramienta CASE PRUDIMA se anexa digitalmente. 3. En el capítulo 4 del presente trabajo, se encuentra el diseño, ejecución y análisis de resultados del estudio de caso.
Estrategias, problemas y/o observaciones	Se hizo un proceso de desarrollo de software utilizando la metodología UP Ágil. Esta etapa consumió bastante tiempo del proyecto, ya que fue necesario aprender nuevas tecnologías, como *JADE*, *Acceleo* y *JUnit*. El estudio de caso fue llevado a cabo en las instalaciones de la Universidad del Cauca. En la primera sesión ejecutada dentro del estudio de caso, se pudieron obtener todos los datos necesarios para analizar los indicadores definidos en el mismo. Sin embargo, los datos obtenidos para ser contrastados no estaban completos, por lo tanto sólo se pudo comparar el indicador de la eficiencia. La segunda sesión del estudio de caso se vio afectada por la limitante del tiempo disponible de los recursos computacionales necesarios para la aplicación del mismo. Por lo que fue necesario realizar pruebas donde la carga de datos fuese de complejidad.

Tabla 32 - *Cumplimiento del tercer objetivo específico*

Capítulo 7

7. CONCLUSIONES, LIMITACIONES Y TRABAJOS FUTUROS

A continuación se describen las conclusiones del presente trabajo de investigación, así como las limitaciones y trabajos futuros propuestos.

7.1. CONCLUSIONES

En esta sección se detallan las conclusiones a las que llegaron los autores del trabajo de investigación.

La industria del software que utiliza MDE, genera modelos que pueden ser reutilizados en diferentes fases del proceso de ingeniería de software. Es en la fase de implementación, donde comúnmente son usadas un conjunto transformaciones a los modelos, generando automáticamente código fuente. En este trabajo se reutilizaron modelos de UML para la fase de pruebas, siguiendo el enfoque MBT. Donde se adaptaron por medio de un perfil UML, para realizar pruebas a sistemas SD en construcción, optimizando el uso de recursos de tiempo y esfuerzo. Evidenciando así, la potencialidad de los enfoques dirigidos por modelos.

Durante el desarrollo de este trabajo de investigación, se aprendieron nuevos conceptos, paradigmas y herramientas asociados a la fase de pruebas, ampliando nuestra visión de las tendencias del mercado del software. Durante la carrera de ingeniería de sistemas, no se profundiza en dicha área, sin embargo, los fundamentos aprendidos de modelado de software, fueron fundamentales para entender estos nuevos conceptos.

Los metamodelos diseñados para SD y sus pruebas, permiten modelar adecuadamente pruebas a SD, basados en el paradigma de comunicación cliente/servidor. No obstante, es a través de su integración dentro de un perfil UML, que son utilizados de manera práctica, por medio de la extensión semántica de algunos elementos de los modelos que componen el lenguaje UML, reutilizando así, algunos de los artefactos generados por metodologías de desarrollo de software, como son los modelos estáticos y dinámicos del sistema a construir.

En este trabajo se definió un proceso que permite la generación de una plataforma de pruebas para SD soportada por agentes software, cuyo código es generado por

un conjunto de transformaciones M2T, utilizando como insumo modelos de UML, más específicamente, utilizando los modelos definidos en diagramas de clases y de secuencia. A los elementos que componen estos diagramas, se les aplican los estereotipos del perfil UML de PRUDIMA, permitiendo extender su semántica hacia modelos de pruebas. De estos modelos se pueden generar manualmente casos de pruebas, que finalmente son ejecutados contra el SUT, obteniendo un análisis y reporte de los mismos, evaluando así, la conformidad de los requerimientos definidos para el SUT.

Los resultados obtenidos en el estudio de caso, realizado en el entorno académico de la Universidad del Cauca, muestran que la utilización del enfoque MBT, a través de una herramienta que lo soporte, permite aumentar los niveles de calidad del proceso de pruebas a SD, además de reducir los tiempos requeridos por parte de los ingenieros encargados de evaluar las prácticas de laboratorios de SD.

En MBT se presentaron errores en las pruebas, principalmente por el factor humano, sin embargo el porcentaje es menor en comparación con el enfoque de pruebas manuales, debido a que en MBT se mitigan por medio de validaciones automáticas en dichas actividades manuales y el uso de código pre-construido para la ejecución de la pruebas.

Se evidencia que para seguir un enfoque MBT, el cual utiliza modelos soportados en gráficos, se debe utilizar una herramienta que tenga un buen nivel de usabilidad, para que el tiempo empleado en las fases de diseño de los modelos no contrarreste los beneficios obtenidos por la automatización de los demás pasos dentro del proceso de pruebas.

Se muestra cómo MBT podría ser portado a diferentes tipos de dominios software. En este trabajo de investigación se utilizó, para realizar pruebas en el ámbito de los SD enfocados en el paradigma cliente servidor.

7.2. LIMITACIONES

En el presente trabajo, se utilizaron los diagramas de clases para representar la parte estática del sistema y los diagramas de secuencia para representar los comportamientos de los componentes del sistema. Sin embargo, no se utilizaron todos los elementos disponibles por este tipo de diagrama, como son: los condicionales, los bucles, las instanciaciones a demanda por otras instancias de las clases del sistema, etc. Solo se utilizaron las líneas de vida, los mensajes asincrónicos, los parámetros de entrada a través de la firma de los métodos.

El oráculo implementado en este trabajo, se genera de forma semi-automatizada a través del prototipo PRUDIMA, pero se necesita la experiencia del ingeniero de pruebas para diseñar y generar los casos de pruebas adecuados, que permitan verificar de manera óptima los requerimientos del SUT. Según la literatura [40] [41], la generación de un oráculo automatizado, supone gran cantidad de tiempo y esfuerzo.

Otro inconveniente presentado en el presente trabajo, es que los metamodelos, solo permiten modelar mensajes secuenciales y ordenados, es decir no se permite envió de mensajes en paralelo, limitando los tipos de interacción a modelar. Además, en los diagramas de clases solo está permitido tener una única interfaz remota que define los servicios remotos del sistema. Por lo tanto, no se pueden evaluar prácticas que utilicen la técnica de *callback*, disminuyendo los tipos de sistemas distribuidos a probar, restringiéndolos a la tecnología RMI.

El prototipo PRUDIMA solo tuvo en cuenta evaluar los requerimientos asociados al componente servidor, ya que los clientes generalmente son basados en interfaces gráficas. Probar estas interfaces graficas tiene un alto grado de complejidad que esta por fuera del alcance de este trabajo. En la literatura se pueden encontrar trabajos asociados a este tema como se ve en [89], [90]. Otra limitante del prototipo, es que únicamente puede desplegar los SUT en redes de área local (LAN), debido a la restricción de la plataforma JADE, que no permite extenderse a redes de área amplia (WAN).

La facilidad de uso en la construcción de los diagramas fue limitada por el editor Papyrus, ya que este tiene algunos problemas en su entorno. Por ejemplo: en el momento de borrar un elemento del modelo gráfico, no actualizaba el modelo UML, fuente principal para la generación de las pruebas.

7.3. LECCIONES APRENDIDAS Y PROBLEMAS RESUELTOS

Se describen en esta sección los problemas presentados durante esta investigación y cómo pudieron resolverse:

- La creación de metamodelos es una tarea compleja que depende de la experiencia de las personas en el dominio del problema. En este trabajo se contó con la ayuda de expertos en el dominio de los SD y de las pruebas para éstos, además se contó con una guía que define una serie de pasos para la construcción de los metamodelos, por lo que esta dificultad pudo ser superada satisfactoriamente.

- Seleccionar el lenguaje adecuado para las transformaciones M2T fue una tarea costosa para el proyecto, porque se tuvieron que probar varios lenguajes de transformaciones como *ATLAS*, *Acceleo,* entre otros*,* para

luego compararlos y escoger el que mejor se adaptara a las necesidades de este proyecto. Al final se escogió *Acceleo* por las razones descritas en la Sección 2.2.1.

- Escoger un modelador UML que soportara el uso de perfiles UML creados con metamodelos *Ecore*, fue también una tarea costosa, porque se requirió probar las características de cada modelador, resultando algunas pruebas con errores que hicieron que se descartara el modelador bajo prueba porque no permitía realizar alguna acción vital para el proceso de modelado de pruebas.

- Surgieron inconvenientes en la integración entre el *framework JUnit* y *JADE*, ya que ambas poseen una plataforma de trabajo basada en hilos, que trabajan de manera asincrónica y se debió realizar un procedimiento para secuenciar su ejecución. De esta manera se pudieron obtener los resultados deseados en el prototipo de PRUDIMA. Esto se pudo realizar gracias a la documentación existente de cada una de estas tecnologías y por la experiencia en desarrollo de software de los autores del presente trabajo.

- Tener un sistema para el manejo de referencias colaborativo, permite ser más eficientes a la hora de escribir la monografía, disminuyendo el tiempo de aplicación de formatos y ubicaciones de las referencias en el documento.

7.4. TRABAJOS FUTUROS

A continuación se consideran algunos ítems que pueden ser realizados en trabajos futuros a corto o mediano plazo.

Debido a la limitante de los metamodelos, se podría adaptar estos metamodelos y el prototipo PRUDIMA para que soporte otros tipos de sistemas software, como los sistemas *pear to pear*, otros soportados por el paradigma cliente-servidor como CORBA, DCOM, RPC, entre otros.

Podría realizarse una comparación entre trabajos similares que realicen pruebas con enfoque MBT para entornos de SD, con el fin de comparar las características y definir algunas mediciones, para establecer la mejor elección al momento de realizar el proceso de pruebas a los SUT. Incluso, se podría comparar el enfoque MBT contra un enfoque automatizado por medio de técnicas de scripting en el mismo contexto de los SD y siguiendo los mismos lineamientos del estudio de caso planteado en este trabajo de tesis.

Sería interesante, crear un oráculo automatizado, donde se puedan establecer reglas de generación de casos de pruebas en diferentes entornos de software utilizando alguna de las técnicas como particiones de equivalencia, datos aleatorios, entre otras, del proceso de MBT enmarcado en este trabajo.

Construir un modelador de pruebas a medida, enfocándose en la facilidad de uso del mismo, ya que *Papyrus,* que fue el modelador utilizado para soportar el proceso, es genérico para UML y por tanto se presentan muchos elementos para su uso. Esto hace que se aumente la carga cognitiva para el ingeniero de pruebas y el tiempo de aprendizaje para la aplicación del proceso. Posteriormente evaluar si la utilización de un modelador propio, hace que sea eficiente el proceso de diseño de pruebas.

REFERENCIAS

[1] K. S. Nizam Uddin Ahameda R. Badlishah Ahmadb, Matiur Rahmanc, Asraf, "A framework for the development of measurement and quality assurance in software-based medical rehabilitation systems," *Procedia Engineering - International Symposium on Robotics and Intelligent Sensors 2012*, vol. 41, pp. 53 – 60, 2012.

[2] K. S. G. Kannabiran, "Determinants of software quality in offshore development - An empirical study of an Indian vendor," *Information and Software Technology*, pp. 1199–1208, 2011.

[3] M. a. Rothenberger, Y.-C. Kao, and L. N. Van Wassenhove, "Total quality in software development: An empirical study of quality drivers and benefits in Indian software projects," *Information & Management*, vol. 47, no. 7–8, pp. 372–379, Dec. 2010.

[4] C. Serrano, "Un Modelo Integral para un Profesional en Ingeniería," *Universidad del Cauca*, 2003.

[5] R. S. Pressman, *Ingeniería del Software. Un enfoque práctico*. McGraw-Hill, 2003.

[6] J. S. Osmundson, J. B. Michael, M. J. Machniak, and M. A. Grossman, "Quality management metrics for software development," vol. 2033, pp. 1–14, 2002.

[7] INTECO, "Calidad de un producto software," *Laboratorio Nacional de Calidad del Software de INTECO*, vol. 1, p. 10, 2009.

[8] D. C. Schmidt, "Model-Driven Engineering." IEEE Computer Society, Vanderbilt University, 2006.

[9] M. R. John Hutchinson Jon Whittle, "Model-Driven Engineering Practices in Industry." School of Computing and Communications, Lancaster University, UK, 2011.

[10] I. García-Magariño, R. Fuentes-Fernández, and J. J. Gómez-Sanz, "Guideline for the definition of EMF metamodels using an Entity-Relationship approach," *Inf. Softw. Technol.*, vol. 51, no. 8, pp. 1217–1230, 2009.

[11] S. J. Mellor, *MDA distilled: principles of model-driven architecture*. Addison-Wesley Professional, 2004.

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

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

[14] M. Hebach, “MDA with QVT,” *Presentación Borland Together*, 2006.

[15] “The VIATRA2 Model Transformation Framework.” [Online]. Available: http://www.eclipse.org/viatra2/.

[16] E. D. López, M. González, M. López, and E. L. Iduñate, “Proceso de Desarrollo de Software Mediante Herramientas MDA,” *Revista Iberoamericana de Sistemas, Cibernética e Informática, 3*, pp. 6–10, 2007.

[17] L. C. Briand, Y. Labiche, and S. He, “Automating regression test selection based on UML designs,” *Information and Software Technology*, vol. 51, no. 1, pp. 16–30, Jan. 2009.

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

[19] M. Wooldridge, *An Introduction to MultiAgent Systems*. Department of Computer Science, University of Liverpool, UK: John Wiley & Sons, Ltd, 2002.

[20] J. D. George Coulouris Tim Kindberg and Gordon Blair, “Distributed Systems: Concepts and Design.” Addison-Wesley Longman Publishing Co, Boston, MA, USA, 2012.

[21] J. D. Ahmad Saifan, “Model-Based Testing of Distributed Systems,” p. 57, 2008.

[22] R. V Binder, “Model-based Testing User Survey: Results and Analysis.” System Verification Associates Technical Report, p. 6, 2011.

[23] M. Utting, A. Pretschner, and B. Legeard, “A Taxonomy of Model-Based Testing.” School of Computing and Mathematical Sciences, University of Waikato, Hamilton, New Zealand, 2006.

[24] G. H. T. Arilo Claudio Dias-Neto, “Model-based testing approaches selection for software projects.” Federal University of Rio de Janeiro, Systems Engineering and Computer Science Program, Rio de Janeiro, RJ, Brazil, 2009.

[25] J. Tretmans, F. Prester, P. Helle, and W. Schamai, “Model-Based Testing 2010: Short Abstracts,” *Electronic Notes in Theoretical Computer Science*, vol. 264, no. 3, pp. 85–99, Dec. 2010.

[26] M. Utting and B. Legeard, “A Taxonomy of Model-Based Testing,” April, 2006.

[27] R. S. Arilo C. Dias Neto Marlon Vieira, Guilherme H. Travassos, “A Survey on Model-based Testing Approaches: A Systematic Review.”

[28] J. Boberg, “Early fault detection with model-based testing,” *Proceedings of the 7th ACM SIGPLAN Workshop*, 2008.

[29] S. Weißleder, “Test Models and Coverage Criteria for Automatic Model-Based Test Generation with UML State Machines,” Humboldt-Universität zu Berlin, 2010.

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

[31] S. Wieczorek, V. Kozyura, and A. Roth, “Applying model checking to generate model-based integration tests from choreography models,” *Testing Software*, 2009.

[32] D. Buchs, L. Lucio, and A. Chen, “Model checking techniques for test generation from business process models,” *Reliable Software Technologies–Ada-Europe*, 2009.

[33] M. Veanes, C. Campbell, W. Grieskamp, W. Schulte, N. Tillmann, and L. Nachmanson, “Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer,” *Microsoft Research*, 2007.

[34] E. Alves and P. Machado, “Uma abordagem integrada para desenvolvimento e teste dirigido por modelos,” *Anais do 2nd*, 2008.

[35] J. Jürjens, “Model-based security testing using umlsec: A case study,” *Electronic Notes in Theoretical Computer Science*, 2008.

[36] P. Mohagheghi, V. Dehlen, and T. Neple, “Definitions and approaches to model quality in model-based software development – A review of literature,” *Information and Software Technology*, vol. 51, no. 12, pp. 1646–1669, 2009.

[37] W. Prenninger and A. Pretschner, “Abstractions for Model-Based Testing,” *Electronic Notes in Theoretical Computer Science*, pp. 59–71, 2005.

[38] M. Fowler, *UML distilled: a brief guide to the standard object modeling language*. 2004.

[39] E. Diaz, R. Blanco, and J. Tuya, “Comparación de tecnicas metaheuristicas para la generación automatica de casos de prueba que obtengan una cobertura software.” CEUR Workshop, 2002.

[40] B. P. Lamancha, M. Polo, D. Caivano, M. Piattini, and G. Visaggio, “Automated generation of test oracles using a model-driven approach,” *Information and Software Technology*, vol. 55, no. 2, pp. 301–319, Feb. 2013.

[41] S. R. Shahamiri, W. M. N. W. Kadir, and S. Z. Mohd-Hashim, “A Comparative Study on Automated Software Test Oracle Methods,” *Fourth International Conference on Software Engineering Advances*, 2009, pp. 140–145.

[42] A. Bauer, M. Leucker, and C. Schallhart, “Model-based runtime analysis of distributed reactive systems,” in *Australian Software Engineering Conference (ASWEC’06)*, 2006, p. 10 pp.–252.

[43] J. Yan, J. Wang, and H. Chen, “UML Based Statistical Testing Acceleration of Distributed Safety-Critical Software,” in in *Parallel and Distributed Processing and Applications SE - 52*, vol. 3358, J. Cao, L. Yang, M. Guo, and F. Lau, Eds. Springer Berlin Heidelberg, 2005, pp. 433–445.

[44] L. Gönczy, R. Heckel, and D. Varro, “Model-Based Testing of Service Infrastructure Components,” *Testing of Software and Communicating*, vol. 4581 LNCS, no. i, pp. 155–170, 2007.

[45] A. T. Endo and A. Simao, *Model-Based Testing of Service-Oriented Applications via State Models*, no. i. IEEE, 2011, pp. 432–439.

[46] P. Hudak and D. S. Warren, Eds., “Model-Based Testing of Thin-Client Web Applications and Navigation Input,” vol. 4902, 2008.

[47] N. V Pakulin, A. N. Tugaenko, and V. Z. Shnitman, “Model-based testing of internet e-mail protocols,” *Programming and Computer Software*, vol. 38, no. 5, pp. 268–275, 2012.

[48] M. Lahami, F. Fakhfakh, M. Krichen, and M. Jmaiel, “Towards a TTCN-3 Test System for Runtime,” pp. 71–86, 2012.

[49] A. F. Karr and A. A. Porter, “Distributed performance testing using statistical modeling,” *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, p. 1, Jul. 2005.

[50] Z. Zhang, J. Thangarajah, and L. Padgham, “Model based testing for agent systems,” *Software and Data Technologies*, vol. 22, pp. 399–413, 2008.

[51] B. Aichernig, A. Griesmayer, E. Johnsen, R. Schlatte, and A. Stam, “Conformance Testing of Distributed Concurrent Systems with Executable Designs,” in in *Formal Methods for Components and Objects SE - 4*, vol. 5751, F. Boer, M. Bonsangue, and E. Madelaine, Eds. Springer Berlin Heidelberg, 2009, pp. 61–81.

[52] N. Jardim Nunes, B. Selic, A. Rodrigues da Silva, and A. Toval Alvarez, Eds., “The AGEDIS Tools for Model Based Testing,” *UML Modeling Languages and Applications*, vol. 3297, pp. 277–280, 2005.

[53] R. Casado, J. Tuya, and M. Younas, “Testing the reliability of web services transactions in cooperative applications,” in *Proceedings of the 27th Annual ACM Symposium on Applied Computing - SAC ’12*, 2012, p. 743.

[54] R. F.-F. Iván García-Magariño * Jorge J. Gómez-Sanz, “Guideline for the definition of EMF metamodels using an Entity-Relationship approach,” *Information and Software Technology*, vol. 51, 2009.

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

[56] D. Watkins and D. Thompson, “Adding semantics to interface definition languages,” in *Proceedings 1998 Australian Software Engineering Conference (Cat. No.98EX233)*, 1998, pp. 66–78.

[57] H. Owens II and J. H. Hill, “Generating Valid Interface Definition Language from Succinct Models,” in *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, 2011, pp. 205–212.

[58] P. Baker, Z. R. Dai, J. Grabowski, Ø. Haugen, I. Schieferdecker, and C. Williams, *Model-driven testing. Using the UML Testing Profile*. 2008, p. 188.

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

[60] “An introduction to UML profiles,” *UML OMG*, 2004.

[61] A. P. Ruiz, “Generación de código ADA para aplicaciones embebidas y de tiempo real desde modelos dinámicos UML,” 2012.

[62] M. Vanzetti, “Desarrollo Dirigido por Modelos de Procesos de Negocio Colaborativos: Análisis de herramientas para la transformación de modelos,” *CIDISI, Universidad Tecnológica acional-FRSF, Lavaisse 610, 3000*, 2010.

[63] L. M. Rose, N. Matragkas, D. S. Kolovos, and R. F. Paige, “A feature model for model-to-text transformation languages,” in *2012 4th International Workshop on Modeling in Software Engineering (MISE)*, 2012, pp. 57–63.

[64] D. Streitferdt, F. Kantz, P. Nenninger, T. Ruschival, H. Kaul, T. Bauer, T. Hussain, and R. Eschbach, “Model-Based Testing of Highly Configurable Embedded Systems in the Automation Domain,” *International Journal of Embedded and Real-Time Communication Systems*, vol. 2, no. 2, pp. 22–41, 2011.

[65] A. Bertolino, E. Marchetti, and H. Muccini, “Introducing a Reasonably Complete and Coherent Approach for Model-based Testing,” *Electronic Notes in Theoretical Computer Science*, vol. 116, pp. 85–97, Jan. 2005.

[66] F. Abbors, A. Bäcklund, and D. Truscan, “MATERA - An Integrated Framework for Model-Based Testing,” in *2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, 2010, pp. 321–328.

[67] J. Botella, F. Bouquet, J.-F. Capuron, F. Lebeau, B. Legeard, and F. Schadle, “Model-Based Testing of Cryptographic Components -- Lessons Learned from Experience,” in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, 2013, pp. 192–201.

[68] K. Nylund, E. Östman, D. Truscan, and R. Teittinen, “Towards Rapid Creation of Test Adaptation in On-line Model-Based Testing,” in *2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*, 2011, pp. 174–179.

[69] K. Nylund, E. Östman, D. Truscan, and R. Teittinen, “Towards Rapid Creation of Test Adaptation in On-line Model-Based Testing,” *IEEE 35th Annual Computer Software and Applications Conference Workshops*, 2011, pp. 174–179.

[70] H. Jin, Y. Wang, N.-W. Chen, Z.-J. Gou, and S. Wang, “Artificial Neural Network for Automatic Test Oracles Generation,” *International Conference on Computer Science and Software Engineering*, 2008, vol. 2, pp. 727–730.

[71] I. Rauf, M. Z. Z. Iqbal, and Z. I. Malik, “UML Based Modeling of Web Service Composition - A Survey,” in *2008 Sixth International Conference on Software Engineering Research, Management and Applications*, 2008, pp. 301–307.

[72] M. Cristia, “Introducción al Testing de Software,” pp. 1–8, 2009.

[73] Jianwei Li and Xunan Wang, “Research and practice of agile Unified Process,” *2nd International Conference on Software Technology and Engineering*, 2010, vol. 2, pp. V2–340–V2–343.

[74] S. W. Ambler, “The Agile Unified Process (AUP),” 2006. [Online]. Available: www.ambysoft.com/unifiedprocess/agileUP.html.

[75] A. Bauer, M. Leucker, and C. Schallhart, “Model-based runtime analysis of distributed reactive systems,” in *Australian Software Engineering Conference (ASWEC)*, 2006, p. 10 pp.–252.

[76] R. Brennan, “Toward real-time distributed intelligent control: A survey of research themes and applications,” *Systems, Man, and Cybernetics, Part C*, 2007.

[77] P. Vrba, “Java-based agent platform evaluation,” *Holonic and Multi-Agent Systems for Manufacturing*, 2003.

[78] X. Jinkai and Y. Weihong, “Study on comparison between JAFMAS and JADE,” *Second Pacific-Asia Conference on*, 2010.

[79] R. Coelho and U. Kulesza, “Unit testing in multi-agent systems using mock agents and aspects,” *Scale multi-agent systems*, 2006.

[80] eclipse.org, “Eclipse Public License - v 1.0,” 2004. [Online]. Available: http://www.eclipse.org/legal/epl-v10.html.

[81] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical Software Engineering*, 2009.

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

[83] H. Maimbo, G. Pervan, and W. Perth, “Designing a case study protocol for application in IS research,” *Proceedings of the Ninth Pacific Asia*, 2005.

[84] I. Benbasat, D. Goldstein, and M. Mead, “The case research strategy in studies of information systems,” *MIS quarterly*, 1987.

[85] R. Yin, “Case study research: Design and methods,” 2009.

[86] H. Klein and M. Myers, “A set of principles for conducting and evaluating interpretive field studies in information systems,” *MIS quarterly*, 1999.

[87] P. B. Lakey, “A Measurement Framework for Assessing Model-Based Testing Quality,” in *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, 2010, pp. 19–27.

[88] U. VRI, “Evaluación de informes finales de proyectos de investigación,” *UNIVERSIDAD DEL CAUCA*, 2003.

[89] J. Strecker and A. Memon, “Testing graphical user interfaces,” *Encyclopedia of Information Science*, 2009.

[90] A. Memon, “A comprehensive framework for testing graphical user interfaces,” 2001.