ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNICACIONES

TRABAJO DE TITULACIÓN, PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

TEMA: DISEÑO DE UN PROTOTIPO DE DSL BASADO EN INGENIERIA DE MODELOS PARA EL CONTROL A DISTANCIA DE UN DRON TERRESTRE

AUTOR: VALENCIA MANOSALVAS, CARLOS VICENTE

DIRECTOR: ALULEMA, DARWIN

SANGOLQUÍ

2017

UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERIA EN ELECTRÓNICA Y TELECOMUNICACIONES

CERTIFICACIÓN

Certifico que el trabajo de titulación, "DISEÑO DE UN PROTOTIPO DE DSL BASADO EN INGENIERIA DE MODELOS PARA EL CONTROL A DISTANCIA DE UN DRON TERRESTRE" realizado por el señor VALENCIA MANOSALVAS CARLOS VICENTE, ha sido revisado en su totalidad y analizado por el software anti-plagio, el mismo cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, por lo tanto me permito acreditarlo y autorizar al señor VALENCIA MANOSALVAS CARLOS VICENTE para que lo sustente públicamente.

Sangolquí, 15 de Agosto del 2017

Ing. Darwin Alulema Msc.

DIRECTOR

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERIA EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTORÍA DE RESPONSABILIDAD

Yo, CARLOS VICENTE VALENCIA MANOSALVAS, con cédula de identidad N° 172423504-7, declaro que este trabajo de titulación "DISEÑO DE UN PROTOTIPO DE DSL BASADO EN INGENIERIA DE MODELOS PARA EL CONTROL A DISTANCIA DE UN DRON TERRESTRE" ha sido desarrollado considerando los métodos de investigación existentes, así como también se ha respetado los derechos intelectuales de terceros considerándose en las citas bibliográficas. Consecuentemente declaro que este trabajo es de mi autoría, en virtud de ello me declaro responsable del contenido, veracidad y alcance de la investigación mencionada.

Sangolquí, 15 de Agosto del 2017

Carlos Vicente Valencia Manosalvas

C.I.: 1724235047

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERIA EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTORIZACIÓN

Yo, CARLOS VICENTE VALENCIA MANOSALVAS, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar en la biblioteca Virtual de la institución el presente trabajo de titulación "DISEÑO DE UN PROTOTIPO DE DSL BASADO EN INGENIERIA DE MODELOS PARA EL CONTROL A DISTANCIA DE UN DRON TERRESTRE" cuyo contenido, ideas y criterios son de mi autoría y responsabilidad.

Carlos Vicente Valencia Manosalvas

C.I.: 1724235047

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERIA EN ELECTRÓNICA Y TELECOMUNICACIONES

CLAUSULA DE LICENCIA DE USO DE PUBLICACIÓN DE TESIS

Yo Carlos Vicente Valencia Manosalvas, autor de la tesis "DISEÑO DE UN PROTOTIPO DE DSL BASADO EN INGENIERIA DE MODELOS PARA EL CONTROL A DISTANCIA DE UN DRON TERRESTRE", mediante el presente documento dejo constancia de que la obra es de mi exclusiva autoría y producción, que la elaborado para cumplir con uno de los requisitos previos para la obtención del título de INGENIERO EN ELECTRONICA Y TELECOMUNICACIONES en la Universidad de las Fuerzas Armadas – ESPE.

- Licencio gratuitamente a la Universidad de las Fuerzas Armadas – ESPE, los derechos de reproducción, comunicación pública, distribución y divulgación, durante 48 meses a partir de mi graduación, pudiendo por lo tanto la Universidad, utilizar y usar esta obra para cualquier medio conocido o por conocer, siempre y cuando no se lo haga para obtener beneficio económico. Esta autorización incluye la reproducción total o parcial en los formatos virtual, electrónico, digital, óptico, como usos en red local y en internet.
- Declaro que, en caso de presentarse cualquier reclamación de parte de terceros respecto de los derechos de autor de la obra antes referida, yo asumiré toda la responsabilidad frente a terceros y a la Universidad.
- En esta fecha entrego a la Secretaria General, el ejemplar respectivo y sus anexos en formato impreso y digital o electrónico.

Quito, 15 de agosto del 2017

Carlos Valencia

DEDICATORIA

La vida se encuentra plagada de retos, los cuales he podido superarlos gracias al apoyo incondicional de mis padres, quienes han fomentado en mi la perseverancia y superación, a la vez son mi motivación para seguir adelante y seguir superando los retos que se presenten en un futuro. Sus consejos van a servir como una enseñanza más de vida.

A mi hermana por el apoyo incondicional, comprensión y confianza en los momentos indispensable de mi vida.

Carlos

AGRADECIMIENTO

Agradezco a Dios ser maravilloso que me dio fuerza y fe para terminar lo que parecía muy lejano. A mi familia por apoyarme y estar a mi lado siempre en todo momento de mi vida.

A mis maestros y compañeros quienes me apoyaron en mi permanencia en la universidad, en especial a mi Director Ing. Darwin Alulema, que gracias a su disposición y sus conocimientos han sabido guiarme como profesor y como amigo para la realización y culminación del presente trabajo.

Para todas estas personas gracias de todo corazón.

Carlos

ÌNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En los últimos años ha tomado relevancia la necesidad de mejorar el proceso de desarrollo de software. Si se reduce la complejidad del modelo, delimitándolo a un ámbito particular, se puede representar utilizando la metodología DSM (Domain-Specific Modeling por sus siglas en inglés), para modelado específico de dominio. Y se puede utilizar para su representación el Lenguaje Específico de Dominio, por sus siglas en ingles DSL (Domain Specific Language).

Visto esto la presente investigación muestra la creación de una herramienta de Lenguaje de Dominio Específico (DSL), que apoyada en ingeniería dirigida por modelos (MDE), permite la planificación y creación de rutas definidas para el control de un dron terrestre; esta herramienta es independiente de la plataforma. La creación de la plataforma para convertir del Modelo a texto ¨Código¨ se realizó sobre una plataforma java particular llamada Eclipse. Las pruebas realizadas evidenciaron que al trabajar con MDE se reduce el tiempo y esfuerzo en la creación y despliegue de los módulos modelados sobre Eclipse y que el metamodelo planteado es compatible con los requerimientos de dicho drone.

PALABRAS CLAVES:

- **DOMAIN SPECIFIC LANGUAGE (DSL)**
- **MODEL DRIVEN ENGINEERING (MDE)**
- **DRONE**
- **UML**

ABSTRACT

In the last years, the people need to improve the software development process has become relevant. If the complexity of the model is reduced, delimiting it to a particular domain, it can be represented using the DSM (Domain-Specific Modeling) methodology, for domain-specific modeling. And the Domain Specific Language (DSL) can be used for its representation.

Given this, the present research shows the creation of a Specific Domain Language (DSL) tool, supported by model-driven engineering (MDE), allows the planning and creation of defined routes for the control of a terrestrial drone; this tool is platform independent. The creation of the platform to convert from the Model to text ¨Code¨ carried out on a particular java platform called Eclipse OXIGEN. The tests showed that working with MDE reduces the time and effort in creating and deploying modules modeled on Eclipse and that the proposed metamodel is compatible with the requirements of the drone.

KEY WORDS:

- **DOMAIN SPECIFIC LANGUAGE (DSL)**
- **MODEL DRIVEN ENGINEERING (MDE)**
- **DRONE**
- **UML**

CAPÍTULO I

INTRODUCCION

1.1. Presentación

En el presente documento se desarrolla la implementación de una herramienta DSL ("Domain-Specific Language") basada en ingeniería de modelos para el control de un dron terrestre a larga distancia con la ayuda de Eclipse.

Está estructurado en 6 capítulos en los cuales se va a detallar todo el proceso investigativo y teórico, el primer capítulo consta de una breve introducción y antecedentes importantes de anteriores investigaciones, desarrolladas del tema propuesto, también se fijarán los objetivos tales como el principal y los específicos, así como la respectiva justificación e importancia además del alcance del proyecto.

El segundo capítulo consta de un marco teórico, que es el Estado del Arte de cada uno de los componentes de la presente investigación además de un estudio científico y técnico de los componentes del dron que se va a emplear.

Tercer capítulo se elabora un esquema del diseño de la Herramienta DSL para el control del Drone Terrestre, utilizando ingeniería de modelos para poder desarrollar el Diagrama UML ("*Unified Modeling Language"*) y posteriormente el código a implementarse.

Cuarto capítulo una vez ya desarrollado el diseño y el modelo, se procede a la implementación de la Herramienta DSL en el Drone Terrestre mediante comunicación serial con la PC y con la ayuda de un Arduino UNO.

Quinto Capitulo una vez ya implementada la Herramienta DSL en el drone terrestre se realizan pruebas de aceptación y de funcionamiento de cada componente de la herramienta creada.

Sexto capítulo se analiza la información recolectada, que de acuerdo a los resultados obtenidos se procederá a realizar las respectivas comparaciones para luego plasmar las recomendaciones y conclusiones.

1.2. Antecedentes

En el desarrollo de software e ingeniería de modelos, un lenguaje específico del dominio (Domain-Specific Language, DSL) es un lenguaje de programación o especificación dedicado a resolver un problema en particular, representar un problema específico y proveer una técnica para solucionar una situación particular. El concepto no es nuevo pero se ha vuelto más popular debido al aumento del uso de modelaje específico del dominio.

Crear un lenguaje específico de dominio (con software que lo soporte) vale la pena cuando permite que un tipo particular de problemas o soluciones puedan ser expresadas más claramente que con otros lenguajes existentes. En los DSL, la semántica del lenguaje está muy cercana al dominio del problema para el cual se diseña. (Marjan Mernik, Jan Heering, y Anthony M. Sloane, 2005)

El Desarrollo de Software Dirigido por Modelos (Model-Driven Development, MDD) es una disciplina alternativa para la producción de software, más orientado al Espacio de la Solución que al Espacio del Problema. Es un proceso robusto de producción de software basado en Modelos Conceptuales y dirigido por las Transformaciones correspondientes entre Modelos definidos de forma precisa. (Pons, C., Giandini, R. S., & Pérez, 2010).

Un dron es cualquier aparato que cumple una misión sin tripulación a bordo, puede ser que el piloto lo controle a distancia o que tenga un sistema de vuelo autónomo. Algunos argumentan que un dron es solo el que vuela de forma autónoma, pero lo cierto es que el término engloba ambos tipos. Aunque el aparato sea capaz de volar autónomamente, requiere que una persona planifique/programe la ruta/misión. Cabe mencionar que ya desde los años 1910 existían ejemplos de aviones por control remoto.

Aunque los drones son vehículos aéreos no tripulados (UAV), es decir aeronave que vuela sin tripulación, no hay que olvidar que se tratan de una tipología de robot para el aire, por lo que el continuo avance de la tecnología y el desarrollo en continuo auge de los drones, está provocando que cada vez más se hable de ellos y se diseñen nuevas sugerencias para los robots.

Este puede ser el caso de los drones terrestres, diseñados para la diversión y también para aplicaciones militares como por ejemplo la detección y desactivación de bombas. (Enciso Jiménez, 2016)

Hoy en día, los drones o vehículos aéreos no tripulados están siendo empleados en muchas áreas, por lo que existen diferentes paquetes encargados de planificar y ejecutar planes de vuelo. El más popular actualmente es el “Mission Planner”, donde cada usuario planifica la ruta de vuelo sobra la zona que debe ser fotografiada, para que emprenda el vuelo.

1.3. Justificación e importancia

Actualmente en el Ecuador se registras apenas 2870 personas con conocimientos avanzados de programación según datos del Ministerio de Educación esto da una referencia que existe un bajo porcentaje con relación al total de población que está casi por los 17 millones de personas y además tampoco existen muchas herramientas para la programación de navegación terrestre.

En vista a estos aspectos el presente proyecto de investigación se encuentra enmarcado con el desarrollo de una herramienta fácil para la programación de rutas y control de Drones terrestres para poder satisfacer las necesidades de las personas que no tienen conocimientos avanzados de programación.

Permite así que las personas dispongan de una herramienta fácil de utilizar para dedicarse a la planificación de rutas según su necesidad sin entrar en el entorno de programación.

Además, al incorporar ingeniería de modelos se tiene la ventaja de brindar una solución a un problema en un solo bloque, esto en el diseño del proyecto permite

crear una arquitectura que puede ser utilizada en proyectos futuros de navegación, ya que al ser un diseño modular permite su desarrollo y adaptación a nuevos modelos.

Para finalmente lograr un modelaje específico de dominio para la construcción de módulos en sistemas de control de un drone independiente de la estructura. Para esto, el punto de partida es un metamodelo para la construcción de un lenguaje específico de dominio (DSL) que con ingeniería dirigida por modelos (MDE) y aplicando las debidas transformaciones de modelo a modelo y de modelo a texto se consiga un modelo dependiente de la plataforma que sea funcional, sencillo, rápido y liviano.

1.4. Alcance del proyecto

A la finalización de la investigación se pretende encontrar un criterio científico que permita determinar la eficiencia de la herramienta DSL, mediante pruebas de usabilidad.

1.5. Metodología

Para el desarrollo del proyecto de investigación se pretende utilizar una metodología mixta, ya que posee un enfoque cualitativo y cuantitativo.

Desde el enfoque cualitativo se pretende realizar el desarrollo del estado del arte, a través de una búsqueda bibliográfica, utilizado los diferentes repositorios y bases de datos indexadas, así como las palabras claves DSL, modelos, control de drones, etc. Con el objetivo de comprender el diseño actual de un metamodelo, que representa la arquitectura del sistema para generar el modelo de control del Drone terrestre.

Desde el enfoque cuantitativo, una vez que se ha determinado el Metamodelo para el diseño particular del control de un drone terrestre se desarrolla la herramienta DSL acoplada a la arquitectura de la interfaz gráfica para la programación y control del Drone terrestre. Con las plataformas como por ejemplo Eclipse con la herramienta EMF (Eclipse Modeling Framework) o OBEO designer de Sirius.

Luego de desarrollar la herramienta DSL se procede a la implementación en la parte física adaptada a la plataforma específica del Dron terrestre para proceder a las pruebas de concepto.

También para el presente proyecto se pretende utilizar un Drone terrestre que tenga una conexión vía wifi o bluetooth con algunas características que permitan la conexión con el software desarrollado que puede ser en Eclipse o en Sirius.

1.6. Objetivos

1.6.1 Objetivo General

- Diseñar un prototipo de lenguaje específico de dominio usando ingeniería de modelos para el control a distancia de un dron terrestre.

1.6.2 Objetivos específicos

- Desarrollar el estado del arte de la ingeniería de modelos.
- Caracterización de la plataforma de un Dron terrestre.
- Analizar las herramientas de software para implementar la herramienta DSL.
- Diseñar la Arquitectura del Meta Metamodelo para la implementación.
- Implementar el DSL a partir del Metamodelo para controlar el Drone terrestre.
- Desarrollar pruebas de concepto para la arquitectura propuesta.
- Analizar los resultados.

1.7. Estado del arte

En (Bats, 2015) se muestra el diseño de una aplicación para el control de un Arduino basado en ingeniería de modelos para la plataforma de java, se puede conocer el uso de las librerías y de los plugins que utiliza Eclipse las cuales están

desarrolladas en un lenguaje diferente a otras plataformas como netbenas, su trabajo muestra cómo se puede integrar las diferentes tecnologías que Arduino utiliza en un software para lograr generar los modelos correspondientes a cada uno de ellos y posterior mente a su implementación.

En (Quiroz, Muñoz y Noël, 2012) habla que en el ámbito de la educación, estudios y trabajos recientes muestran que el uso de la robótica es un pilar fundamental el crecimiento de la sociedad, también muestra como acoplar esta tecnología con las Ciencias de la Computación para generar un diseño, construcción y programación de robots. Actualmente la robótica ha dejado de estar al margen de la sociedad y se aplica actualmente en diversos ámbitos. Este trabajo expone la creación de un Lenguaje de Dominio Específico, y su respectivo Entorno de Desarrollo, con lo que se pretende un mayor acercamiento de los estudiantes a los robots LEGO Mindstorms que pueden tener las mismas características que un Drone.

En (Ayala Ramírez, Ortega García, & Parada Salado, 2016) presenta una investigación a fondo a cerca del manejo y construcción de un vehículo no tripulado manejado por RF o drone para poder operarlo en zonas de difícil acceso de personas, para sistemas de seguridad y de comercio de productos pequeños entre comerciantes.

En (Quintero & Anaya, 2008) habla acerca de que el papel de los modelos es fundamental en el desarrollo de software para potenciar el reúso de los diferentes elementos del software y facilitar la labor de los diferentes roles que participan del proceso. En este artículo también se puede ver que la Arquitectura Dirigida por Modelos (MDA) propone un proceso de desarrollo basado en la realización y transformación de modelos. Los principios en los que se fundamenta MDA son la abstracción, la automatización y la estandarización. El proceso central de MDA es la transformación de modelos que parten del espacio del problema (CIM) hasta modelos específicos de la plataforma (PSM), pasando por modelos que describen una solución independientemente de la computación (PIM). Para explicar el papel de los modelos en el proceso de desarrollo de software este artículo explora los principales conceptos presentados en la propuesta de MDA.

En (Montenegro Marin, Gaona Garcia, & Cueva, 2011) habla acerca de la realización de un modelado específico de dominio para la construcción de módulos en sistemas de gestión del aprendizaje (LMS) independientes de la plataforma. Para

esto, toma como punto de partida es un metamodelo para la construcción de un lenguaje específico de dominio (DSL) que con ingeniería dirigida por modelos (MDE) y aplicando las debidas transformaciones se consiga de un modelo independiente de la plataforma, el despliegue de este modelo se realizara en varias plataformas LMSs con este artículo se puede tener un base para la generación de la interfaz de la plataforma que se va a desarollar.

En cuanto a investigaciones propias de plataformas que permitan la generación de código automático para el control de drones no se a encontrado ninguna por el momento por lo que la presente investigación es un gran aporte en este campo que cada día crece más y más.

Los estudios analizados previamente brindan un panorama donde la tecnología es una herramienta fundamental inmiscuida en los ámbitos del día a día, el uso de la ingeniería de modelos se puede emplear tanto para la generación de plataformas para controlar elementos físicos como en este caso Drones como para generar programas para interactuar con diversos usuarios, mediante se presenta un panorama infinito de aplicaciones que se podrían realizar basados en esta tecnología.

CAPÍTULO II

INVESTIGACION DE INGENIERIA DE MODELOS.

2.1 El desarrollo de software basado en modelos

La ingeniería de software establece que el problema de construir software debe ser encarado de la misma forma en que los ingenieros construyen otros sistemas complejos, como puentes, edificios, barcos y aviones. La idea básica consiste en observar el sistema de software a construir como un producto complejo y a su proceso de construcción como un trabajo ingenieril. Es decir, un proceso planificado basado en metodologías formales apoyadas por el uso de herramientas. (Pons, 2010).

Hacia finales de los años 70, Tom Demarco en su libro "Structured Analysis and System Specification". (Demarco 79). Introdujo el concepto de desarrollo de software basado en modelos o MBD (por sus siglas en inglés Model Based Development). DeMarco destacó que la construcción de un sistema de software debe ser precedida por la construcción de un modelo, tal como se realiza en otros sistemas ingenieriles (ver figura 1). El modelo del sistema es una conceptualización del dominio del problema y de su solución. El modelo se focaliza sobre el mundo real: identificando, clasificando y abstrayendo los elementos que constituyen el problema y organizándolos en una estructura formal. (Pons, 2010).

La abstracción es una de las principales técnicas con la que la mente humana se enfrenta a la complejidad. Ocultando lo que es irrelevante, un sistema complejo se puede reducir a algo comprensible y manejable. Cuando se trata de software, es sumamente útil abstraerse de los detalles tecnológicos de implementación y tratar con los conceptos del dominio de la forma más directa posible. De esta forma, el modelo de un sistema provee un medio de comunicación y negociación entre usuarios, analistas y desarrolladores que oculta o minimiza los aspectos relacionados con la tecnología de implementación. (Pons, 2010).

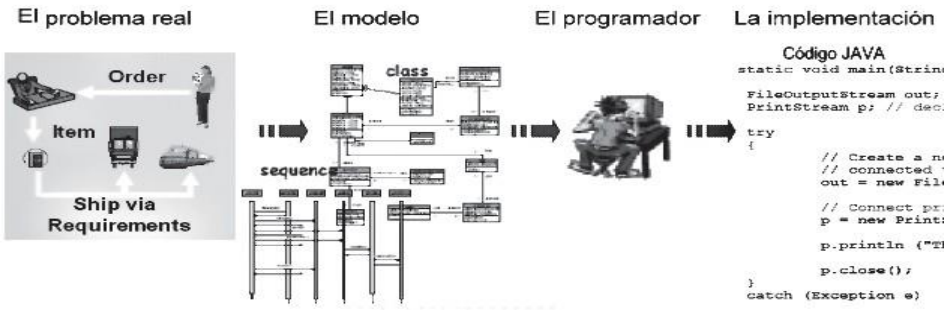


Figura 1: Desarrollo de software basado en modelos

Fuente: (Pons, 2010)

Si bien el desarrollo de software basado en modelos ha representado un paso importante en la ingeniería de software, la crisis sigue existiendo. El proceso sigue careciendo de características que permitan asegurar la calidad y corrección del producto final. El proceso de desarrollo de software, como se utiliza hoy en día, es conducido por el diseño de bajo nivel y por el código. Un proceso típico es el que se muestra en la figura 2.2, el cual incluye seis fases: (Pons, 2010).

- La conceptualización y la determinación de los requisitos del usuario
- Análisis y descripción funcional
- Diseño
- Codificación
- Testeo
- Emplazamiento

Figura 2: Ciclo de vida del desarrollo de software basado en modelos

Fuente: (Pons, 2010)

En las primeras fases se construyen distintos modelos tales como los modelos de los requisitos (generalmente escritos en lenguaje natural), los modelos de análisis y los modelos de diseño (frecuentemente expresa- dos mediante diagramas). Estas fases pueden realizarse de una manera iterativa-incremental o en forma de cascada. En la práctica cotidiana, los modelos quedan rápidamente desactualizados debido al atajo que suelen tomar los desarrolladores en el ciclo de vida de este proceso. (Pons, 2010).

2.2 El desarrollo de software dirigido por modelos (MDD)

El Desarrollo de Software Dirigido por Modelos MDD (Model Driven software Development) se ha convertido en un nuevo paradigma de desarrollo software. MDD promete mejorar el proceso de construcción de software basándose en un proceso guiado por modelos y soportado por potentes herramientas. El adjetivo "dirigido" (*driven)* en MDD, a diferencia de "basado" (*Based)*, enfatiza que este paradigma asigna a los modelos un rol central y activo: son al menos tan importantes como el código fuente. Los modelos se van generando desde los más abstractos a los más concretos a través de pasos de transformación y/o refinamientos, hasta llegar al código aplicando una última transformación. La transformación entre modelos constituye el motor de MDD. Los puntos claves de la iniciativa MDD fueron identificados en (Booch 04b) de la siguiente forma: (Pons, 2010).

1- El uso de un mayor nivel de **abstracción** en la especificación tanto del problema a resolver como de la solución correspondiente, en relación con los métodos tradicionales de desarrollo de software.

2- El aumento de confianza en la **automatización** asistida por computadora para soportar el análisis, el diseño y la ejecución.

3- El uso de **estándares** industriales como medio para facilitar las comunicaciones, la interacción entre diferentes aplicaciones y productos, y la especialización tecnológica.

La figura 1 muestra la parte del proceso de desarrollo de software en donde la intervención humana es reemplazada por herramientas automáticas. Los modelos pasan de ser entidades contemplativas (es decir, artefactos que son interpretadas por

los diseñadores y programadores) para convertirse en entidades productivas a partir de las cuales se deriva la implementación en forma automática. (Pons, 2010).

2.2.1 El ciclo de vida dirigido por modelo

El ciclo de vida de desarrollo de software usando MDD se muestra en la figura 3 Este ciclo de vida no luce muy distinto del ciclo de vida tradicional. Se identifican las mismas fases. Una de las mayores diferencias está en el tipo de los artefactos que se crean durante el proceso de desarrollo. Los artefactos son modelos formales, es decir, modelos que pueden ser comprendidos por una computadora. En figura 4 las líneas punteadas señalan las actividades automatizadas en este proceso. (Pons, 2010).

MDD identifica distintos tipos de modelos:

- Modelos con alto nivel de abstracción independientes de cualquier metodología computacional, llamados CIMs (Computational Independent Model),
- Modelos independientes de cualquier tecnología de implementación llamados PIMs (Platform Independent Model),
- Modelos que especifican el sistema en términos de construcciones de implementación disponibles en alguna tecnología específica, conocidos como PSMs (Platform Specific Model),
- y finalmente Modelos que representan el código fuente en sí mismo, identificados como IMs (Implementation Model).

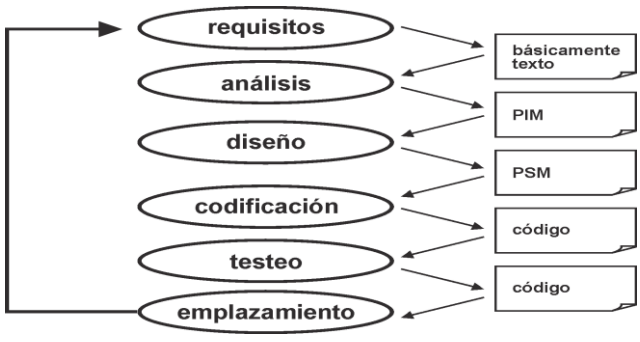


Figura 3: Ciclo de vida del desarrollo de software dirigido por modelos

Fuente: (Pons, 2010)

En el proceso de desarrollo de software tradicional, las transformaciones de modelo a modelo, o de modelo a código son hechas mayormente con intervención humana. Muchas herramientas pueden generar código a partir de modelos, pero generalmente no van más allá de la generación de algún esqueleto de código que luego se debe completar manualmente. (Pons, 2010).

En contraste, las transformaciones MDD son siempre ejecutadas por herramientas, como se muestra en la figura 4. Muchas herramientas pueden transformar un PSM a código; no hay nada nuevo en eso. Dado que un PSM es un modelo muy cercano al código, esta transformación no es demasiado compleja. Lo novedoso que propone MDD es que las transformaciones entre modelos (por ejemplo, de un PIM a PSMs) sean automatizadas. (Pons, 2010).

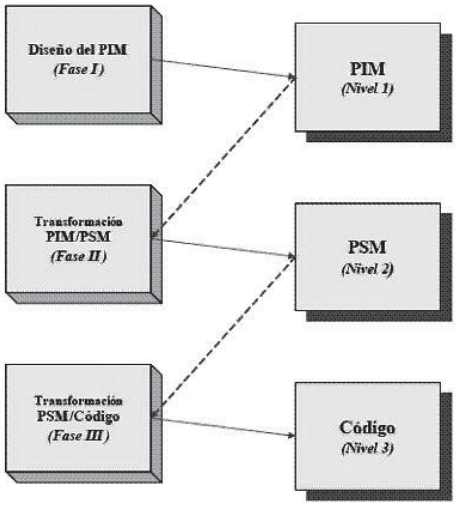


Figura 4: Los tres pasos principales en el proceso de desarrollo MDD.

Fuente: (Pons, 2010)

2.2.2 Orígenes de MDD

Si bien MDD define un nuevo paradigma para el desarrollo de software, sus principios fundamentales no constituyen realmente nuevas ideas, sino que son reformulaciones y

asociaciones de ideas anteriores. MDD es la evolución natural de la ingeniería de software basada en modelos enriquecida mediante el agregado de transformaciones automáticas entre modelos. Por su parte, la técnica de transformaciones sucesivas tampoco es algo novedoso. Se puede remitirnir al proceso de abstracción y refinamiento presentado por Edsger W. Dijkstra en su libro "A Discipline of Programming". (Dijkstra 76).

Donde se define que refinamiento es el proceso de desarrollar un diseño o implementación más detallado a partir de una especificación abstracta a través de una secuencia de pasos matemáticamente justificados que mantienen la corrección con respecto a la especificación original. Es decir, de acuerdo con esta definición, un refinamiento es una transformación semánticamente correcta que captura la relación esencial entre la especificación (es decir, el modelo abstracto) y la implementación (es decir, el código). (Pons, 2010).

Consecuentemente se puede señalar que el proceso MDD mantiene una fuerte coincidencia, en sus orígenes e ideas centrales, con el seminal concepto de abstracción y refinamientos sucesivos, el cual ha sido estudiado extensamente en varias notaciones formales tales como Z (DB01) y B (Lano 96) y diversos cálculos de refinamientos [BW98]. En general estas técnicas de refinamiento se limitan a transformar un modelo formal en otro modelo formal escrito en el mismo lenguaje (es decir, se modifica el nivel de abstracción del modelo, pero no su lenguaje), mientras que MDD es más amplio pues ofrece la posibilidad de transformar modelos escritos en distintos lenguajes (por ejemplo, se puede transformar un modelo escrito en UML en otro modelo escrito en notación Entidad-Relación). (Pons, 2010).

La plataforma tecnológica subyacente o la arquitectura de implementación. Dichos cambios se realizan modificando la transformación del PIM al PSM. La nueva transformación es reaplicada sobre los modelos originales para producir artefactos de implementación actualizados. Esta flexibilidad permite probar diferentes ideas antes de tomar una decisión final. Y además permite que una mala decisión pueda fácilmente ser enmendada. (Pons, 2010).

Adaptación a los cambios en los requisitos: poder adaptarse a los cambios es un requerimiento clave para los negocios, y los sistemas informáticos deben ser capaces de soportarlos. Cuando se usa un proceso MDD, agregar o modificar una

funcionalidad de negocios es una tarea bastante sencilla, ya que el trabajo de automatización ya está hecho. Cuando se agrega una nueva función, sólo se necesita desarrollar el modelo específico para esa nueva función. El resto de la información necesaria para generar los artefactos de implementación ya ha sido capturada en las transformaciones y puede ser reusada. (Pons, 2010).

Consistencia: la aplicación manual de las prácticas de codificación y diseño es una tarea propensa a errores. A través de la automatización MDD favorece la generación consistente de los artefactos.

Re-uso: en MDD se invierte en el desarrollo de modelos y transformaciones. Esta inversión se va amortizando a medida que los modelos y las transformaciones son reusados. Por otra parte, el re-uso de artefactos ya probados incrementa la confianza en el desarrollo de nuevas funcionalidades y reduce los riesgos ya que los temas técnicos han sido previamente resueltos. (Pons, 2010).

Mejoras en la comunicación con los usuarios: los modelos omiten detalles de implementación que no son relevantes para entender el comportamiento lógico del sistema. Por ello, los modelos están más cerca del dominio del problema, reduciendo la brecha semántica entre los conceptos que son entendidos por los usuarios y el lenguaje en el cual se expresa la solución. Esta mejora en la comunicación influye favorablemente en la producción de software mejor alineado con los objetivos de sus usuarios. (Pons, 2010).

Mejoras en la comunicación entre los desarrolladores: los modelos facilitan el entendimiento del sistema por parte de los distintos desarrolladores. Esto da origen a discusiones más productivas y permite mejorar los diseños. Además, el hecho de que los modelos son parte del sistema y no sólo documentación, hace que los modelos siempre permanezcan actualizados y confiables. (Pons, 2010).

Captura de la experiencia: las organizaciones y los proyectos frecuentemente dependen de expertos clave quienes toman las decisiones res- pecto al sistema. Al capturar su experiencia en los modelos y en las transformaciones, otros miembros del equipo pueden aprovecharla sin requerir su presencia. Además, este conocimiento se mantiene aún cuan- do los expertos se alejen de la organización. (Pons, 2010).

Los modelos son productos de larga duración: en MDD los modelos son productos importantes que capturan lo que el sistema informático de la organización

hace. Los modelos de alto nivel son resistentes a los cambios a nivel plataforma y sólo sufren cambios cuando lo hacen los requisitos del negocio. (Pons, 2010).

Posibilidad de demorar las decisiones tecnológicas: cuando se aplica MDD, las primeras etapas del desarrollo se focalizan en las actividades de modelado. Esto significa que es posible demorar la elección de una plataforma tecnológica específica o una versión de producto hasta más adelante cuando se disponga de información que permita realizar una elección más adecuada. (Pons, 2010).

2.2.3 Propuestas concretas para MDD

Dos de las propuestas concretas más conocidas y utilizadas en el ámbito de MDD son, por un lado, MDA desarrollada por el OMG y por otro lado el modelado específico del domino (DSM acrónimo inglés de Domain Specific Modeling) acompañado por los lenguajes específicos del dominio (DSLs acrónimo inglés de Domain Specific Language). Ambas iniciativas guardan naturalmente una fuerte conexión con los conceptos básicos de MDD. Específicamente, MDA tiende a enfocarse en lenguajes de modelado basados en estándares del OMG, mientras que DSM utiliza otras notaciones no estandarizadas para definir sus modelos. (Pons, 2010).

2.2.4 Beneficios de MDD

El desarrollo de software dirigido por modelos permite mejorar las prácticas corrientes de desarrollo de software. Las ventajas de MDD son las siguientes:

Incremento en la productividad: MDD reduce los costos de desarrollo de software mediante la generación automática del código y otros artefactos a partir de los modelos, lo cual incrementa la productividad de los desarrolladores. Se nota que se deberia sumar el costo de desarrollar (o comprar) transformaciones, pero es esperable que este costo se amortice mediante el re-uso de dichas transformaciones.

Adaptación a los cambios tecnológicos: el progreso de la tecnología hace que los componentes de software se vuelvan obsoletos rápidamente. MDD ayuda a solucionar este problema a través de una arquitectura fácil de mantener donde los cambios se

implementan rápida y consistentemente, habilitando una migración eficiente de los componentes hacia las nuevas tecnologías. Los modelos de alto nivel están libres de detalles de la implementación, lo cual facilita la adaptación a los cambios que pueda sufrir. (Pons, 2010).

2.3 Arquitectura dirigida por modelos (MDA)

MDA, acrónimo inglés de *Model Driven Architecture* (Arquitectura Dirigida por Modelos), es una de las iniciativas más conocida y extendida dentro del ámbito de MDD. (Pons, 2010).

MDA es un concepto promovido por el OMG *(Object management Group)* a partir de 2000, con el objetivo de afrontar los desafíos de integración de las aplicaciones y los continuos cambios tecnológicos. MDA es una arquitectura que proporciona un conjunto de guías para estructurar especificaciones expresadas como modelos, siguiendo el proceso MDD. (Pons, 2010).

En MDA, la funcionalidad del sistema es definida en primer lugar como un modelo independiente de la plataforma (Platform-Independent Model o PIM) a través de un lenguaje específico para el dominio del que se trate. En este punto aparece además un tipo de modelo existente en MDA, no mencionado por MDD: el modelo de definición de la plataforma (Platform Definition Model o PDM). Entonces, dado un PDM correspondiente a CORBA, .NET, Web, etc., el modelo PIM puede traducirse a uno o más modelos específicos de la plataforma (Platform-Specific Models o PSMs) para la implementación correspondiente, usando diferentes lenguajes específicos del dominio, o lenguajes de propósito general como Java, C#, Python, etc. El proceso MDA completo se encuentra detallado en un documento que actualiza y mantiene el OMG denominado la Guía MDA. (Pons, 2010).

MDA no se limita al desarrollo de sistemas de software, sino que también se adapta para el desarrollo de otros tipos de sistemas. Por ejemplo, MDA puede ser aplicado en el desarrollo de sistemas que incluyen a personas como participantes junto con el software y el soporte físico. (Pons, 2010).

Asimismo, MDA está bien adaptado para el modelado del negocio y empresas. Es claro que el alcance del uso de los conceptos básicos de MDA es más amplio. MDA

está relacionada con múltiples estándares, tales como el Unified Modeling Language (UML), el Meta-Object Facility (MOF), XML Metadata Interchange (XMI), Enterprise Distributed Object Computing (EDOC), el Software Process Engineering Metamodel (SPEM) y el Common Warehouse Metamodel (CWM).

El OMG mantiene la marca registrada sobre MDA, así como sobre varios términos similares incluyendo Model Driven Development (MDD), Model Driven Application Development, Model Based Application Developtment, Model Based Programming y otros similares. El acrónimo principal que aún no ha sido registrado por el OMG hasta el presente es MDE, que significa Model Driven software Engineering. A consecuencia de esto, el acrónimo MDE es usado actualmente por la comunidad investigadora internacional cuando se refieren a ideas relacionadas con la ingeniería de modelos sin centrarse exclusivamente en los estándares del OMG. (Pons, 2010).

Cumpliendo con las directivas del OMG, las dos principales motivaciones de MDA son la interoperabilidad (independencia de los fabricantes a través de estandarizaciones) y la portabilidad (independencia de la plataforma) de los sistemas de software; las mismas motivaciones que llevaron al desarrollo de CORBA. (Pons, 2010).

Además, el OMG postula como objetivo de MDA separar el diseño del sistema tanto de la arquitectura como de las tecnologías de construcción, facilitando así que el diseño y la arquitectura puedan ser alterados independientemente. El diseño alberga los requisitos funcionales (casos de uso, por ejemplo) mientras que la arquitectura proporciona la infraestructura a través de la cual se hacen efectivos los requisitos no funcionales como la escalabilidad, fiabilidad o rendimiento. MDA asegura que el modelo independiente de la plataforma (PIM), el cual representa un diseño conceptual que plasma los requisitos funcionales, sobreviva a los cambios que se produzcan en las tecnologías de fabricación y en las arquitecturas de software. Por supuesto, la noción de transformación de modelos en MDA es central. (Pons, 2010).

La traducción entre modelos se realiza normalmente utilizando herramientas automatizadas, es decir herramientas de transformación de modelos que soportan MDA. Algunas de ellas permiten al usuario definir sus propias transformaciones.

Una iniciativa del OMG es la definición de un lenguaje de transformaciones estándar denominado QVT [QVT] que aún no ha sido masivamente adoptado por las herramientas, por lo que la mayoría de ellas aún definen su propio lenguaje de transformación, y sólo algunos de éstos se basan en QVT. Actualmente existe un amplio conjunto de herramientas que brindan soporte para MDA. (Pons, 2010).

2.4 Modelado específico del dominio (DSM Y DSLS)

Por su parte, la iniciativa DSM (Domain-Specific Modeling) es principalmente conocida como la idea de crear modelos para un dominio, utilizando un lenguaje focalizado y especializado para dicho dominio. Estos lenguajes se denominan DSLs (Domain- Specific Language) y permiten especificar la solución usando directamente conceptos del dominio del problema. Los productos finales son luego generados desde estas especificaciones de alto nivel. Esta automatización es posible si ambos, el lenguaje y los generadores, se ajustan a los requisitos de un único dominio. (Pons, 2010).

Se define como dominio a un área de interés para un esfuerzo de desarrollo en particular. En la práctica, cada solución DSM se enfoca en dominios pequeños porque el foco reductor habilita mejores posibilidades para su automatización y estos dominios pequeños son más fáciles de definir. Usualmente, las soluciones en DSM son usadas con relación a un producto particular, una línea de producción, un ambiente específico, o una plataforma. (Pons, 2010).

El desafío de los desarrolladores y empresas se centra en la definición de los lenguajes de modelado, la creación de los generadores de código y la implementación de frameworks específicos del dominio. (Pons, 2010).

Estos elementos no se encuentran demasiado distantes de los elementos de modelado de MDD. Actualmente puede observarse que las discrepancias entre DSM y MDD han comenzado a disminuir. Se puede comparar el uso de modelos, así como la construcción de la infraestructura respectiva en DSM y en MDD. En general, DSM usa los conceptos dominio, modelo, metamodelo, meta-metamodelo como MDD, sin mayores cambios y propone la automatización en el ciclo de vida del software. (Pons, 2010).

Los DSLs son usados para construir modelos. Estos lenguajes son frecuentemente pero no necesariamente gráficos. Los DSLs no utilizan ningún estándar del OMG para su infraestructura, es decir no están basados en UML, los metamodelos no son instancias de MOF, a diferencia usan por ejemplo MDF, el framework de Metadatos para este propósito. (Pons, 2010).

Finalmente, existe una familia importante de herramientas para crear soluciones en DSM que ayudan en la labor de automatización. En tal sentido, surge el término "*Software Factories*", que ha sido acuñado por Microsoft. Su descripción en forma detallada puede encontrarse en el libro de Greenfields [GS04] del mismo nombre. Una *Software Factory* es una línea de producción de software que configura herramientas de desarrollo extensibles tales como Visual Studio Team System con contenido empaquetado como DSLs, patrones y *frameworks*, basados en recetas para construir tipos específicos de aplicaciones. Visual Studio provee herramientas para definir los metamodelos, así como su sintaxis concreta y editores. En el capítulo 4 se verán con más detalle ésta y otras propuestas de herramientas que asisten en la tarea de construcción de modelos y lenguajes de modelado. (Pons, 2010).

2.5 Modelo

Durante el proceso de desarrollo de software se crean diferentes modelos (ver figura 5). Los modelos de análisis capturan sólo los requisitos esenciales del sistema de software, describiendo lo que el sistema hará independientemente de cómo se implemente. Por otro lado, los modelos de diseño reflejan decisiones sobre el paradigma de desarrollo (orientado a objetos, basado en componentes, orientado a aspectos, etc.), la arquitectura del sistema (distintos estilos arquitecturales), la distribución de responsabilidades (GRASP patterns (Larman 04), GoF patterns (GHJV94)).

Finalmente, los modelos de implementación describen cómo el sistema será construido en el contexto de un ambiente de implementación determinado (plataforma, sistema operativo, bases de datos, lenguajes de programación, etc.). Si bien algunos modelos pueden clasificarse claramente como un modelo de análisis, o de diseño o de implementación, por ejemplo, un diagrama de casos de uso es un

modelo de análisis, mientras que un diagrama de interacción entre objetos es un modelo de diseño y un diagrama de deployment es un modelo de implementación. En general, esta clasificación no depende del modelo en sí mismo sino de la interpretación que se de en un cierto proyecto a las etapas de análisis, diseño e implementación. (Pons, 2010).

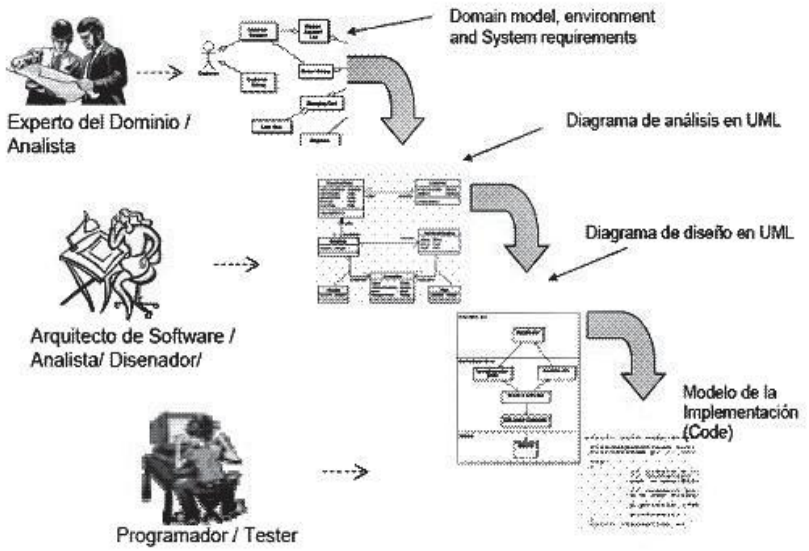


Figura 5: Diferentes modelos a lo largo de desarrollo de software

Fuente: (Pons, 2010)

2.5.1 ¿Cómo se define un modelo?

El modelo del sistema se construye utilizando un lenguaje de modelado (que puede variar desde lenguaje natural o diagramas hasta fórmulas matemáticas). Los modelos informales son expresados utilizando lenguaje natural, figuras, tablas u otras notaciones. Se habla de modelos formales cuando la notación empleada es un formalismo, es decir posee una sintaxis y semántica (significado) precisamente definidos. Existen estilos de modelado intermedios llamados semiformales, ya que en

la práctica los ingenieros de software frecuentemente usan una notación cuya sintaxis y semántica están sólo parcialmente formalizadas. (Pons, 2010).

El éxito de los lenguajes gráficos de modelado, tales como el Unified Modeling Language (UML) se basa principalmente en el uso de construcciones gráficas que transmiten un significado intuitivo; por ejemplo, un cuadrado representa un objeto, una línea uniendo dos cuadrados representa una relación entre ambos objetos. Estos lenguajes resultan atractivos para los usuarios ya que aparentemente son fáciles de entender y aplicar. Sin embargo, la falta de precisión en la definición de su semántica puede originar malas interpretaciones de los modelos, inconsistencia entre los diferentes sub-modelos del sistema y discusiones acerca del significado del lenguaje. Además, los lenguajes utilizados en MDD deben estar sustentados por una base formal de tal forma que las herramientas sean capaces de transformar automáticamente los modelos escritos en tales lenguajes. (Pons, 2010).

Por otro lado, los lenguajes formales de modelado poseen una sintaxis y semántica bien definidas. Sin embargo, su uso en la industria es poco frecuente. Esto se debe a la complejidad de sus formalismos matemáticos que son difíciles de en- tender y comunicar. En la mayoría de los casos los expertos en el dominio del sistema que deciden utilizar una notación formal focalizan su esfuerzo sobre el manejo del formalismo en lugar de hacerlo sobre el modelo en sí. Esto conduce a la creación de modelos formales que no reflejan adecuadamente al sistema real. (Pons, 2010).

La necesidad de integrar lenguajes gráficos, cercanos a las necesidades del dominio de aplicación con técnicas formales de análisis y verificación puede satisfacerse combinando ambos tipos de lenguaje. La idea básica para obtener una combinación útil consiste en ocultar los formalismos matemáticos detrás de la notación gráfica. De esta manera el usuario sólo debe interactuar con el lenguaje gráfico, pero puede contar con la base formal provista por el esquema matemático subyacente. Esta propuesta ofrece claras ventajas sobre el uso de un lenguaje informal, así como también sobre el uso de un lenguaje formal, ya que permite que los desarrolladores de software puedan crear modelos formales sin necesidad de poseer un conocimiento profundo acerca del formalismo que los sustenta. (Pons, 2010).

En sus orígenes los lenguajes gráficos como el UML carecían de una definición formal. A partir de su aceptación como estándar y de su adopción masiva en la industria, los investigadores se interesaron en construir una base formal para dichos lenguajes. El grupo pUML ("the precise UML group") fue creado en 1997 para reunir a investigadores y desarrolladores de todo el mundo con el objetivo de convertir al Unified Modelling Language (UML) en un lenguaje de modelado formal (es decir, bien definido). El grupo ha desarrollado nuevas teorías y técnicas destinadas a:

- Clarificar y hacer precisa la sintaxis y la semántica de Uml;
- Razonar con propiedades de los modelos Uml;
- Verificar la corrección de diseños Uml;
- Construir herramientas para soportar la aplicación rigurosa de Uml.

2.6. Transformación

El proceso MDD, muestra el rol de varios modelos, PIM, PSM y código dentro del framework MDD. Una herramienta que soporte MDD, toma un PIM como entrada y lo transforma en un PSM. La misma herramienta u otra, tomará el PSM y lo transformará a código. Estas transformaciones son esenciales en el proceso de desarrollo de MDD. En la figura 6 se muestra la herramienta de transformación como una caja negra, que toma un modelo de entrada y produce otro modelo como salida. (Pons, 2010).

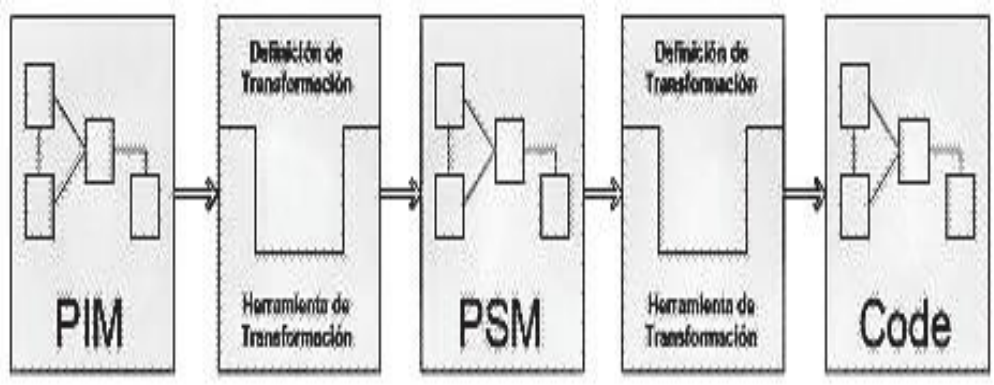


Figura 6: Definición de herramientas de transformación

Fuente: (Pons, 2010)

Si se abre la herramienta de transformación y se mira dentro, se muestra ver qué elementos están involucrados en la ejecución de la transformación. En algún lugar dentro de la herramienta hay una definición que describe como se debe transformar el modelo fuente para producir el modelo destino. Esta es la definición de la transformación. La figura 6 muestra la estructura de la herramienta de transformación. Se nota que hay una diferencia entre la transformación misma, que es el proceso de generar un nuevo modelo a partir de otro modelo, y la definición de la transformación. (Pons, 2010).

Para especificar la transformación, (que será aplicada muchas veces, independientemente del modelo fuente al que será aplicada) se relacionan construcciones de un lenguaje fuente en construcciones de un lenguaje destino. Se podría, por ejemplo, definir una transformación que relaciona elementos de UML a elementos Java, la cual describiría como los elementos Java pueden ser generados a partir de los elementos UML. Esta situación se muestra en la figura 7. (Pons, 2010).

En general, se puede decir que una definición de transformación consiste en una colección de reglas, las cuales son especificaciones no ambiguas de las formas en que un modelo (o parte de él) puede ser usado para crear otro modelo (o parte de él).

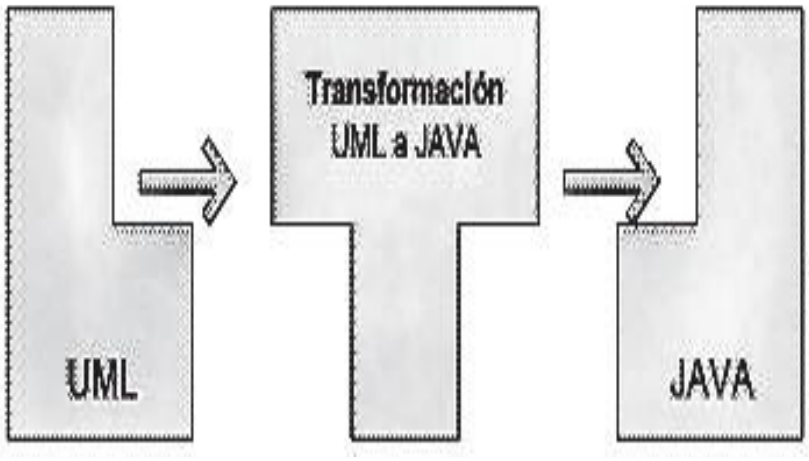


Figura 7: Definición de transformaciones entre lenguajes.

Fuente: (Pons, 2010)

Las siguientes definiciones fueron extraídas del libro de Anneke Kepple:

- Una transformación es la generación automática de un modelo destino desde un modelo fuente, de acuerdo con una definición de transformación.
- Una definición de transformación es un conjunto de reglas de transformación que juntas describen como un modelo en el lenguaje fuente puede ser transformado en un modelo en el lenguaje destino.
- Una regla de transformación es una descripción de como una o más construcciones en el lenguaje fuente pueden ser transformadas en una o más construcciones en el lenguaje destino. (Pons, 2010).

2.7. Reglas de transformación

A continuación se mostrara un ejemplo de una transformación de un modelo PIM escrito en UML a un modelo de implementación escrito en Java. Se transforma el diagrama de clases de un sistema a las clases de Java correspondientes a ese modelo. La figura 8 muestra gráficamente la transformación que se intenta realizar, la cual consta de 2 pasos: Se transforma el PIM en un PSM y luego se transforma el PSM resultante a código Java. (Pons, 2010).

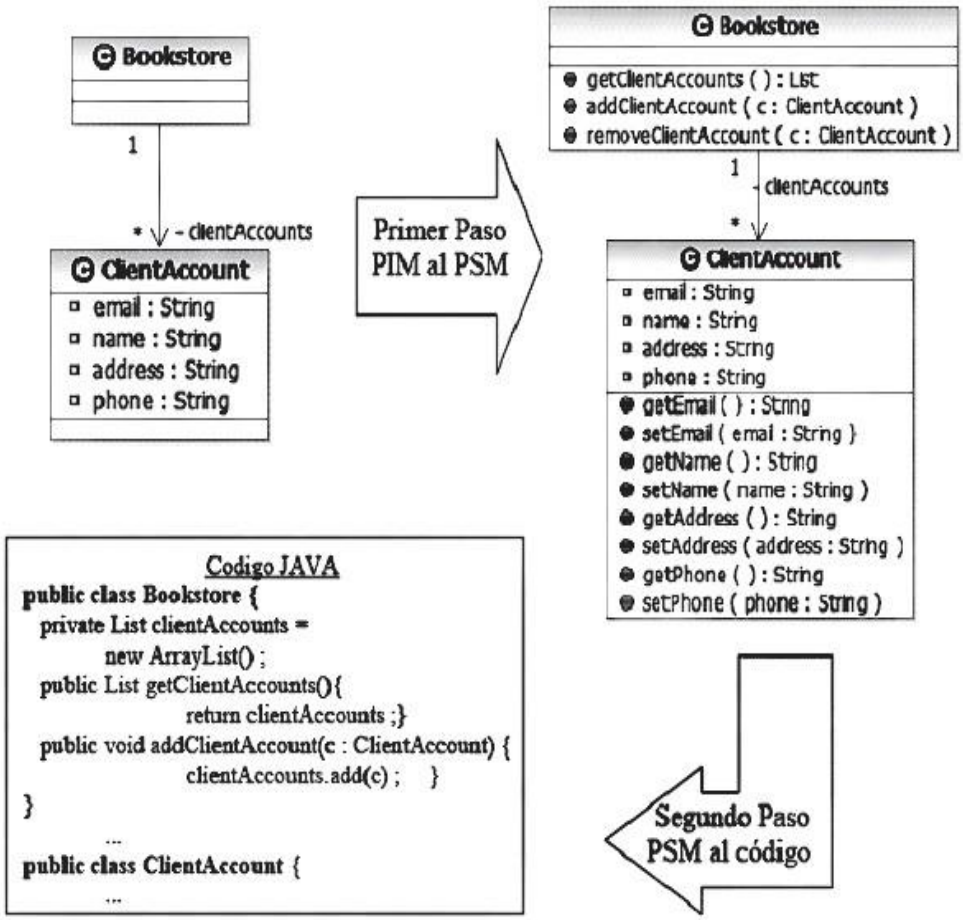


Figura 8: Ejemplo de transformación de PIM a PSM y de PSM a código.

Fuente: (Pons, 2010)

En (Pons, 2010) indica que para la generación del PIM como del PSM son modelos útiles ya que proveen el nivel de información adecuado para diferentes tipos de desarrolladores y otras personas involucradas en el proceso de desarrollo de software. Existe una clara relación entre estos modelos. La definición de la transformación que debe aplicarse para obtener el PSM a partir del PIM consiste en un conjunto de reglas. Estas reglas son:

- Para cada clase en el PIM se genera una clase con el mismo nombre en el PSM.
- Para cada relación de composición entre una clase llamada class A y otra clase llamada class B, con multiplicidad 1 a n, se genera un atributo en la clase classA con nombre classB de tipo Collection.

- Para cada atributo público definido como attributeName: Type en el PIM los siguientes atributos y operaciones se generan como parte de la clase destino:

- Un atributo privado con el mismo nombre: attributeName: Type o Una operación pública cuyo nombre consta del nombre del atributo precedido con "get" y el tipo del atributo como tipo de retorno: getAttributeName(): Type.
- Una operación pública cuyo nombre consta del nombre del atributo precedido con "set" y con el atributo como parámetro y sin valor de retorno: setAttributeName(att: Type).

El siguiente paso consistirá en escribir una transformación que tome como entrada el PSM y lo transforme a código Java. Combinando y automatizando ambas transformaciones se puede generar código Java a partir del PIM. (Pons, 2010).

2.8 Características Técnicas Drone Syma x9

- 2-en-1 Fly & Drive
- Estructura de 4 ejes - Capaz de volar en interiores y exteriores y es resistente al viento.
- 360 ° Eversion & Throwing Flight - Un giro de 360 °.
- 2.4GHz Radio control - Vuela o conduce a larga distancia con RC múltiples al mismo tiempo con interferencia mínima.
- 6 AXIS GYRO - ¡Equipado con los últimos sistemas de control de vuelo de 6 ejes, bloqueo 3D, Más vuelo regular, operando más a la fuerza!

2.9 Características únicas Drone Syma x9

El Syma X9 tiene bastantes características únicas que lo diferencian de otros drones en el mercado. Para los arrancadores, es una cruz entre un drone de quadrocopter y un coche de RC. Se las arregla para combinar ambos aspectos con

facilidad, incluso cambiando de uno a otro con un solo movimiento de un interruptor, sin tener que parar o apagar el drone.

Otra característica que es única para el X9 es la batería real que se ejecuta en el momento de la conexión. Mientras que las estadísticas y la carga podrían ser similar a otras baterías drone por ahí, el diseño y el formato de la batería no lo son. Dicho esto, mientras que los drones diferentes tienen baterías intercambiables, el X9 tiene una batería especial que puede trabajar con los dos modos independientemente.

Por un lado, esto puede parecer un poco estrecho avistado por parte de la empresa, sin embargo, es por una buena razón. La batería X9 es una de las baterías de drones más pequeñas y ligeras del mercado, por no mencionar el hecho de estar diseñada para trabajar de forma específica y exclusiva con la Syma X9, la batería tiene menores posibilidades de mal funcionamiento de la fritura del motor, problemas que tienen Se ha sabido que se producen con los drones que han estado utilizando diferentes baterías a lo largo de los años.

Una última cosa que es, en toda justicia, único para el Syma X9 es la capacidad de cambiar de quadrocopter a RC coche sin ningún problema. Ha habido otros drones híbridos en el mercado en el pasado, sin embargo, cambiar entre los modos necesarios para apagar el drone y reiniciarlo para que los cambios surtan efecto.

CAPITULO III

DISEÑO DE LA ARQUITECTURA

3.1. Levantamiento de requerimientos a partir de un dron especifico.

El dron especifico que se va a utilizar es un dron terrestre por lo tanto se va a proceder a controlar los movimientos del dron, para poder generar una ruta definida que el dron podrá seguir independientemente.

A continuación se describen los requerimientos funcionales del sistema, tomando como base el levantamiento de requerimientos efectuado personalmente, éstos se han categorizado y detallado para definir el alcance de la solución a implementar.

3.1.1. Requerimientos funcionales dron terrestre.

- Definición de PWM para el control de movimiento.
- Definición de los movimientos.
- Establecimiento de conexión entre PC y drone terrestre.
- Definición de distancia de cobertura entre emisor y receptor.
- Establecimiento de tiempos para cada movimiento.

3.1.2. Requerimientos funcionales Arduino uno.

- ConFiguración de pines de entrada y salida.
- ConFiguración de pines análogos de salida.
- Establecimientos de voltajes de salida para cada uno de los movimientos.
- ConFiguración de dutty cicle para PWM.
- Establecimiento de Delay para cada pausa.
- Establecimiento de conexión entre Arduino y PC.

Para la selección de pines de salida se conFigura el Arduino Uno como análogo para poder utilizar el PWM para poder generar los voltajes adecuados para cada uno de los movimientos correspondientes de acuerdo a lo que indica la tabla 1 donde se puede ver cada una de las conFiguraciones y los voltajes que se deben emplear para el correcto funcionamiento teniendo en cuenta que para que el drone terrestre se encuentre en pausa se debe colocar el pin 3 y pin 5 del Arduino en 1.6 v.

Tabla 1.

Definición de voltajes para salida.

MOVIMIENTOS	Voltaje (v)		# BITS SALIDA	
	Pin 3	**Pin 5**	**Pin 3**	**Pin 5**
ADELANTE	0 V	1.6 V	0	155
ADELANTE DERECHA	0V	3.3 V	0	255
ADELANTE IZQUIERDA	0V	0V	0	0
ATRÁS	3.3 V	1.6 V	255	155
ATRÁS DERECHA	3.3 V	3.3 V	255	255
ATRÁS IZQUIERDA	3.3 V	0 V	255	0
PAUSA	1.6 V	1.6 V	155	155

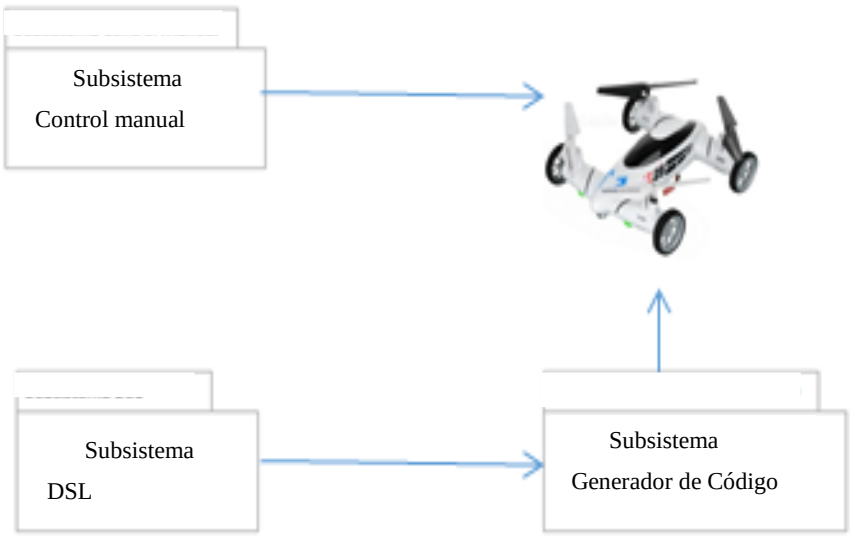


Figura 9: Modelo PIM del sistema

En la figura 9 se puede ver el modelo independiente de plataforma ¨PIM¨ del proyecto donde se puede diferenciar tres packages bien definidos que van a ser detallados a continuación.

- **Subsistema control Manual:** En este package se encuentra presente toda las herramientas y clases con la que se puede generar una comunicación serial con el Drone y poder manejarlo en tiempo real en su interior este package presenta la arquitectura definida en la figura 10.

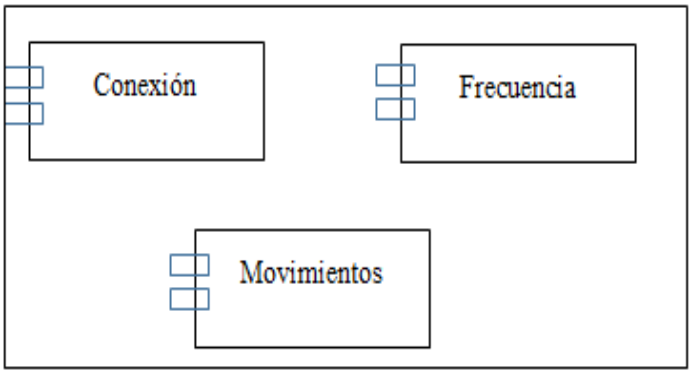


Figura 10: Arquitectura interna package Subsistema de control manual

Como se puede apreciar en el subsistema de control manual se definen tres clases más que son las encargadas de la generación de rutas en un tiempo real mediante la comunicación serial con el Arduino, donde se debe establecer la conexión con ayuda del package conexión, se dirige el dron con la ayuda del package movimientos y el package frecuencia lo que realiza es el establecimiento del ciclo de trabajo del puerto análogo para que no se presenten interrupciones en el movimientos.

- **Subsistema DSL:** En este package se encuentra todo lo referente a la generación de la ruta que seguirá el dron también la generación de cada uno de los tiempos para que la aplicación funcione, es decir esta es la base de la investigación con ayuda del package de generación de código.
- **Subsistema de Generación de Código:** Este package presenta todas las herramientas, clases y subsistemas que son encargados de la transformación de modelo a texto que es la base de esta aplicación, este package es el encargado de la generación de código para grabar en Arduino tanto este package como el package Subsistema DSL se explicaran en el Diseño del metamodelo.

3.2. Diseño del Meta Metamodelo y Metamodelo.

Primera para el desarrollo del metamodelo del DSL que se va a implementar se debe definir las características de Drone que se va a utilizar, el drone a utilizarse es un Drone Syma x9 como el que se muestra en la figura 11, este dron presenta algunas características propias con las que se va a trabajar para definir el metamodelo mas optimo, este drone al ser un dispositivo básico lo que se va a proceder a controlar son sus movimientos como serán:

- Adelante
- Atrás
- Izquierda
- Derecha
- Detener

Figura 11: Drone Syma X9

También se va a definir unos tiempos en segundos que van a ser de 1 10 segundos dependiendo lo que el usuario requiera para el diseño de la ruta a seguir, para el diseño que se va a modelar se debe también tomar en cuenta que el dron para sus movimientos cuenta con dos motores en su interior que son uno para mover adelante-atrás y otro para derecha a izquierda, tomando en cuenta esto se debe definir cómo funciona cada uno de estos motores.

El funcionamiento de los motores se controla mediante dos potenciómetros colocados en el interior del control remoto con los cuales se procede a definir los voltajes de operación que se encuentra definidos en la tabla 1, con estos voltajes ya definidos y las pausas definidas se precede al diseño del caso de uso para el control de cada una de las características del Drone Syma x9 (ver FIGURA 3.5).

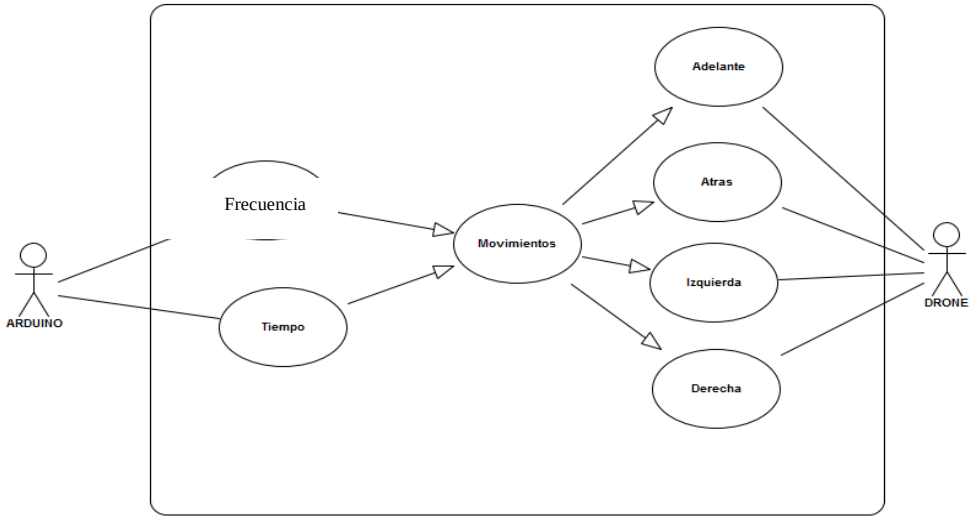


Figura 12: Diagrama de casos de uso del sistema a implementar

Mediante el diagrama de casos de uso se procede al modelado de la Herramienta DSL con cada una de sus características y clases como se muestran en la figura 13 este modelado muestra la estructura interna del Subsistema DSL mencionado anteriormente.

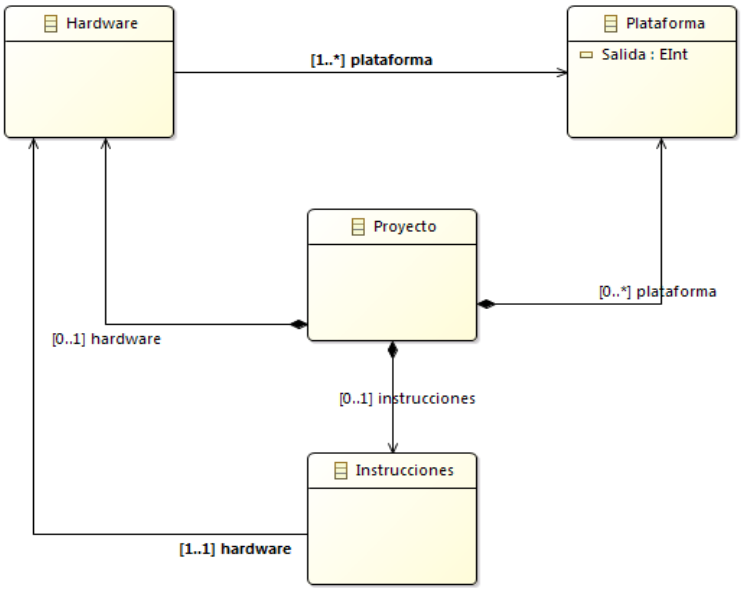


Figura 13: Metamodelo de la Herramienta DSL

El metamodelo desarrollado para esta herramienta DSL posee cuatro partes bien diferenciada o paquetes, tal y como se denomina en el ámbito de modelado. Cada package estará conectado a uno o varios packages, formando en conjunto el modelo que definirá el lenguaje de dominio que hará uso nuestra herramienta. A continuación se describe cada una de las partes del metamodelo.

3.2.1. Proyecto

Este paquete define todas las acciones que la herramienta puede realizar es el esquema general de la configuración y el package que administra todas las características y propiedades de la herramienta.

Este package se conecta a su vez directamente con todos los package de la arquitectura planteada es el modelo más abstracto de la representación.

Por último, el proyecto podrá poseer una o varias propiedades definidas sobre el nivel de abstracción PIM. Este package estará conectado a su vez con el package "Instrucciones", el package "Plataforma" y el package "Hardware", dando la posibilidad definir independientemente cada una de las características propias que presenta el Drone.

3.2.2. Hardware

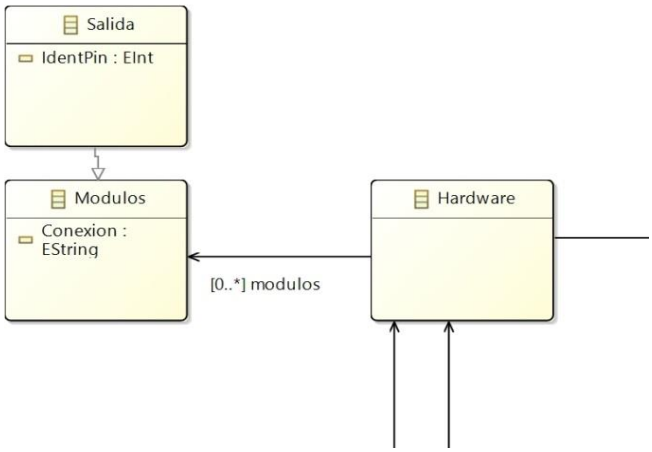


Figura 14: Modelo de hardware

En la figura 14 se puede apreciar la configuración del paquete Hardware, este package se encarga de realizar las acciones físicas o de hardware del Arduino con el drone, es el encargado de la configuración de la parte física de la arquitectura. En este package cuenta con algunos paquetes como son:

- Módulos: Es paquete que se encarga mediante sus atributos del establecimiento de la comunicación con el puerto serial mediante la configuración del puerto COM con su atributo Conexión.
- Salida: Que cuenta con un atributo propio que es el encargado de la selección de pin de salida y de su voltaje de salida.

Este package está directamente relaciona con el paquete plataforma que se procederá a explicar sus características más adelante.

3.2.3. Plataforma

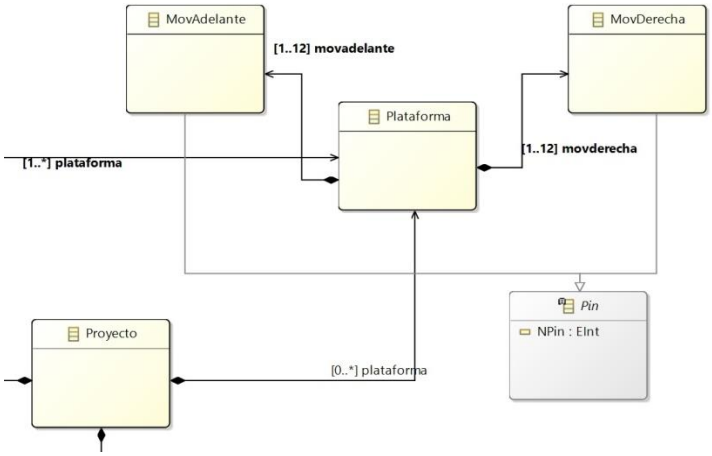


Figura 15: Modelo de plataforma

Este package se encuentra la configuración y definición de cada uno de los movimientos en la parte física, esta a su vez conectado directamente al package de Hardware, mediante la unión de estos 2 paquetes se procede a la configuración de los movimientos en la parte física, es decir conFigura los pines de salida y que pin es el indicado para mover cada uno de los motores que se definieron en el drone.

La arquitectura interna de este package es como se muestra en la figura 15, donde posee tres package que sirven para la selección de cada movimiento y la selección del pin de salida con ayuda del paquete Pin que es el encargado de la selección del pin de salida.

3.2.4. Instrucciones

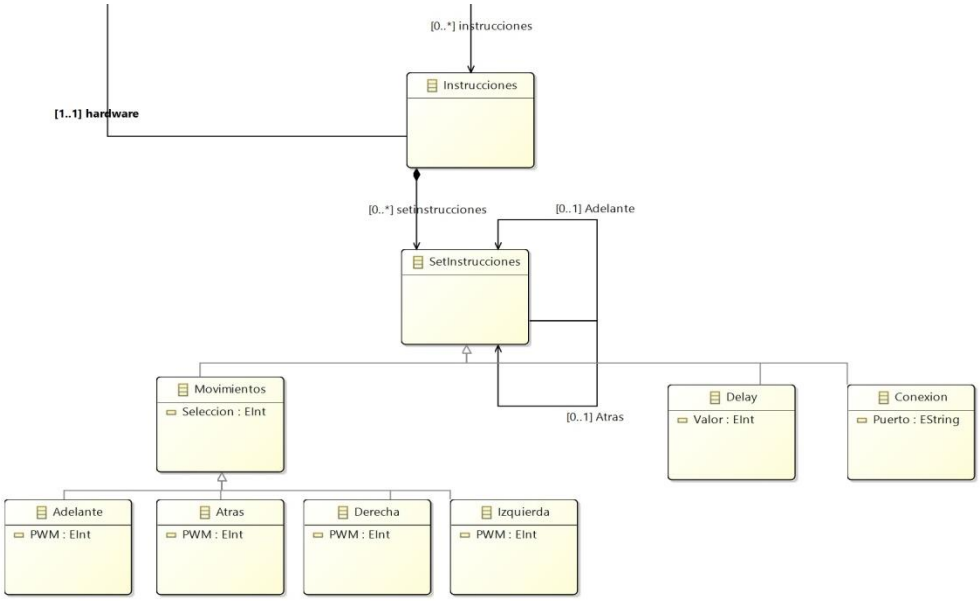


Figura 16: Modelo de instrucciones

Este paquete muestra la arquitectura interna del subsistema generación de código descrito anteriormente, es el encargado directamente de la generación del código "software" para el control de la herramienta DSL, en este package está todo el set de instrucciones para el manejo del drone.

Este paquete presenta la arquitectura que se muestra en la figura 16, este paquete se encuentra conectado directamente con el paquete Hardware y mediante la unión de estos dos paquetes se genera a totalidad la herramienta tanto en la generación de código como en el control del dron.

Este paquete consta de los siguientes paquetes en su interior:

- **Movimientos:** Este paquete consta de 4 paquetes más en su interior que son encargados de la generación de las direcciones que puede asumir el dron y mediante el atributo propio que es PWM el control de los voltajes para el control, de los motores.
- **Tiempo:** Esta clase ayuda en la generación del set de instrucciones para la configuración de tiempos para poder generar la ruta del dron con tiempos que oscilan entre 1 y 10 segundos.

- **Conexión:** Este paquete es el encargado de la generación del set de instrucciones de código que se encargan de la configuración del puerto serial y de todas sus características.

Luego de haber establecido cada uno de los paquetes que conforman el metamodelo de la aplicación se obtiene el modelo final mostrado en la figura 17, que es el metamodelo de la herramienta DSL total que se va a implementar.

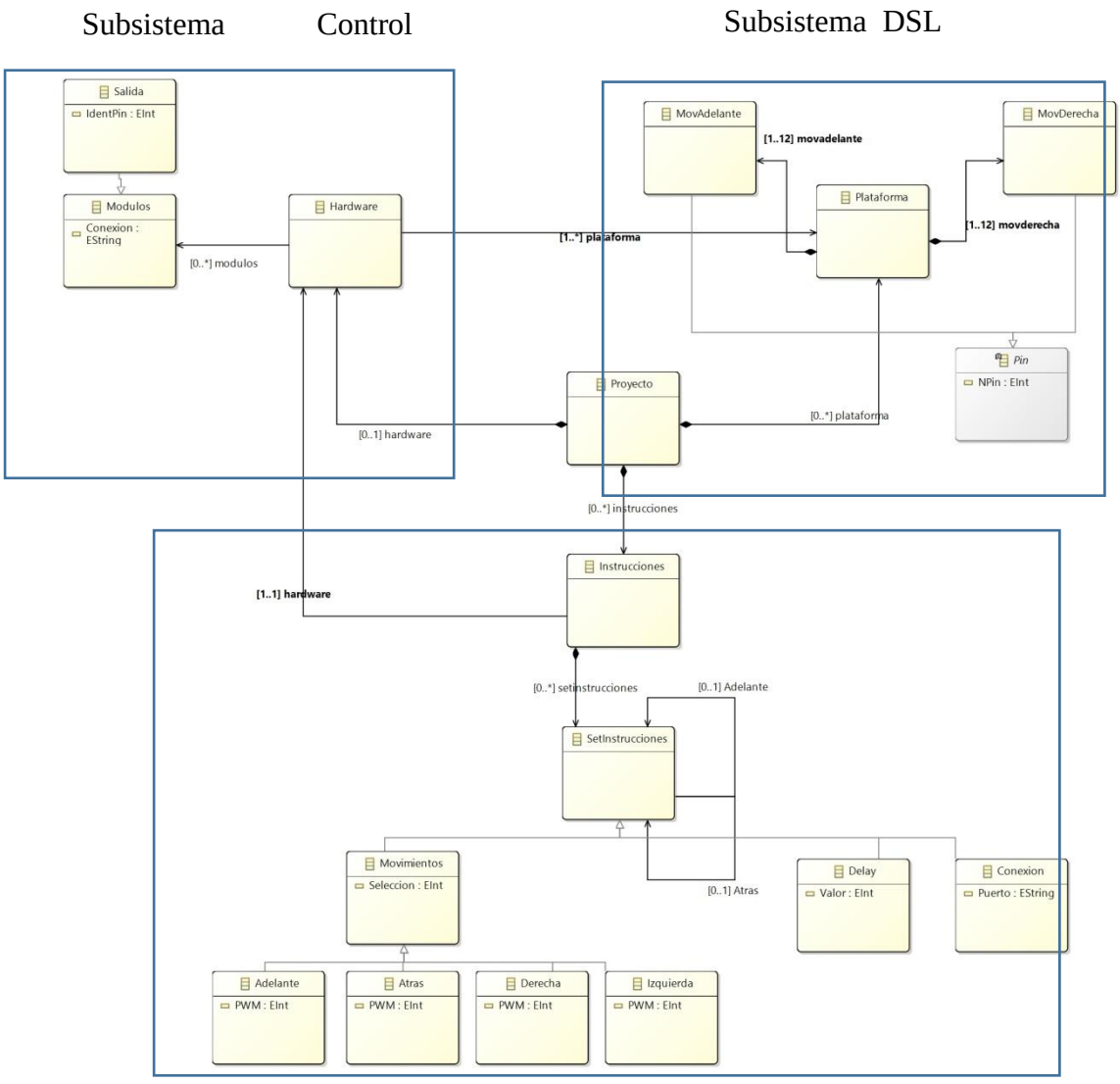


Figura 17: Metamodelo completo final

CAPITULO IV

IMPLEMENTACION DE LA HERRAMIENTA DSL.

4.1 Implementación del modelo.

En este capítulo se presenta las diferentes partes que integran el desarrollo de la implementación del editor gráfico del control de un Dron Terrestre. Este editor tendrá como objetivo el apoyo en el diseño y la planeación de una ruta definida para el control de un drone terrestre.

Primero para implementar el modelo y realizar la herramienta DSL se debe tener en cuenta los siguientes requerimientos funcionales y tomar en cuenta el diagrama de caso de uso de la aplicación (ver figura 18).

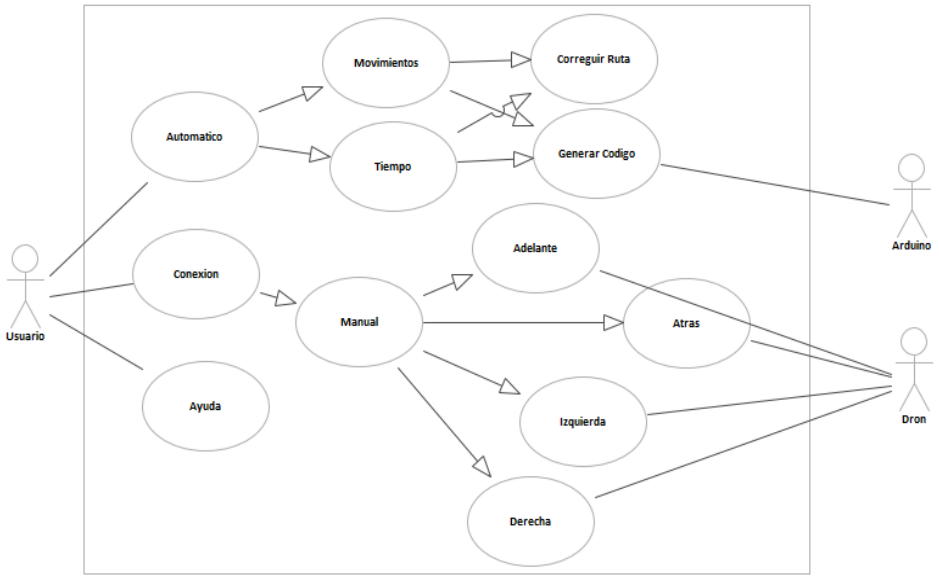


Figura 18: Diagrama de caso de uso aplicación.

4.1.1. Requerimientos funcionales de interfaz gráfica.

- La solución conecta automáticamente el usuario propietario con el dron.
- El campo de conexión aceptara únicamente letras en mayúscula para poder conFigurar el puerto serie.
- El campo tiempo se asigna con la selección del número de botones seleccionados.
- El campo movimientos está definido por fechas que se pueden seleccionar independientes para elegir la dirección del movimiento.
- La aplicación consta con un botón de stop para parar el dron automáticamente al presionarlo.
- La aplicación consta de un botón de Generar código que será el encargado al presionarlo de crear un archivo .ino para poder grabarlo en el Arduino y manejar el dron.
- La aplicación consta de un botón de cambio de ruta que tiene la función de que el usuario puede definir otra ruta si cometió un error.
- En el campo del panel de direcciones muestra automáticamente las direcciones seleccionas y se puede visualizar la ruta a seguir.

4.1.2. Requerimientos de seguridad.

- El sistema controla el acceso únicamente a la función terrestre del Dron mediante conexión serial.
- El sistema envía una alerta al administrador del sistema cuando ocurra alguno de los siguientes eventos: Al ingresar un puerto serial no indicado, Al no tener ningún dispositivo conectado.
- Los datos generados al presionar el botón Generar código solo se pueden cambiar modificando el archivo .ino que se genera o al cambiar de ruta.
- Al abrir varias veces la aplicación se generara un error al grabar la ruta definida.

4.1.3. Requerimientos de interfaces externas (Hardware y Software).

- El software puede ser utilizado en los sistemas operativos Windows, Linux.
- La aplicación debe poder utilizarse sin necesidad de instalar ningún software adicional a excepción de java.

4.2 Implementación de DSL y definición de áreas.

En este apartado se procede a la generación del modelo para la generación de la plataforma de generación de código a partir de modelos en Eclipse, primero una vez ya definido el esquema en el capítulo anterior, se basa en eso para la generación del modelo en java con la ayuda de los casos de uso definidos en la figura 18, para la generación del modelo de implementación en java se deben acoplar con las características definidas anteriormente para que sea un modelo homogéneo como se muestra en la figura 19.

En este Uml se pueden distinguir algunos bloques que lo conforman a continuación se detalla los más importantes:

- **Ventana:** En este paquete se encuentra ubicados todos los componentes de la interfaz gráfica del Subsistema de control manual.
- **Ventana 1:** En esta interfaz se encuentran ubicados todos los componentes de la interfaz gráfica de la herramienta DSL en este paquete se encuentran las clases de definir ruta y de definir tiempos que se encargan de los movimientos con la conexión de los paquetes de dirección y con la configuración de la tiempo como se explicara más adelante.
- **Conexión:** Con este paquete se conFiguran todos los componentes para el establecimiento de una comunicación seria con el Arduino.
- **Arriba = Abajo = Derecha = Izquierda:** Este grupo de paquetes son los encargados de la generación del set de instrucciones de cada uno de los movimientos que puede ejecutar el dron.

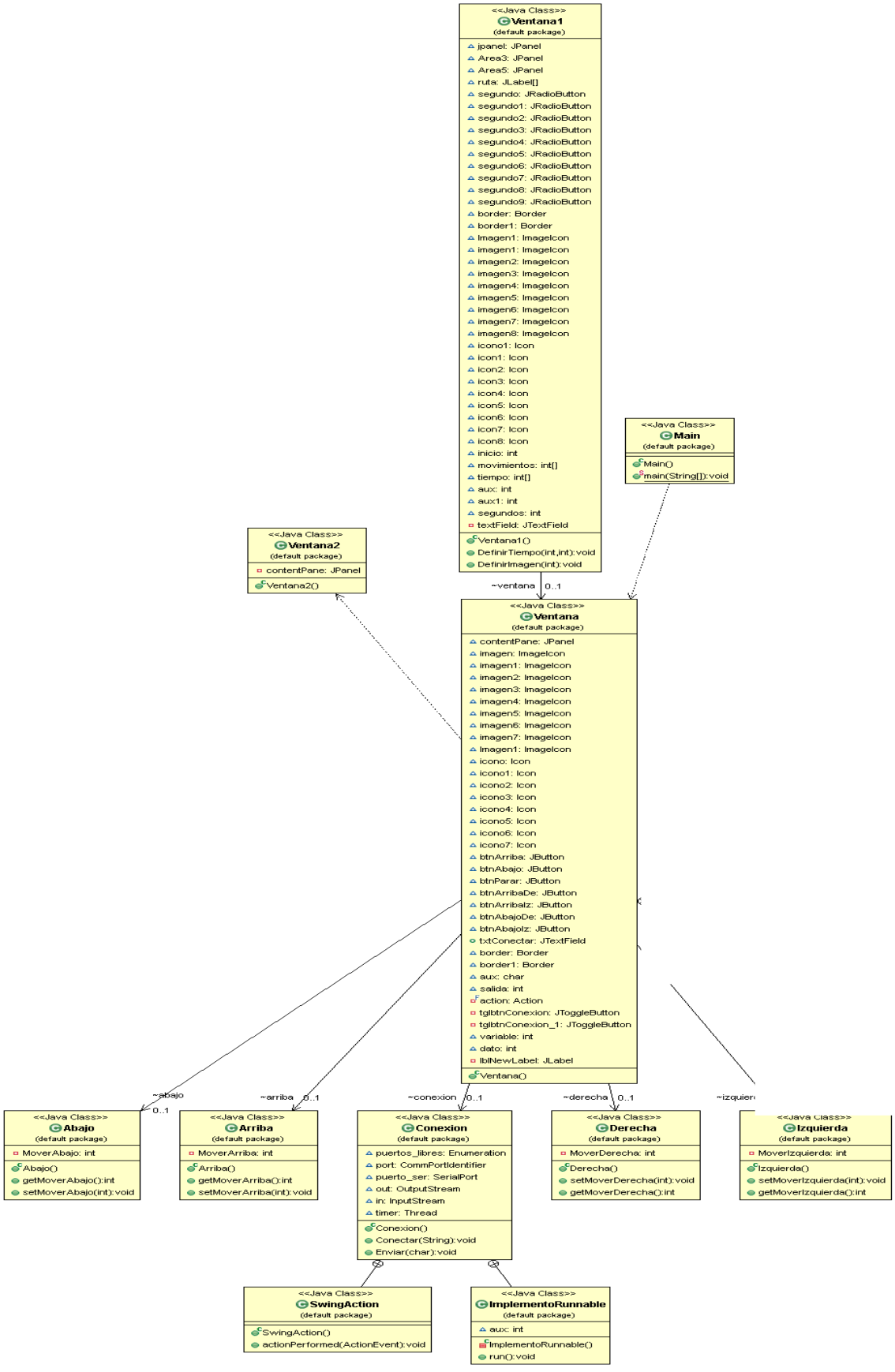


Figura 19: Diagrama UML de la aplicación.

Antes de la trasformación de modelos a códigos se debe definir correctamente las áreas que van a conforman la plataforma y cada una de sus características.

4.2.1 Definición de áreas Modo Manual

Primero se debe mencionar que este modo es un extra de generación de la herramienta ya que la herramienta DSL de generación de código se encuentra en el Modo Automático.

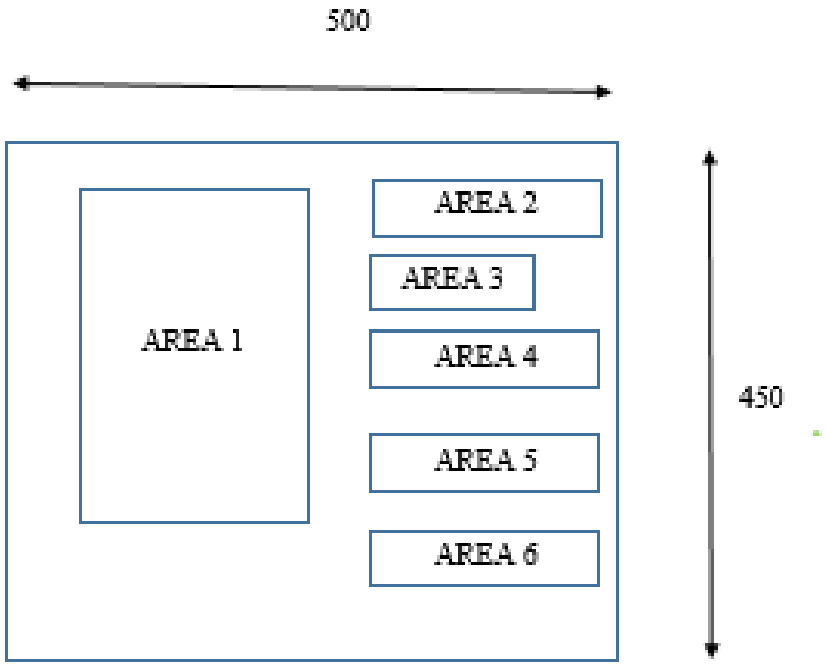


Figura 20: Distribución de áreas de FRAME Manual

Para de la definición de la interfaz del Modo Manual como se ve en la figura 20, se presenta una serie de áreas que van a estar conformadas por diferentes componentes propios de java, el área 1 para su despliegue necesita que se presione el botón con el nombre de MANUAL ubicado en el area5 y el área 1 se encuentra dividido en partes donde se despliegan una serie de botones en total 7 botones que son los encargados de cada una de las direcciones y la pausa que el dron puede asumir, mientras que en el área 2 se encuentra un botón llamado CONECTAR que es el encargado con la ayuda del COM que se obtiene de extraer el texto que se

encuentra en el textField del área 3 de la conexión del puerto serial con el Arduino para la ejecución de movimientos del dron en tiempo real, el área 4 presenta un botón llamado AUTOMATICO el cual al ser presionado genera la interfaz gráfica de la herramienta DSL que se va a implementar, el área 5 tiene un botón llamado Manual donde su única función es la desplegar el panel de movimientos en el área 1 y finalmente el área 6 tiene un botón llamado AYUDA el cual al ser accionado muestra un menú de ayuda para que el usuario tengo una explicación del funcionamiento de la plataforma. La interfaz que se genera en el modo manual es la que se presenta en la figura 21.

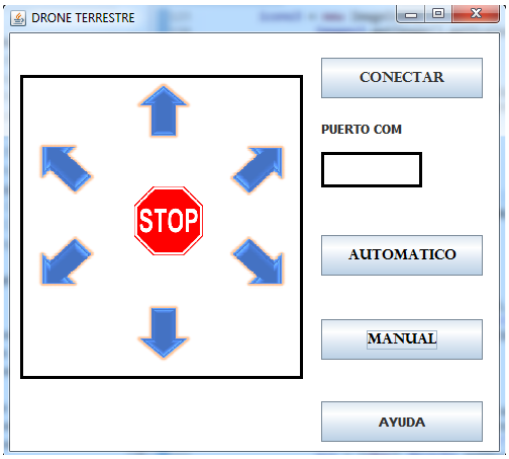


Figura 21: Ventana Modo Manual

4.2.2 Definición de áreas Modo Automático

Para la generación del modo automático se genera un área de trabajo como se muestra en la figura 22, donde se puede apreciar la distribución de cada uno de los elementos en el FRAME de esta interfaz, cabe recalcar que esta interfaz es la herramienta DSL propia del proyecto.

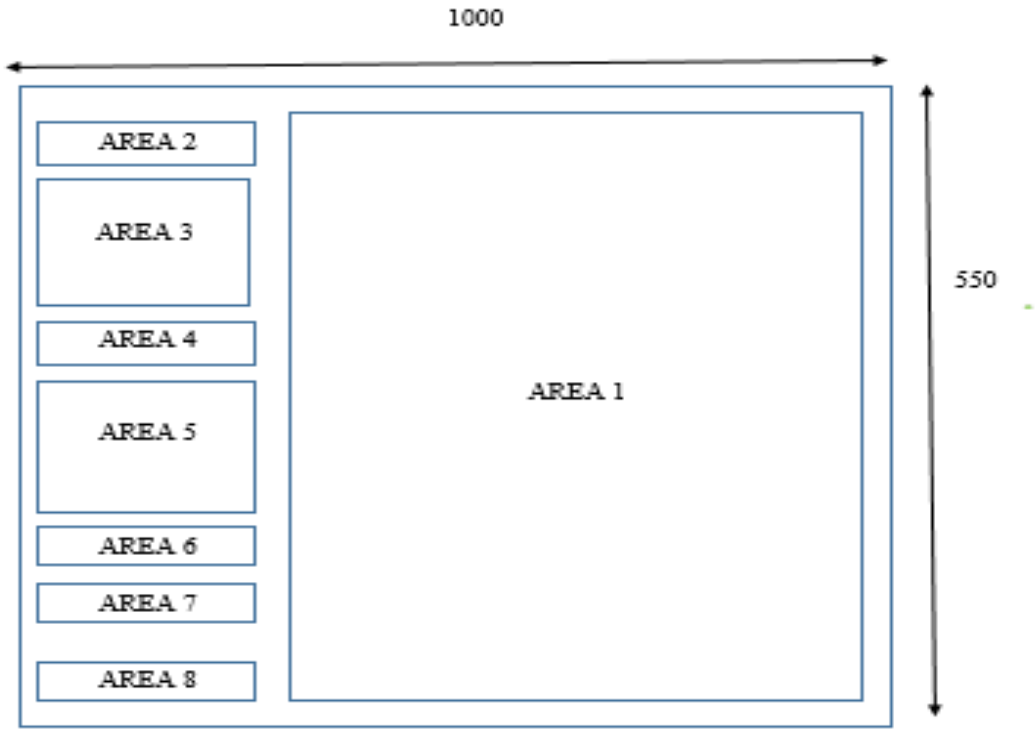


Figura 22: Distribución de áreas de FRAME Automático

Como se puede ver en la figura 22, el Frame de Automático está definido por 8 áreas bien marcadas las cuales tienen cada una función específica en la interfaz gráfica como se explicara a continuación:

- **Área 1:** Esta área presenta un panel que se encuentra dividido en forma de matriz de 4 columnas por 3 filas en donde se muestra de manera visual cada uno de los movimientos y de los tiempos que sean seleccionados en el Área 3 y Área 5.
- **Área 2:** Esta área presenta un botón llamado MOVIMIENTOS el cual es el encargado de mostrar u ocultar el panel del área 3.
- **Área 3:** Esta área está compuesta por un panel en donde en su interior tiene 7 botones distribuidos uniformemente para la selección de cada uno de los movimientos establecidos para el drone.
- **Área 4:** Esta área presenta un botón llamado TIEMPOS el cual es el encargado de mostrar u ocultar el panel del área 5.
- **Área 5:** Esta área está compuesta por un panel en donde se encuentra 10 radiobotons que dependiendo el número de estos que sea seleccionado ayudaran para

la colocación del tiempo del DELAY en segundos mediante unas reglas de transformación que se indicaran a continuación.

- **Área 6:** En esta área se encuentra el botón de REGRESAR el cual al ser accionado regresa la plataforma al modo manual anteriormente explicado.
- **Área 7:** En esta área se encuentra el botón de CAMBIAR RUTA en cual al ser accionado limpia por completo el área 1 para que con esto se pueda definir una nueva ruta si el usuario cometió alguna equivocación.
- **Area 8:** En esta área se encuentra la función principal de la herramienta DSL generada ya que presenta un botón llamado GENERAR CODIGO el cual al ser accionado muestra una ventana en donde el usuario puede seleccionar en que ubicación va a ser guardado el archivo .ino que va a ser grabado en el Arduino para la ejecución de ruta del drone.

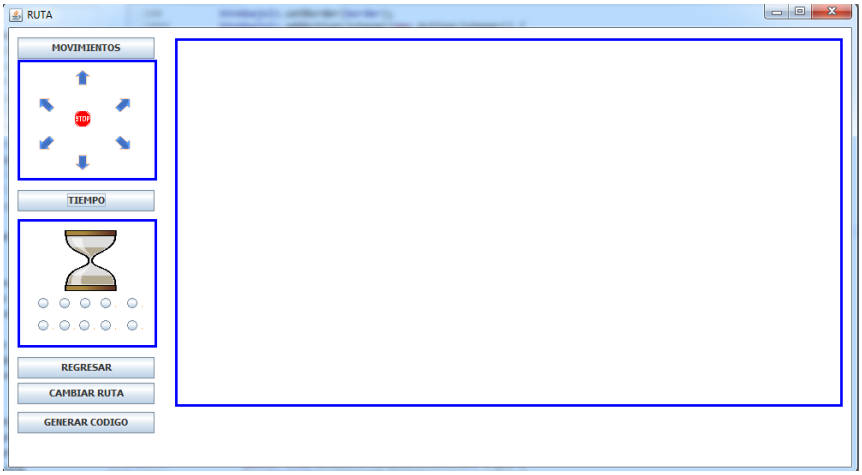


Figura 23: Ventana Modo Automático

Como muestra la figura 23 al estar configurada cada una de las áreas como se explicó se obtiene una plataforma de esta manera con cada uno de sus componentes.

4.3 Conversión modelo a texto M2T.

Para la generación del texto de la herramienta DSL se deben definir algunas normas de transformación de PSM a texto "código" para con esto ya tener lista nuestra plataforma, para el establecimiento de estas reglas de transformación se debe tomar en cuenta el diagrama de flujo de la figura 24.

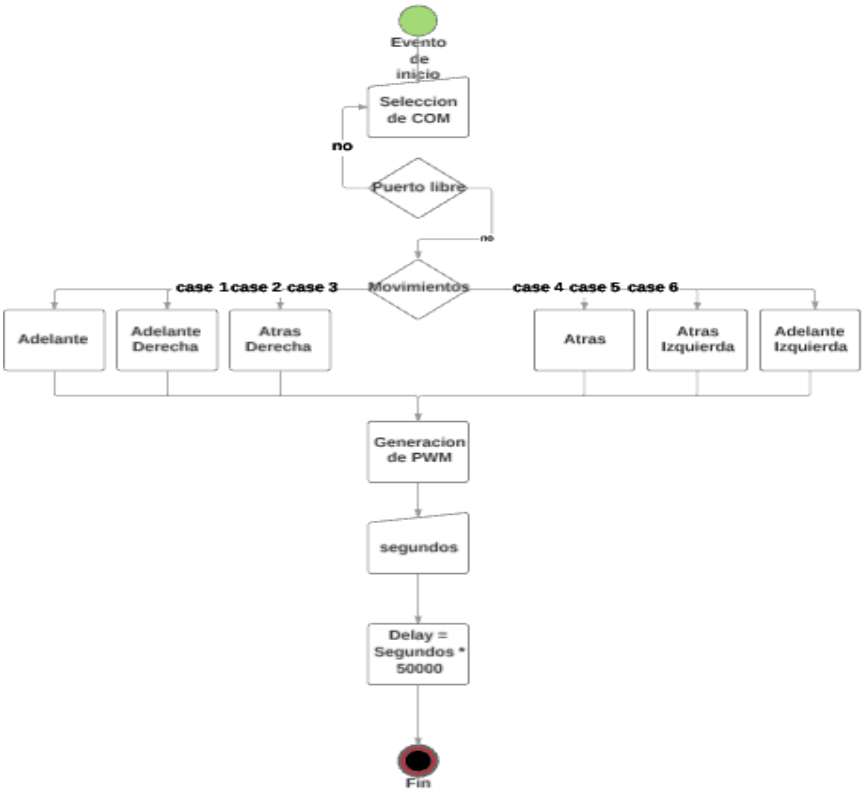


Figura 24: Diagrama de flujo de la herramienta DSL.

Como se puede observar en la figura 24 el sistema consta con algunas transformaciones propias para la generación de modelo a texto como se va a explicar a continuación.

- Para el establecimiento de conexión se emplea un ciclo de repetición condicional para saber si el puerto se encuentra libre, si es si se establece la conexión y si es no se vuelve a pedir otro puerto.

- Para la selección de cada uno de los movimientos se emplea un ciclo de selección para que dependiendo la opción se genere un número que muestra que movimiento.
- Para la pausa se emplea la Ecuación 1.
- El establecimiento de PWM ya se encuentra definido por default en cada movimiento.
- Cada bloque puede poseer otras reglas de transformación que se explicaran en su momento.

$$Delay = numero\ de\ segundos * 50000 \quad \textbf{(1)}$$

4.3.1 Ventana

En el paquete ventana se encuentran todas las instrucciones que son necesarias para el modo Manual a continuación se definirán algunas reglas para la trasformación de cada uno de sus elementos (ver figura 25):

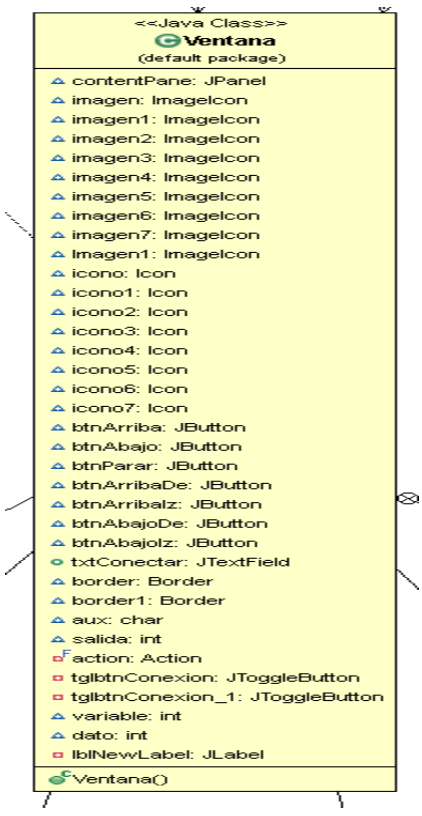


Figura 25: Package Ventana

En este package definen todas las herramientas que van a ser empleadas para la selección de mando tanto manual y automático, la conexión con Arduino y el menú de ayuda que muestra una idea de cómo utilizar la plataforma.

En este package se encuentra algunas herramientas que van a ayudar a la generación de la interfaz gráfica y a la ayuda de la generación de código.

En este package la transformación más importante que se emplea es la del establecimiento de la conexión.

4.3.1.1. Conexión

La transformación que se va a realizar para la obtención del texto es la del paquete de Conexión el cual presenta unos atributos bien definidos y unas clases propias de este paquete como se puede ver en la figura 26.

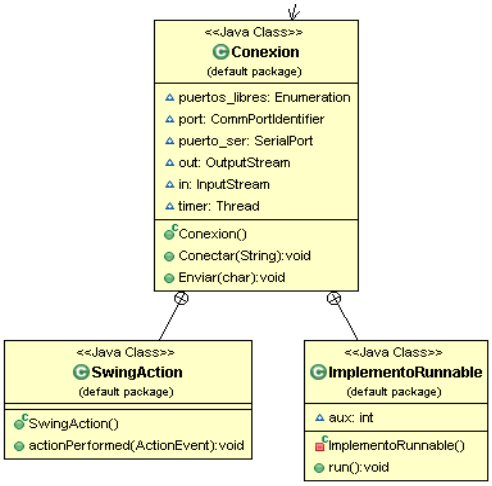


Figura 26: Package conexión

Este package es el encargado de la conexión entre Arduino y la PC para su transformación se toma en las siguientes reglas de transformación las cuales muestran lo siguiente:

1. Para verificar si se puede establecer conexión se ejecuta la siguiente línea de código puertos_libres = CommPortIdentifier.*getPortIdentifiers*(); la cual indica que puerto se encuentran libres para la conexión.

2. Para que a comunicación se mantenga estable un bucle WHILE el cual se encuentra activo mientras la variable puertos_libres se encuentre en sí.

3. Para el establecimiento de la comunicación se necesita la función IF la cual pregunta si el nombre del puerto colocado en área 3 del modo manual es igual al del puerto libre.

4. Posteriormente se hace la configuración de cada una de las variables que se encuentran en la comunicación serial y se va a obtener un codigo como el siguiente:

```
public void Conectar(String Nombre) {
    puertos_libres = CommPortIdentifier.getPortIdentifiers();
    int aux=0;
    while (puertos_libres.hasMoreElements())
    {
     port = (CommPortIdentifier) puertos_libres.nextElement();
     int type = port.getPortType();
     if (port.getName().equals(Nombre))
     {
         try {
            System.out.println("Conexion establecida");
            puerto_ser = (SerialPort) port.open("puerto serial", 2000);
              int baudRate = 9600; // 9600bps
              //conFiguracion de arduino
               puerto_ser.setSerialPortParams(
                     baudRate,
                     SerialPort.DATABITS_8,
                     SerialPort.STOPBITS_1,
                     SerialPort.PARITY_NONE);
               puerto_ser.setDTR(true);
               out = puerto_ser.getOutputStream();//salida de java
               in = puerto_ser.getInputStream(); // entrada de java
               timer.resume();
               Enviar('8');
        } catch (  IOException e1) {
        } catch (PortInUseException e1) {
           e1.printStackTrace();
        } catch (UnsupportedCommOperationException e1) {
           e1.printStackTrace();
```

```
            }

        break;
            }
    }
    }
```

Con las reglas de trasformación definidas anteriormente se logra la generación de este set de instrucciones completo.

4.3.3 Ventana 1

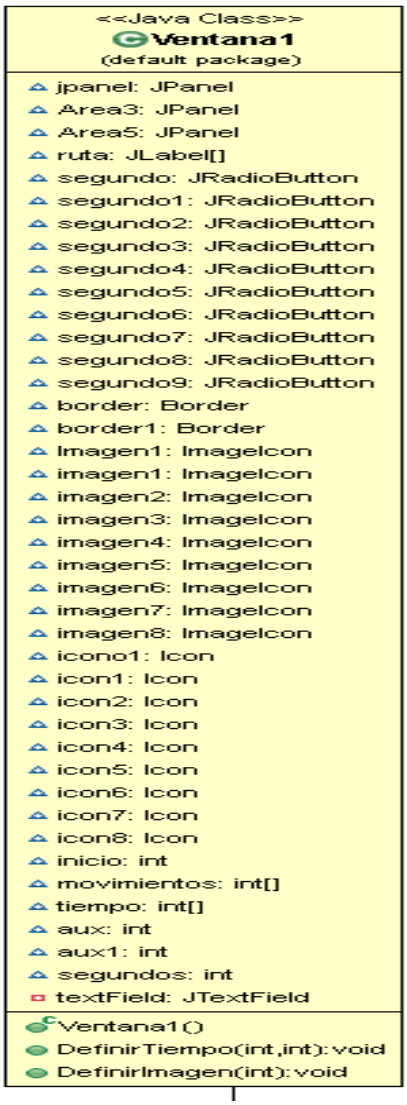


Figura 27: Package Ventana 1

En este package (ver figura 27) es el encargado de la generación de la interfaz gráfica del modo automático que es la herramienta DSL, este package tiene algunos transformaciones propias que se va a definir a continuación.

Para la selección de cada uno de los movimientos se usa la siguiente regla de transformación definida por el diagrama de la figura 28.

Para la generación de cada una de las pausas generadas en segundos por el Delay se debe emplear la siguiente formula de la Ecuación 1.

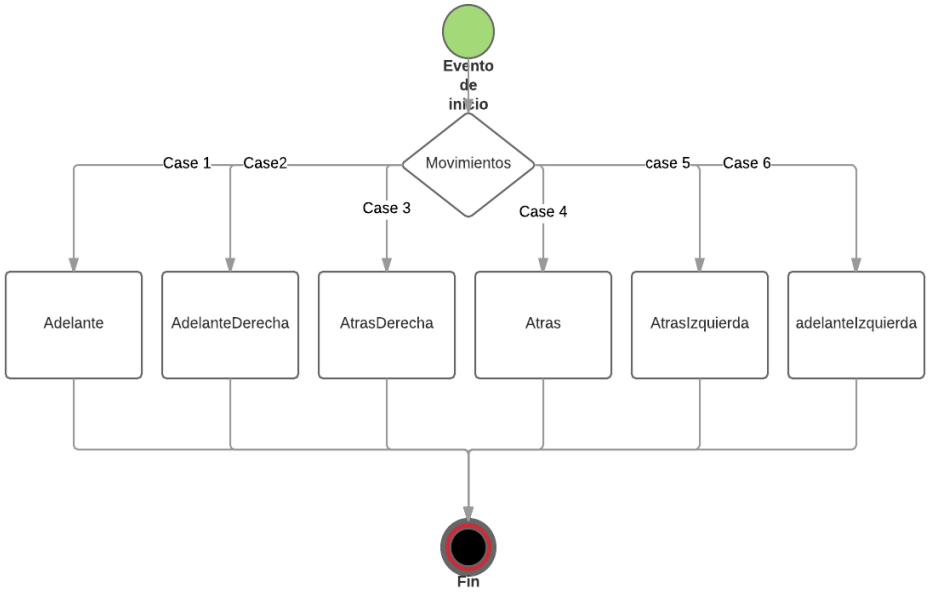


Figura 28: Diagrama de flujo selección de movimiento

Primero se debe saber que cada dirección y cada pausa que se establece en la ruta son guardados independientemente en dos vectores, el uno de movimientos y el otro de tiempos, los cuales con ayuda de un ciclo for que va de 1 a 12 que son el número máximo de instrucciones seguidas que se pueden generar en la plataforma, el ciclo for va recorriendo cada uno de los vectores para poder establecer el que código ejecutar.

```
public void DefinirTiempo(int a, int t) {
        movimientos[inicio] = a;
        tiempo[inicio] = t;
}
```

Estas líneas de código muestran cómo se almacenan cada uno de los movimientos y de las pausas con ayuda de la función DefinirTiempos, posteriormente se debe definir la función que se encarga de la generación del archivo .ino que va a ser grabado en el Arduino.

```
String ubicacion = "";
try {
        if (jF1.showSaveDialog(null) == jF1.APPROVE_OPTION) {
        ubicacion = jF1.getSelectedFile().getAbsolutePath();
    }
        } catch (Exception ex) {
                ex.printStackTrace();
        }
        FileWriter escritura = null; // la extension al archivo
        try {
        File salida = new File(ubicacion);
        BufferedWriter bw = new BufferedWriter(new FileWriter(salida));
        PrintWriter salArch = new PrintWriter(bw);
        salArch.print("""¨);
```

Con estas líneas de código se genera la opción para guarda el archivo .ino dependiendo en donde el usuario desee guardar con la ubicación que el usuario decida, y con la función PrintWriter se decide que líneas se van a guardar en el archivo .ino.

Para la generación de cada uno de los tiempos del ¨Delay¨ se emplea el siguiente código con la regla de transformación definida anteriormente.

```
salArch.print("delay(" + Ventana1.this.tiempo[i] * 50000 + ");");
salArch.println();
```

Este código muestra con ayuda de la Ecuacion 1, como se genera cada uno de los retardos empleados en el diseño de la ruta a efectuarse.

Para la generación de cada uno de los movimientos se basa en la figura 28 y la regla de transformación que se indicó anteriormente.

```
for (int i = 0; i < 12; i++) {

        switch (Ventana1.this.movimientos[i]) {
        case 1:
```

```
                    salArch.print("adelante();");
                    salArch.println();
                    salArch.print("delay(50000);");
                    salArch.println();
                    break;
                case 2:
                    salArch.print("adelanteDerecha();");
                    salArch.println();
                    salArch.print("delay(50000);");
                    salArch.println();
                    break;
                case 3:
                    salArch.print("atrasDerecha();");
                    salArch.println();
                    salArch.print("delay(50000);");
                    salArch.println();
                    break;
                case 4:
                    salArch.print("atras();");
                    salArch.println();
                    salArch.print("delay(50000);");
                    salArch.println();
                    break;
                case 5:
                    salArch.print("atrasIzquierda();");
                    salArch.println();
                    salArch.print("delay(50000);");
                    salArch.println();
                    break;
                case 6:
                    salArch.print("adelanteIzquierda();");
                    salArch.println();
                    salArch.print("delay(50000);");
                    salArch.println();
                    break;

            }
        }
```

Con estas líneas de código se elige cada una de las rutas con guardadas en el vector de movimientos y con ayuda de un seleccionar de caso condicional se puede realizar la configuración de la ruta automática que se va a definir para el correcto funcionamiento de la interfaz.

CAPITULO V

PRUEBAS DE LA HERRAMIENTA DSL.

5.1 Definición del escenario de pruebas

Para la definición del escenario de pruebas se deben tomar en cuenta algunas características de funcionamiento del drone para poder definir un escenario adecuado y propicio para la ejecución de la herramienta DSL creada.

Visto lo citado anteriormente y con ayuda del datasheet del drone específico para el diseño de la herramienta, se procede a definir el escenario de la aplicación. Visto todo esto para definir el escenario se sabe que la cobertura máxima del dron con línea de vista es de 50m y que su batería cargada al 100 por ciento solo puede funcionar por 7 min seguidos se define el escenario 1 que se puede ver en la figura 29 que son de las siguientes dimensiones 28 metros de largo por 15 metros de ancho que este es el tamaño de una cancha de básquet.

Se decidió utilizar una cancha de básquet en vista que presenta las medidas necesarias para que se tenga línea de vista en todos los puntos y se encuentra debidamente delineada para poder trazar rutas para la aplicación de la herramienta en la figura 30 se puede ver una vista aérea del escenario de pruebas.

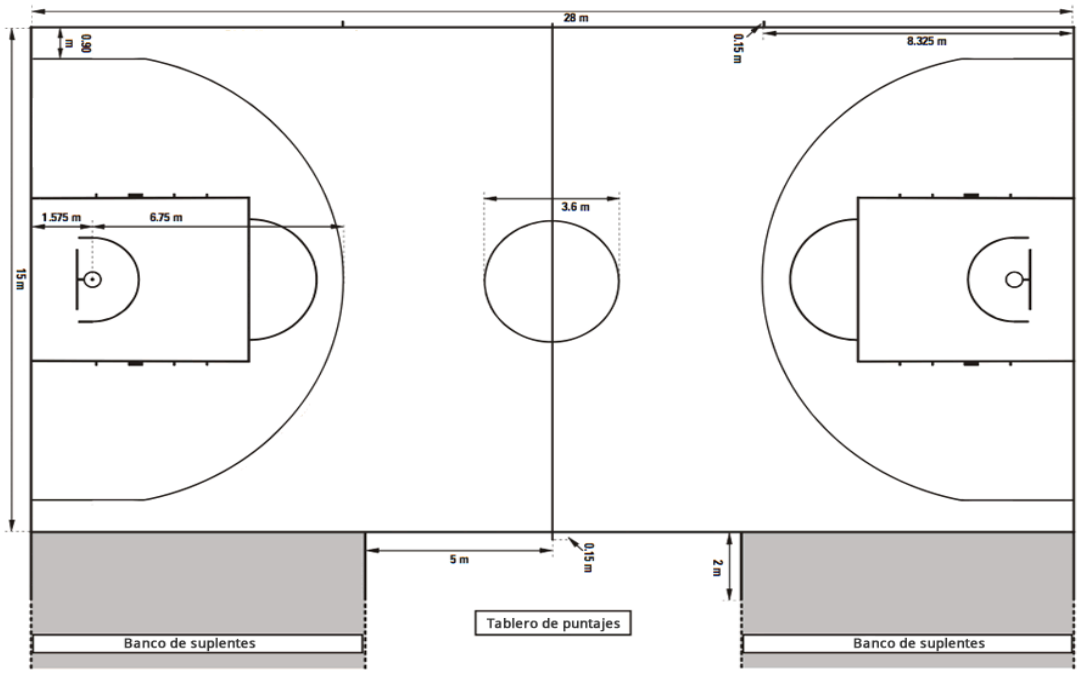


Figura 29: Definición de escenario de pruebas

Figura 30: Vista aérea Escenario de prueba

5.2 Pruebas realizadas.

En este escenario se procede a realizar las siguientes pruebas:

- Pruebas de generación de ruta
- Pruebas de usabilidad

5.2.1. Pruebas de generación rutas:

En estas pruebas se desarrollaron generación de varias rutas para poder todas las rutas necesarias para no tener errores de dirección las rutas probadas fueron las mostradas en las figuras 31 y 32

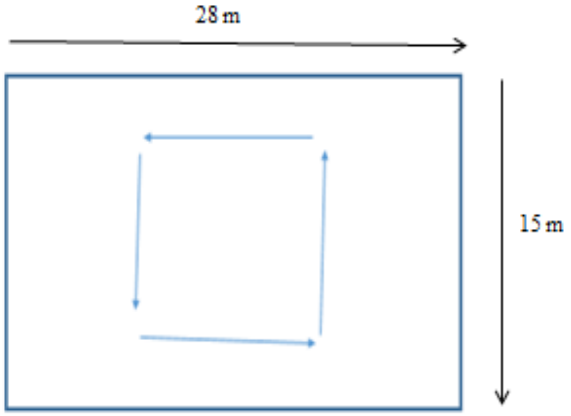


Figura 31: Primera ruta de prueba

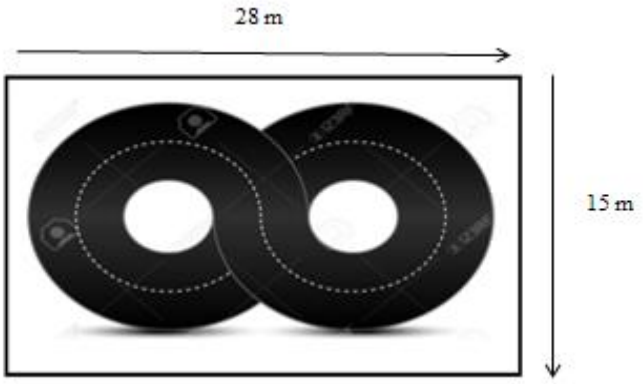


Figura 32: Segunda ruta de prueba

En el caso de la ruta de la figura 31, la ruta generada en la plataforma es la de la figura 33, donde se muestra que se puede generar con facilidad dicha ruta

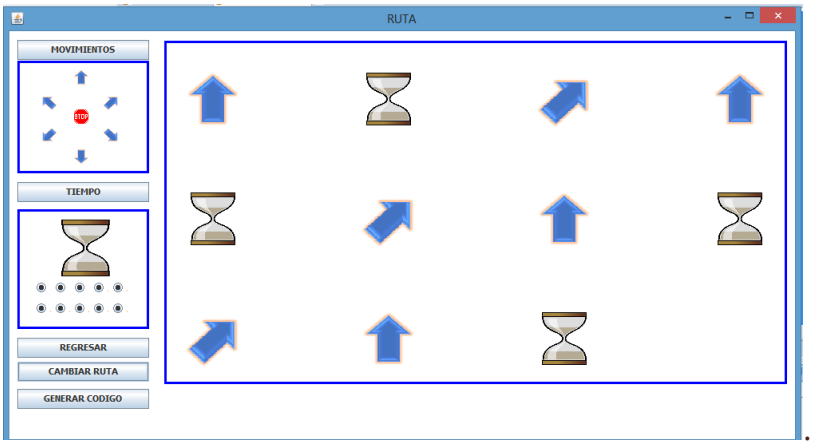


Figura 33: Generación ruta 1.

Como se puede observar la plataforma muestra a través de una interfaz gráfica en el área 1 cada uno de los movimientos y pausas definidos en la ruta a implementar, en la tabla 2 se muestra como se configura cada una de las posiciones de los vectores de movimientos y de tiempos para la generación de la ruta mediantes las transformaciones definidas anterior mente.

Tabla 2.

Datos de configuración de vectores de movimientos y tiempo.

Posición	Vector movimiento (Posición)	Vector tiempo (Posición)
1	1	1
2	1	10
3	2	1
4	1	1
5	1	10
6	2	1
7	1	1
8	1	10
		Continua →

9	2	1
10	1	1
11	1	10

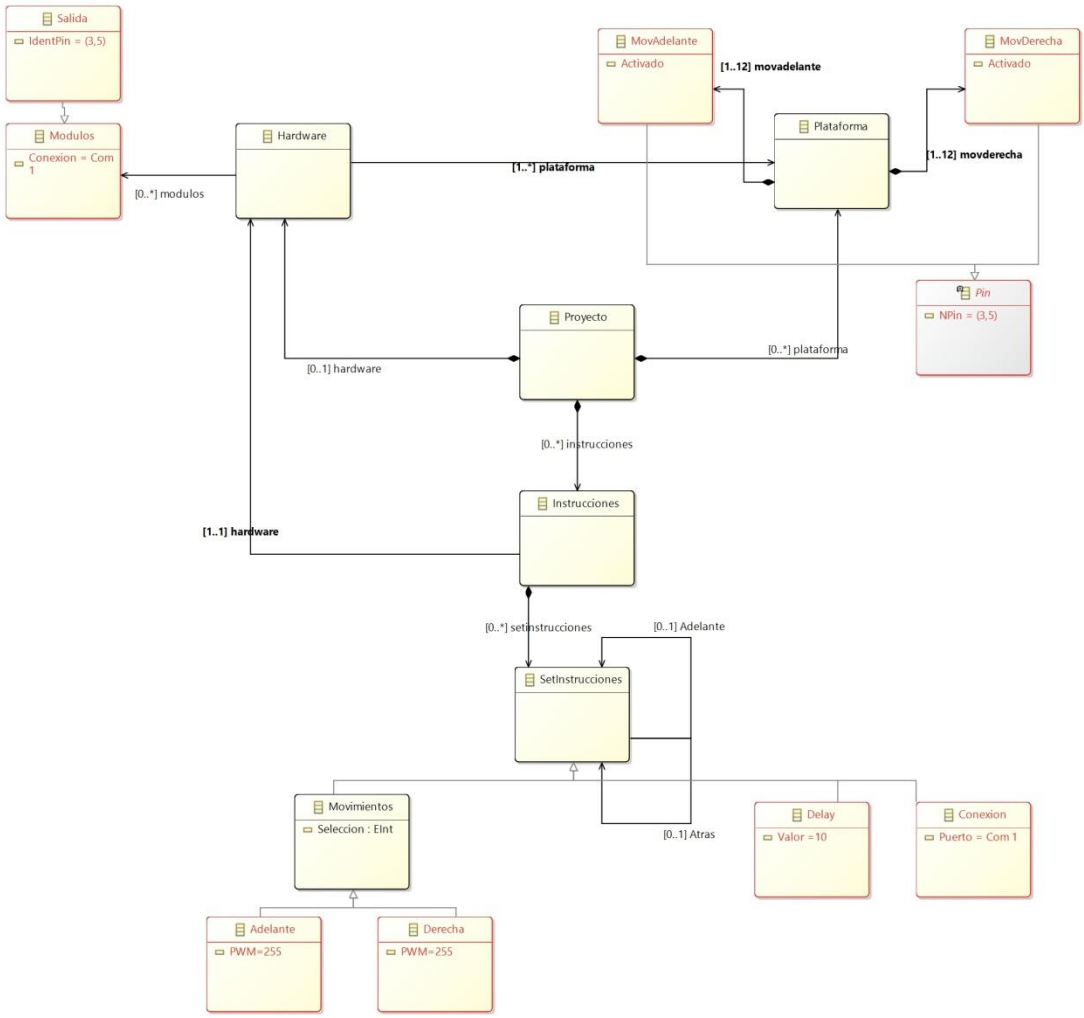


Figura 34: UML Generado por la aplicación

Como se muestra en la figura 34, el diagrama UML generado por la aplicación cumple con los requerimientos de la arquitectura propuesta por la herramienta.

El código generado para ser implementado en el Arduino es el siguiente:

Como se puede observar en el código primero se despliegan cada una de las clases que definen cada uno de los movimientos a implementarse dependiente el orden de

ingreso de cada instrucción en el vector de movimientos, también se genera el PWM adecuado para cada uno de los movimientos y su respectivo Pin de salida.

```
void adelante() {
  analogWrite(3, 0);
  analogWrite(5, 155);
}
```

Función que define movimiento adelante como muestra el diagrama UML

```
void Derecha() {
  analogWrite(3, 0);
  analogWrite(5, 255);
}
```

Función que define movimiento derecha como muestra el diagrama UML

```
void Delay(int valor) {
   Delay = valor * 50000;
}
```

Función de generación de tiempos

```
void setup() {
  // MOTOR 1
  pinMode(3, OUTPUT);
  //MOTOR 2
  pinMode(5, OUTPUT);
}
```

Función que los pines de salida para cada instrucción, corresponde al package plataforma

```
void loop() {
adelante();
delay(1);
delay(10);
adelanteDerecha();
delay(1);
adelante();
delay(1);
delay(10);
adelanteDerecha();
delay(1);
adelante();
delay(1);
delay(10);
adelanteDerecha();
delay(1);
adelante();
```

Orden general de ejecución de ruta dependiendo el orden de ingreso de cada una de las direcciones y tiempos

```
delay(1);
delay(10);
}
```

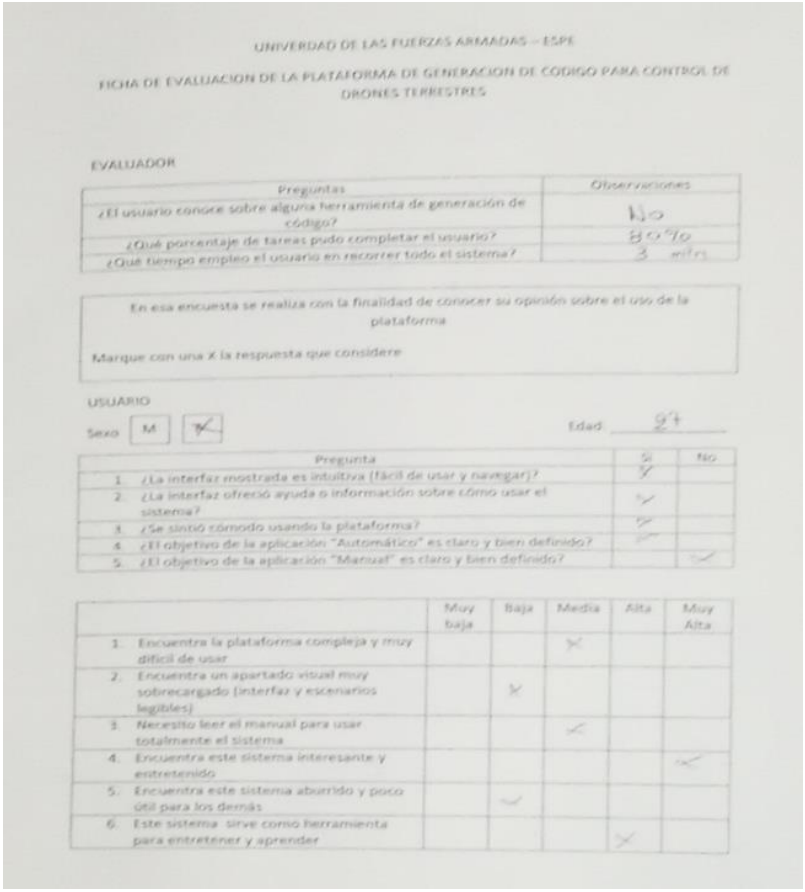
Con este código y con la tabla 2 se puede comprobar que la plataforma está generando correctamente cada uno de los códigos que se necesitan para la ejecución de la ruta en base al diagrama UML de la figura 34.

5.2.2. Pruebas de Usabilidad.

Para conocer los resultados sobre el impacto y usabilidad de la plataforma generada se realiza una ficha de evaluación que se aplica al usuario inmediatamente después de que el Usuario haya interactuado con la plataforma. Para la elaboración de esta encuesta se toma en cuenta la norma ISO / IEC 9126-4, el manual de encuestas de satisfacción al consumidor y la heurística de NIELSE (ver figura 35).

Los aspectos a tomar en cuenta en la plataforma son:

- Lenguaje entendible y legible para el usuario
- Facilidad de usabilidad del usuario
- Visualización clara y legible de las interfaces
- Eficiencia de uso del sistema
- Métricas de eficiencia



UNIVERDAD DE LAS FUERZAS ARMADAS – ESPE

FICHA DE EVALUACION DE LA PLATAFORMA DE GENERACION DE CODIGO PARA CONTROL DE DRONES TERRESTRES

EVALUADOR

Preguntas	Observaciones
¿El usuario conoce sobre alguna herramienta de generación de código?	No
¿Qué porcentaje de tareas pudo completar el usuario?	80%
¿Qué tiempo empleo el usuario en recorrer todo el sistema?	3 mins

En esa encuesta se realiza con la finalidad de conocer su opinión sobre el uso de la plataforma

Marque con una X la respuesta que considere

USUARIO

Sexo M [X] Edad 21

Pregunta	Si	No
1. ¿La interfaz mostrada es intuitiva (fácil de usar y navegar)?	X	
2. ¿La interfaz ofreció ayuda o información sobre cómo usar el sistema?	X	
3. ¿Se sintió cómodo usando la plataforma?	X	
4. ¿El objetivo de la aplicación "Automático" es claro y bien definido?	X	
5. ¿El objetivo de la aplicación "Manual" es claro y bien definido?		✓

	Muy baja	Baja	Media	Alta	Muy Alta
1. Encuentra la plataforma compleja y muy difícil de usar			X		
2. Encuentra un apartado visual muy sobrecargado (interfaz y escenarios legibles)		X			
3. Necesito leer el manual para usar totalmente el sistema			✓		
4. Encuentra este sistema interesante y entretenido					✓
5. Encuentra este sistema aburrido y poco útil para los demás		✓			
6. Este sistema sirve como herramienta para entretener y aprender				X	

Figura 35: Ficha de evaluación

5.3 Análisis de resultados

La muestra empleada para realizar las pruebas de usabilidad es de una población variada de edad y género, con y sin estudios universitarios con conocimientos de tecnología, ya que como se había indicado el sistema está enfocado para jóvenes y adultos de cualquier área de conocimientos, sin embargo la plataforma no está exenta de ser usada por niños quienes aprenden mediante el juego y se relacionan de manera favorable con la tecnología. En la tabla 3 se puede observar los usuarios que utilizaron el sistema.

Tabla 3.

Muestra utilizada en la evaluación

Genero \ Edad	21	22	23	24	25	26	28	30	32	Total general
Masculino	3	1	3	2	2	2	2	2	2	19
Femenino	1	0	2	1	1	3	1	1	1	11
Total general	4	1	5	3	3	5	3	3	3	30

Al realizar la tabulación de cada uno de los datos de la encuesta realizada (Anexo 1) se puede analizar una de las métricas de usabilidad, la cual mostrara que el porcentaje de personas que conocen acerca de plataformas de generación de código automático para manejo de drones terrestres (ver figura 36) se llegó a la respuesta de que solo 1 persona que corresponde al 3% conoce una herramienta similar mientras que 29 personas que responden al 97 % encuestadas no conoce de la existencia de algo similar y esto les atrae mucho la atención.

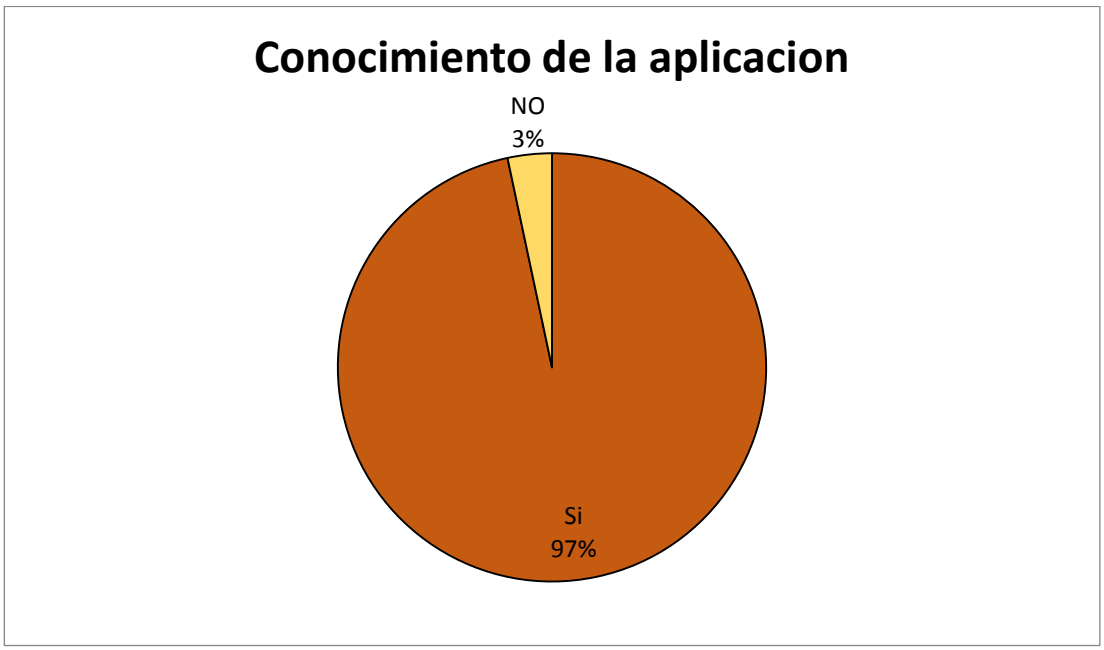


Figura 36: Porcentaje de personas que conoce de una plataforma similar

Se ha determinado un tiempo máximo de 5 minutos para que el usuario manipule la plataforma, en donde se pueden obtener dos graficas que son, la una del tiempo de manipulación del sistema y la otra el porcentaje de navegación del sistema completado en donde como se indica en la figura 37 en donde se muestra que la media de tiempo está entre 3.5 minutos y la de porcentaje del sistema es del 90%.

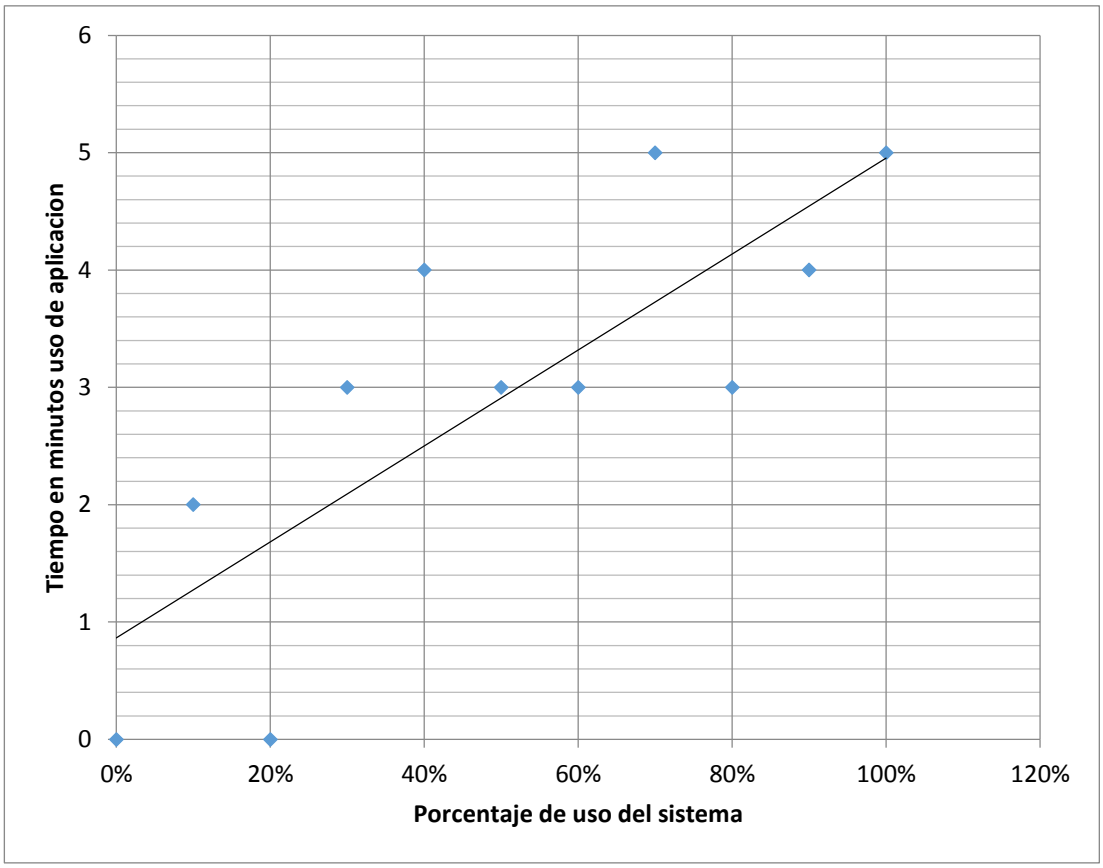


Figura 37: Porcentaje usado del sistema vs Tiempo empleado

La estimación del sistema se estructura de dos formas: conocer la aceptación del usuario ante la plataforma generada y la segunda y más importante la calidad y usabilidad de la interfaz y de los servicios que esta brinda al usuario con vista a esto la respuesta a la primera cuestión que se planteó con preguntas de si o no me da los resultados descritos en la figura 38.

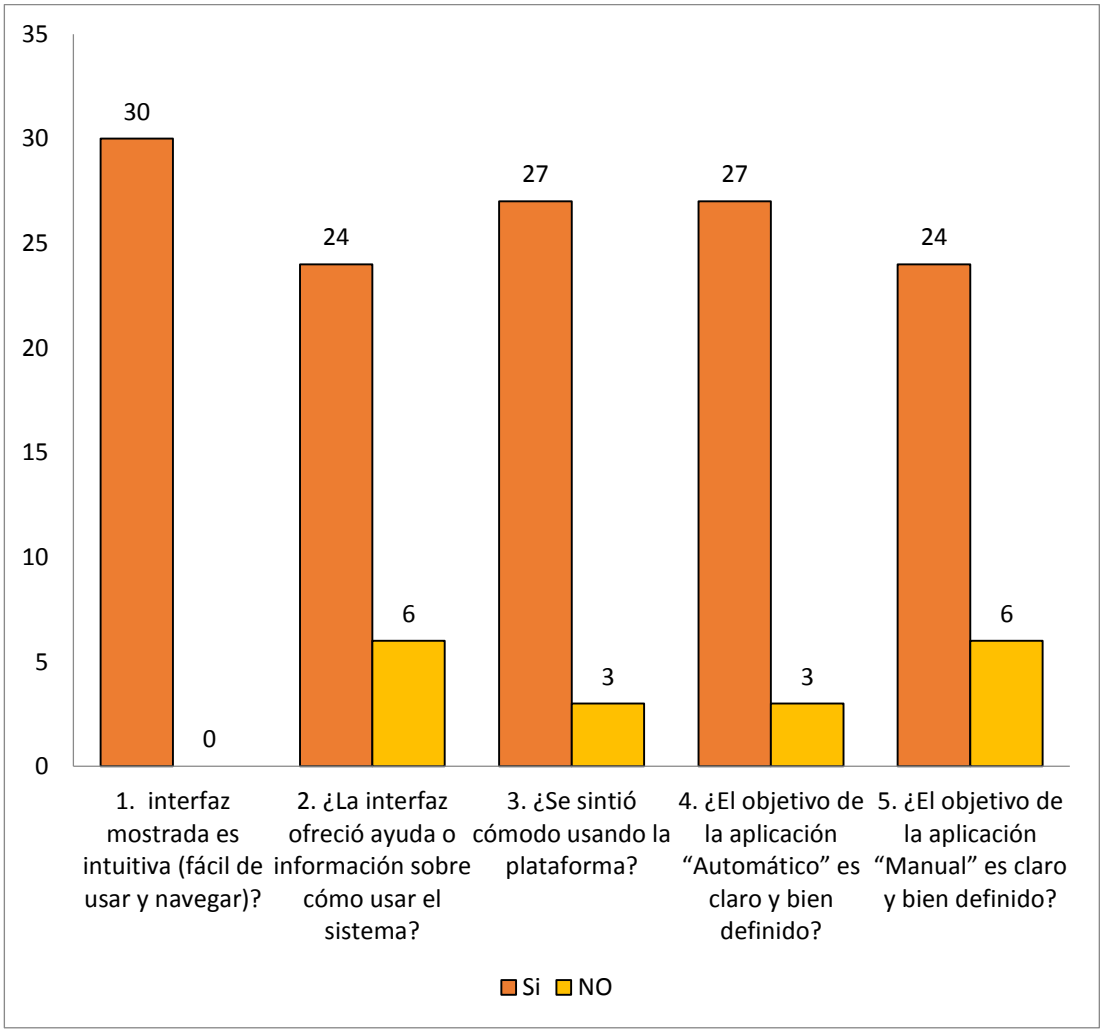


Figura 38: Resultados de los usuarios sobre la interfaz de la plataforma

Con los datos obtenidos en las figura 38 se puede llegar a la conclusión de que la Figura presenta una interfaz amigable con el usuario y que es de fácil utilización, en cuanto a las preguntas 2 y 5 que muestran un índice mayor de NO que las demás se ofrece la solución de colocar un menú de ayuda más favorable al usuario para que pueda desenvolverse con mayor facilidad en la interfaz.

En cuanto a la calidad y usabilidad del sistema se observan los resultados mostrados en la tabla 4 a cada una de las preguntas.

Tabla 4.

Número de usuarios evaluando nivel de satisfacción de la plataforma. 1-6

	Muy baja	Baja	Media	Alta	Muy Alta
1. Encuentra la plataforma compleja y muy difícil de usar	7	16	4	2	1
2. Encuentra un apartado visual muy sobrecargado (interfaz y escenarios legibles)	6	13	5	4	2
3. Necesito leer el manual para usar totalmente el sistema	4	8	13 50 %	2	3
4. Encuentra este sistema interesante y entretenido	2	2	2	6	18
5. Encuentra este sistema aburrido y poco útil para los demás	14 70 %	7	3	2	2
6. Este sistema sirve como herramienta para entretener y aprender	2	3	1	8	16

En donde se encuentra un índice muy bajo única mente en la pregunta 3 con un 50% que dice que necesita de leer un manual para entender la aplicación mientras que con respecto a la pregunta 5 que índica si la interfaz es óptima para el usuario y lo atrae se presenta un 70% de aceptación que es muy bueno en la figura 39 se muestra la gráfica con respecto a la pregunta 3, mientras que en la figura 40 se muestra el nivel de aceptación del usuario con respecto a la plataforma que son las preguntas 4-6.

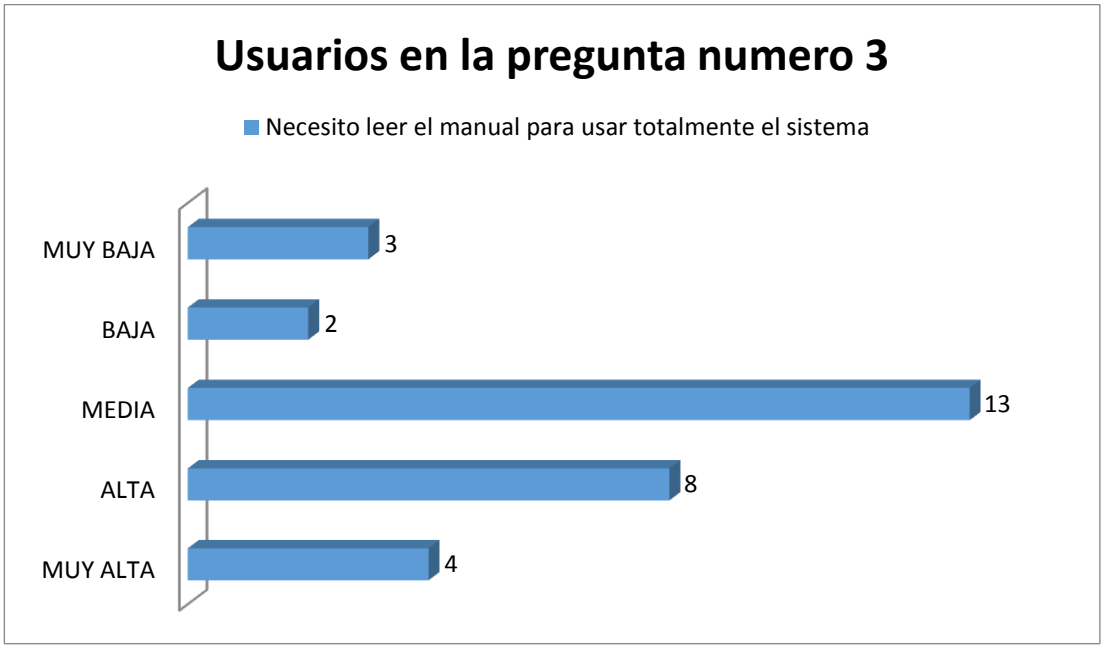


Figura 39: Tabulación sobre ayuda para utilización de la plataforma

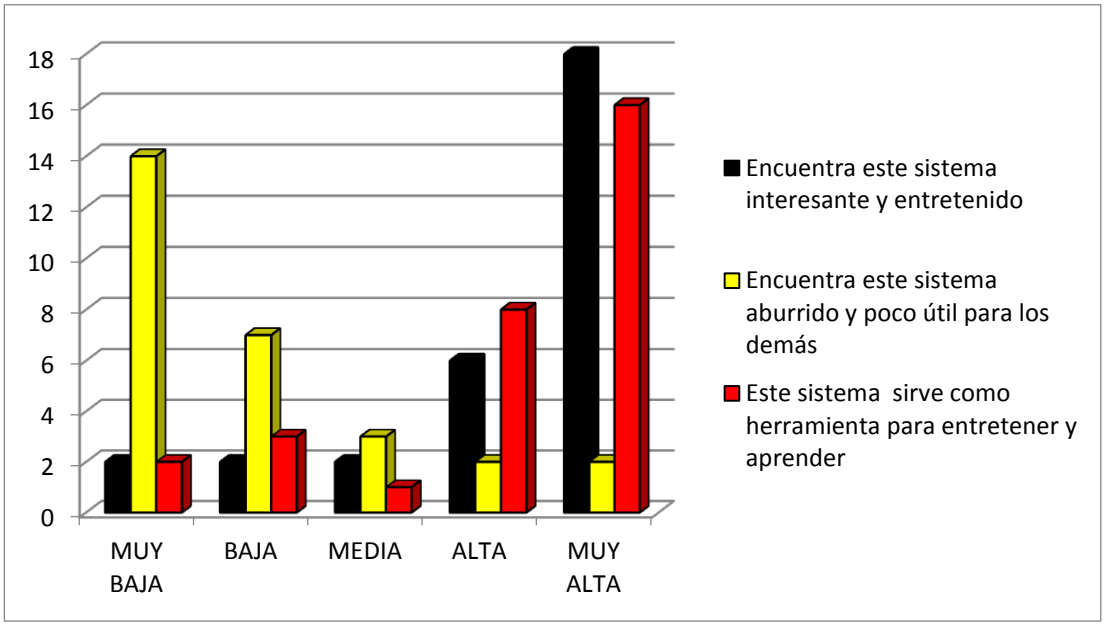


Figura 40: Evaluación de satisfacción de la plataforma preguntas 4-6

Con el resultado de todas estas preguntas se puede decir que la plataforma cumple con las funciones de usabilidad y que brinda una interacción fácil y amigable con el usuario sin importar la edad y sexo del usuario.

Capítulo VI

CONCLUSIONES Y RECOMENDACIONES

6.1 Conclusiones

- Este trabajo se presenta como una solución a la programación y generación de códigos para el manejo de drones o vehículos no tripulados, mediante una tarjeta controladora de drones ¨Arduino¨ y con ayuda de Lenguajes de Programación, debido a su legibilidad y nivel de escritura, y también a IDE's específicos por el nivel de accesibilidad y abstracción que suponen estos.
- Se ha logrado desarrollar un Entorno de Desarrollo con el que se busca reducir la brecha de entrada propia de Lenguajes de Programación más complejos, pero no esconder la complejidad de la lógica de Programación, mediante la implementación de modelos que los cuales tienen un gran nivel de abstracción.
- Se desarrolló una arquitectura con la ayuda del uso de los modelos para poder generar una plataforma para generación de códigos para control a distancia de Drones que presenta una interfaz visual y que con ayuda de pruebas de funcionamiento se comprobó que puede generar la mayoría de rutas que el usuario desee.
- Se logró mediante la investigación acoplar los modelos generados con lenguajes de programación a elementos físicos como en este caso mediante la conformación de una plataforma de modelado y con lenguajes de programación bien definidos.
- Se logró validar la interfaz gráfica descrita en esta herramienta de generación de código mediante pruebas de usabilidad realizadas a grupos de personas de diferentes edades y sexos los cuales arrojaron un 70% de aceptación de la interfaz que dice que es visualmente atractiva.
- Con respecto a la parte de innovación mediante encuestas se logró conocer que el 97% de la población no conoce o no ha visto una plataforma de generación de códigos para control a distancia de drones, en vista a esto se concluye que esta

aplicación puede ser muy útil para familiarizar más a los usuarios con este tipo de tecnologías.

- Se definió un set de instrucciones mediante modelos para la optimización de recursos físicos como de software de la tarjeta de control “Arduino” para lograr una interfaz funcional con la plataforma de generación de código diseñada en java, esto permite un visualizar de una manera amigable cada uno de los bloques.

6.2 Recomendaciones

- Se recomienda que para la implementación de la plataforma de generación de código para manejo de drones conseguir drones que presenten un control físico y no por software debido a que esto ayuda en la instalación.
- Se recomienda para la utilización de la comunicación serial de la plataforma con Arduino comprobar el funcionamiento de los cables de conexión y de los buses de datos que salen de Arduino hacia el Drone.
- En cuanto a la generación de rutas se recomienda al usuario no intentar generar una ruta de más de 12 instrucciones seguidas en vista que la plataforma solo puede generar un máximo de 12 instrucciones.

6.3 Trabajos Futuros

En esta aplicación se han encontrado algunas limitaciones que pueden ser resueltas en trabajos futuros, que además aporten otras funcionalidades, permitiendo así el crecimiento de la herramienta. A continuación, se valora los principales trabajos futuros a realizar:

- Diseño de un simulador que permita al usuario de la plataforma poder agregar una referencia del terreno en que se va a desarrollar la ruta, tanto de imperfecciones de terreno como de conexión.

- Proveer al diseñador la posibilidad de instalar la herramienta desarrollada a través de un instalador guiado sin necesidad del uso de Eclipse o cualquier otra herramienta java.
- Habilitar el uso de la plataforma de generación de código para drones creada a través de dispositivos móviles, o también a través de navegadores web.

BIBLIOGRAFÍA

- Canelón, R. (2010). Un Proceso para la Ingeniería del Dominio Basado en Calidad de Software. Una aplicación al dominio del aprendizaje móvil sensible al contexto (Doctoral dissertation, Tesis Doctoral. Universidad Central de Venezuela. Caracas).
- Colque, D. C., & Valdivia, R. P. (2006). Integración de tecnologías en una plataforma j2ee dirigida por modelos/technologies integration in a model driven j2ee platform. Ingeniare: Revista Chilena de Ingenieria, 14(3), 265.
- Enciso Jiménez, J. (2016). Desarrollo de un sistema y modelo de control por radiofrecuencia para manejo de drones.
- García Díaz, V., & Cueva Lovelle, J. M. (2010). Ingeniería Dirigida por Modelos, Ed.
- García, P. A. G., Marín, C. E. M., Lovelle, j. M. C., & Martínez, O. S. (2011). Aplicación de ingeniería dirigida por modelos (MDA), para la construcción de una herramienta de modelado de dominio especifico (DSM) y la creación de módulos en sistemas de gestión de aprendizaje (LMS) independientes de la plataforma. Dyna, 78(169), 43-52.
- Hadad, G. D. S. (2010). Uso de Escenarios en la Derivación de Software. In XII Workshop de Investigadores en Ciencias de la Computación.
- Herrera, J., Losavio, F., & Ordaz, O. (2015). Ingeniería del Dominio con el Estándar ISO/IEC 26550 para Líneas de Productos de Software considerando la faceta calidad. Memorias Conferencia Nacional de Computación. Informática y Sistemas, 107-118.
- López, E., González, M., López, M., & Iduñate, E. L. (2006). Proceso de desarrollo de software mediante herramientas MDA. Revista Iberoamericana de Sistemas, Cibernética e Informática, 3(2), 6-10.
- Marjan Mernik, Jan Heering, and Anthony M. Sloane (2005). When and how to develop domain-specific languages. ACM Computing Surveys.
- Montenegro Marín, C. E., García, G., Alonso, P., Cueva Lovelle, J. U. A. N., & Sanjuán Martínez, Ó. (2011). Application of model-driven engineering (MDA) for the construction of a tool for domain-specific modeling (DSM) and the creation of

modules in learning management systems (LMS) platform independent. Dyna, 78(169), 43-52.

- Passuni Córdova, J. (2017). Diseño y programación de add-on para el software de control y monitoreo" Mission Planner" que permita visualizar el área fotografiada de cada imagen.
- Pons, C., Giandini, R. S., & Pérez, G. (2010). Desarrollo de software dirigido por modelos.
- Pressman, R. S., & Troya, J. M. (1988). Ingeniería del software.
- Quintero, J. B., & Anaya, R. (2007). MDA y el papel de los modelos en el proceso de desarrollo de software. Revista EIA, (8), 131-146.
- Passuni Córdova, J. (2017). Diseño y programación de add-on para el software de control y monitoreo" Mission Planner" que permita visualizar el área fotografiada de cada imagen.
- Pons, Claudia. (2010). Desarrollo de software dirigido por modelos: conceptos teóricos y su aplicación práctica, La Plata: Universidad Nacional de La Plata.