UNIVERSIDAD CATÓLICA DE LA SANTÍSIMA CONCEPCIÓN

Facultad de Ingeniería

Ingeniería Civil Informática

GENERADOR DE APLICACIONES MÓVILES EN EL DOMINIO DE ADMINISTRACIÓN DE EMERGENCIAS

BRAULIO ALBERTO QUIERO HERNÁNDEZ

INFORME DE PROYECTO DE TÍTULO PARA OPTAR AL TITULO DE

INGENIERO CIVIL INFORMÁTICO

Profesor Guía

Pedro Osvaldo Rossel Cid

Concepción, abril 2017

Resumen

En las siguientes páginas se detalla el desarrollo del proyecto Generador de Aplicaciones Móviles en el Dominio de la Administración de Emergencias, cuyo objetivo es construir aplicaciones móviles para el dominio de Administración de emergencias utilizando un software que, en base a componentes reutilizables creados en un proyecto de título anterior, sea capaz de generar estas aplicaciones.

En base a lo anterior cobra importancia el reuso de software que consiste en desarrollar soluciones para problemas similares reutilizando funciones o procedimientos, lo que reduce el costo del desarrollo del software y la mantención entre otras características y el concepto de línea de productos de software que consiste en un conjunto de sistemas que comparte un conjunto de características comunes que utiliza un enfoque orientado al reuso.

Las diferentes funciones que realiza cada persona en la administración de una emergencia, donde se desencadena un conjunto de procedimientos tales como la verificación de la emergencia, la activación de la alarma, las comunicaciones por radio y el despacho del personal, implica la coordinación de las distintas instituciones y del personal de cada una de ellas, lo que justifica la creación de aplicaciones móviles que sirvan de apoyo a la coordinación de estos equipos de emergencia.

Abstract

In the following pages we can see the development of the Mobile Applications Generator project in the Domain of Emergency Management, with the aim of building mobile applications for the Emergency Management domain using software that in the base of reusable components, created in a project of previous title, is able to generate these applications.

Based on the above, reusable software, which consists of developing solutions to the problems of reuse of procedures, reduces the cost of software development and maintenance among other characteristics and the concept of line of software products consisting of a set of Systems that share a set of common features that use a reuse-oriented approach.

Due to the different functions that each person performs in the management of an emergency, where a set of procedures are triggered, such as emergency verification, alarm activation, radio communications and dispatch of personnel which implies the coordination of the different institutions and the staff of each one of them is that the creation of applications that support the coordination of these emergency teams is justified.

Dedicatoria y Agradecimientos

Agradezco a mis padres Pedro y Griselda por brindarme su apoyo, disciplina, paciencia y sobre todo su amor durante todo este tiempo. Gracias por comprender el hecho de dejar de lado tantas reuniones familiares para sacar adelante este proyecto.

A mis hermanas Katherine y Karol. Gracias por siempre escucharme hablar sobre programas, requisitos, modelos y tantas otras cosas que no comprendían solo para darme ánimo.

A mis amigos que siempre creyeron en mis capacidades, aun en los momentos en que yo dudaba de ellas. Gracias por todo su apoyo.

A mis profesores Pedro, Claudia y Marcia gracias por su apoyo y sobre todo su paciencia hasta el último momento.

A Roxana, secretaria de mi carrera, por siempre estar disponible para resolver mis dudas, apoyarme en todo lo que necesité y tratarme con cordialidad y respeto.

A Elson Díaz, gracias por brindarme su confianza y apoyo desde mis primeros años en la Universidad, atender mis llamadas, ayudar a resolver mis dudas.

A Víctor Valdés, gracias por tener tanta confianza en mí y alentarme todos estos años con esas largas conversaciones cuando salía de la Universidad.

Y mayormente agradecer a Dios por estar conmigo en todo momento, salir corriendo a mi encuentro, darme una nueva oportunidad y un motivo para vivir.

Índice de Contenidos

Índice de Ilustraciones

Índice de Tablas

Nomenclaturas y Abreviaciones

1. ADT: siglas de Android Developer Tools; es un plugin para Eclipse que provee una interfaz gráfica para acceder a muchas de las herramientas del SDK de Android de la línea de comandos (Android Developers, 2014).
2. Algoritmo: conjunto ordenado y finito de operaciones que permite hallar la solución de un problema (Muñoz Caro, Niño Ramos, & Vizcaino Barceló, 2003).
3. Android: sistema operativo de código abierto bajo licencia Apache, perteneciente a Google desde julio de 2006 (Nolasco Valenzuela, 2014).
4. Ant: siglas de "Another Buil Tools"; es una herramienta para la construcción de aplicaciones Java mediante la línea de comandos del sistema operativo (Bailliez et al., 2015).
5. API: siglas de Application Program Interface que en español quiere decir interfaz de programación de aplicaciones (Villate, 2002).
6. App: se usa como abreviación de "application" para referirse a aplicaciones creadas para dispositivos móviles (Cambridge Dictionary, 2016).
7. Artefacto: término general para cualquier información creada, producida, cambiada o utilizada por los trabajadores en el desarrollo del sistema, ejemplo de artefacto son los diagramas de caso de uso, glosario, etc (Larman, 1999).
8. Componente: pieza de software funcional que es liberada independientemente de otras y que proporciona acceso a sus servicios a través de sus interfaces(Montilva, 2006).
9. Dispositivo Móvil: Un dispositivo móvil se puede definir como un aparato de pequeño tamaño, con algunas capacidades de procesamiento, con conexión permanente o intermitente a una red, con memoria limitada, que ha sido diseñado específicamente para una función, pero que puede llevar a cabo otras funciones más generales (Alonso, Artime, & Rodríguez, 2011).

10. Java: lenguaje de programación potente se encuentra hoy en día en 3 mil millones de teléfonos móviles (Java, 2014) y en distintos dispositivos como blue-ray, televisores, etc, funciona en múltiples dispositivos instalando la máquina virtual de Java en el sistema operativo que desea ejecutar Java.
11. LPS: siglas de línea de producto de software; se define como un conjunto de sistemas, que comparten un conjunto de características comunes, las que satisfacen las necesidades específicas de un dominio particular y se desarrollan a partir de un sistema común de activos base (Clements & Northrop, 2002).
12. PATH: representa una variable de entorno del Sistema operativo que contiene el conjunto de directorios utilizados para buscar archivos ejecutables (The Open Group, 2008).
13. Plugin: complemento que puede ser agregado a un programa para añadir funcionalidades adicionales.
14. POO: Puede referirse a Programación Orientada a Objetos o Paradigma Orientado a Objetos; en este informe se utiliza como la primera sigla (Muñoz Caro et al., 2003).
15. Reuso: Se refiere a reutilizar un programa o componente de software y buscando funciones o procedimientos que pueden reutilizarse para la generación de un nuevo programa (Frakes, 2005).
16. Script: Se refiere a un archivo que contiene comandos (Villate, 2002).
17. SDK: Siglas de Software development kit; se utiliza para referirse a un conjunto de herramientas que permite el desarrollo de un software (Christensson, 2010).
18. UML: siglas de Unified Modeling Lenguage; es un lenguaje de modelado cuyo principal objetivo es proporcionar a expertos en el área del desarrollo de software herramientas para el análisis, diseño e implementación de sistemas basados en software (Group, 2010).
19. UP: siglas de Unified Process; se refiere a un método de desarrollo que consiste en 4 etapas o fases iterativas que van desde el inicio, la elaboración, construcción para finalizar con la etapa de transición (Sommerville, 2005).

20. XML: siglas de eXtensible Markup Language o Lenguaje de Marcado extensible; es un lenguaje de texto simple y muy flexible, describe una clase de objetos de datos llamados documentos XML (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 1997), se caracteriza por el uso de etiquetas (par nombre, valor).
21. XPatch: Lenguaje construido para el direccionamiento de partes de un documento XML; sirve para identificar elementos en un documento XML mediante la ruta válida del elemento (XSL Working Group & XML Linking Working Group, 2016).

1. Introducción

1.1. Presentación del tema

Este proyecto propone la creación de un software con la capacidad de crear aplicaciones móviles a partir de componentes bases realizados en otro proyecto de título (Ormeño, 2015), lo cual se justifica al ser una investigación en curso que se enmarca en el dominio de administración de emergencias y pretende la adaptación del software o aplicación al usuario, a través de la creación de una familia de aplicaciones las cuales son creadas con estos componentes basados en un modelo de características, el que muestra los aspectos relacionados a la gestión de la emergencia y los aspectos de colaboración (Rossel, Herskovic, & Ormeño, 2016).

Los componentes bases fueron construidos para la realización de aplicaciones en dispositivos móviles con sistema operativo Android; esto se debe a que se pretende ayudar en el trabajo colaborativo durante la administración de la emergencia, permitiendo a los interesados instalar la aplicación es su dispositivo móvil y acceder a ella en cualquier momento. De esta manera las aplicaciones que se generen a partir del software creado en este proyecto corresponden a aplicaciones Android, lo que hoy en día es una ventaja debido a que este sistema operativo es el más utilizado en los dispositivos móviles a nivel mundial (Forni & van der Meulen, 2016) y una gran variedad de dispositivos de distinta gama lo soportan.

Se llamará al software construido en este proyecto "Generador App", el cual fue construido con el objetivo de mostrar que es posible, utilizando reuso de software a través de componentes, crear un Generador de Aplicaciones móviles para el dominio de administración de emergencias. Este software fue construido en el lenguaje de programación Java, utilizando la herramienta NetBeans IDE para crear las funciones encargadas de separar los componentes y elegir cuáles de ellos son necesarios para construir una nueva aplicación móvil, las

funciones que sirven de soporte para el manejo de archivos y la interfaz de usuario. El módulo encargado de generar la aplicación está basado en Apache Ant, una herramienta que es utilizada mediante línea de comandos, con el principal objetivo de construir aplicaciones en Java, basado en el plugin para eclipse llamado ADT con el cual fue construida la aplicación Mapa Móvil. Para la realización de los diagramas UML de este documento se utilizó la herramienta StarUML; los mapas conceptuales fueron construidos con FreeMind 1.01 y para realizar las conversiones de formato se utilizó Inkscape 0.91.

1.2. Objetivo general

Construir aplicaciones móviles para el dominio de Administración de Emergencias, usando un Generador de Código basado en los componentes reutilizables de la aplicación Mapa Móvil.

1.3. Objetivos específicos

Los Objetivos específicos del proyecto son los siguientes:

- Estudiar la aplicación Mapa Móvil creada en un proyecto de título anterior.
- Construir componentes de software reutilizables adicionales al Mapa Móvil.
- Desarrollar un generador de aplicaciones.
- Probar las aplicaciones generadas.

1.4. Justificación del problema

A continuación se presentan las razones que justifican la realización de este proyecto.

- Se aprovecha el desarrollo realizado en un proyecto anterior al utilizar los componentes generado en la aplicación Mapa Móvil.
- Es parte de una investigación en curso que pretende la creación de una familia de productos mediante una línea de productos de software para apoyar el trabajo colaborativo en el manejo de emergencias en el combate contra incendios.
- Las aplicaciones generadas podrían facilitar la coordinación de equipos durante la administración de una emergencia.
- Resuelve la necesidad de generar aplicaciones personalizadas en no más allá del tiempo que se demora un usuario en seleccionar los componentes de su aplicación y el tiempo de compilación del programa.
- Al generar aplicaciones personalizadas serán construidas solo con los componentes seleccionados lo que permite un ahorro de espacio de almacenamiento que puede ser significativo en dispositivos móviles.

1.5. Delimitación del problema

El generador de aplicaciones está enfocado al dominio de administración de emergencias y debe basarse en los componentes de Mapa Móvil ya desarrollados en un proyecto de título anterior (Ormeño, 2015), y en el modelo de características de la línea de productos de software ya especificada en un proyecto de investigación. Las aplicaciones que se generen deben funcionar en dispositivos móviles Android Versión 2.3.3, 4.2.1 o superior.

Para la generación de las aplicaciones se debe ejecutar el entorno ADT y las aplicaciones producidas deben ser totalmente operativas.

1.6. Metodología

1.6.1. Aprendizaje del lenguaje de programación y entorno de desarrollo

Se estudió cómo desarrollar aplicaciones móviles aprendiendo el lenguaje de programación orientado a objetos Java pero enfocado al desarrollo de aplicaciones Android y cómo trabajar con el ambiente integrado ADT, esto con el fin de entender cómo estaban construidos los componentes de Mapa Móvil. También se estudió Java para el desarrollo de la aplicación Generador App construida en este proyecto.

Otros lenguajes aprendidos fueron XML, esto debido al uso de archivos escritos en este lenguaje para configurar un proyecto Android y generar aplicaciones móviles. Como consecuencia, también se estudió el lenguaje XPatch, que es un lenguaje construido para el direccionamiento de partes de un documento XML (XSL Working Group & XML Linking Working Group, 2016), esto quiere decir que con una ruta válida dentro de un documento se puede obtener uno o varios elementos en el documento.

Para la generación de aplicaciones se estudió Apache Ant, herramienta para construir programas Java mediante línea de comandos, lo que hizo tener que estudiar comandos básicos en la línea de comando del sistema operativo, en este caso Windows.

El estudio de estos lenguajes de programación se realizó primeramente leyendo libros de programación disponibles en la biblioteca de la universidad. Para java se utilizó en especial el libro de Caro, Ramos, & Barceló (2002) que introduce a la POO de la mano de ejemplos prácticos en Java. También se buscó información en la web revisando páginas oficiales de desarrollo con el fin de conocer de manera más exacta los parámetros de los procedimientos y funciones. Entre las páginas revisadas, se encuentra la documentación del

API 7 de Java que fue la que se revisó durante la mayor parte del proyecto. Otra página consultada fue la de la Word Wide Web Fundation, para consultar información relativa a XML y XPatch. Para el estudio de Apache Ant se buscó gran parte de las funciones y comandos en el manual oficial (Bailliez et al., 2015).

Finalmente las dudas que no pudieron ser resuelta con los métodos antes mencionados fueron resueltas consultando foros en internet sobre el entorno de desarrollo y lenguaje de programación.

1.6.2. Método de Desarrollo (UP)

El método que más se ajustó es el Proceso Unificado (UP), debido a la naturaleza iterativa de este método que permite tener entregas parciales del software y poder avanzar en el diseño y desarrollo, sin establecer todos los requisitos. Además, es un método orientado a objetos, que se ajusta al lenguaje de programación en el que se ha desarrollado la aplicación Generador App. Este método consiste en 4 etapas o fases iterativas que van desde el inicio, la elaboración, construcción para finalizar con la etapa de transición (Sommerville, 2005). Estas etapas no representan las etapas de un método de un ciclo de vida en cascada, en el que primero se definen todos los requisitos, pues la fase de inicio no es una fase de requisitos, al igual que la fase de elaboración no es una de diseño (Larman, 1999) sino que en cada iteración se incluye el trabajo de requisitos, diseño e implementación en mayor o menor medida dentro de las fases del proyecto como lo muestra la Figura 1.

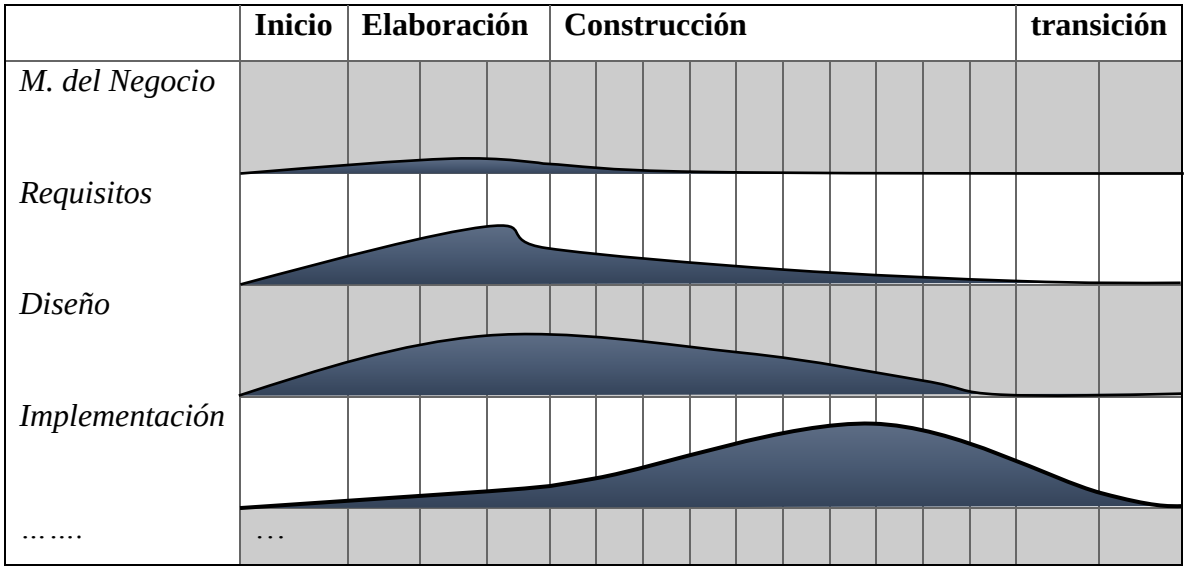


Figura 1: Disciplinas y Fases del proceso unificado

1.6.2.1. Fases

Cada ciclo de desarrollo (iteración) está dividido en cuatro fases que comienza en la fase de inicio, pasando por la elaboración y construcción para terminar en la fase de transición.

Aquí se describen brevemente cada una de estas fases.

Inicio

Se realiza una descripción del producto final a partir de unas buenas ideas; se presenta el análisis de negocio para el producto (Jacobson, Rumbaugh, Jacobson, Booch, & Rumbaugh, 2000), es decir, una visión aproximada, análisis del negocio, alcance y estimaciones imprecisas (Larman, 1999).

Elaboración

En esta etapa se especifican en detalle la mayoría de los casos de uso del producto y se diseña la arquitectura del sistema. La visión se refina, se resuelven los riesgos altos, identificando requisitos faltantes y estimaciones más realistas (Larman, 1999).

Construcción

Aquí se implementan de forma iterativa el resto de requisitos de menor riesgo y elementos más fáciles (Larman, 1999); la línea base de la arquitectura crece hasta convertirse en un sistema completo; esto no quiere decir que estén libre de errores, pero estos defectos se pueden cubrir en la fase de transición (Jacobson et al., 2000).

Transición

El producto se convierte en una versión beta: solo un número reducido de usuarios con experiencia prueba el producto e informa de defectos y deficiencias. De esta manera, los desarrolladores corrigen los problemas e incorporan algunas mejoras sugeridas.

1.6.2.2. Disciplinas

Las actividades de trabajo que se realizan dentro de las fases se denominan disciplinas, que son un conjunto de actividades en un área determinada.

Dentro de los flujos de trabajo he decidido describir tres disciplinas, por ser las más conocidas y las utilizadas para el desarrollo de este proyecto, estas de describen a continuación:

Requisitos

En este punto se realiza un análisis de los requisitos para una aplicación, como escribir los casos de uso e identificar los requisitos no funcionales (Larman, 1999). El esfuerzo principal en esta fase es desarrollar un modelo del sistema que se va a construir; esto se logra de forma adecuada mediante la utilización de casos de uso (Jacobson et al., 2000).

Diseño

En este punto se modela el sistema para dar soporte a todos los requisitos; es el centro de atención al final de la fase de elaboración y comienzo de la iteración de la construcción. El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrando en cómo los requisitos y restricciones relacionadas con el entorno de implementación tienen impacto en el sistema (Jacobson et al., 2000).

Implementación

La implementación se refiere a programar y construir el sistema y no se incluye el despliegue. Los objetivos más importantes de esta disciplina son las de planificar las integraciones de sistema necesarias en cada iteración, implementar clases y subsistemas encontrados durante el diseño, probar los componentes individualmente y a continuación integrarlos (Jacobson et al., 2000).

1.6.2.3. Artefacto

Cada proyecto se organiza en etapas iterativas (inicio, elaboración, construcción y transición); estas etapas poseen en mayor o menor medida trabajo en las distintas disciplinas tradicionales de desarrollo las cuales se desglosan en distintos artefactos. Un artefacto es un término general para cualquier información creada, producida, cambiada o utilizada por los trabajadores en el desarrollo del sistema (Jacobson et al., 2000). En este proyecto se utilizan como artefactos diagramas UMLy Glosario, entre otros. Los diagramas UML se detallan en la sección 2.6.

1.7. Revisión Bibliográfica

La revisión bibliográfica se realizó con libros disponibles en la biblioteca de la universidad; se utilizó Google Académico para buscar citas de libros, artículos y publicaciones, y material disponible de la aplicación Mapa Móvil como información de dominio de Administración de Emergencias, reuso, línea de productos de software y la documentación de la aplicación.

1.8. Organización de Capítulos

Este proyecto fue organizado con los siguientes capítulos:

- Capítulo 1. Marco teórico: Se presentan los temas que se utilizan cómo base para la realización de este proyecto, entre estos temas se encuentra el Reuso de software, Generador de código, UML, la aplicación Mapa Móvil, entre otros .
- Capítulo 2. Estado del Arte: Se presenta un resumen de los proyectos existentes que permiten la generación automática de aplicaciones mediante reuso de software.
- Capítulo 3. Desarrollo: En este capítulo se detallan el desarrollo de la aplicación creada en este proyecto. Entre los temas abordados en este capítulo se encuentra una breve introducción del método de desarrollo utilizado, elección de la herramienta para la construcción de aplicaciones y las disciplinas empleadas en el desarrollo.
- Capítulo 4. Conclusiones: Se encuentran en este capítulo las conclusiones y resultados generados en el proyecto y se indican de manera breve algunas recomendaciones.

2. Marco teórico

2.1. Administración de emergencias

Día a día, los bomberos se deben enfrentar a situaciones de emergencias; estas situaciones pueden ser accidentes de tránsito, incendios, fugas de gas, emergencias eléctricas, rescates, etc. Se puede definir una emergencia como situaciones críticas que requieren que las medidas se tomen inmediatamente para reducir las consecuencias adversas (Engelbrecht, Borges, & Vivacqua, 2011). Con esta definición, se puede observar que se debe actuar de manera inmediata ante una emergencia para reducir estas consecuencias que pueden ser pérdidas de bienes materiales hasta pérdidas humanas, por lo que se deben tener de antemano los procedimientos estándares que ayuden a agilizar en primer lugar la llegada al lugar de la emergencia y en segundo lugar a controlar la situación.

El trabajo de Monares, Ochoa, Pino, & Herskovic (2012) relata un procedimiento llevado a cabo en estas situaciones. Cuando llega el primer camión al lugar de la emergencia, la fase de respuesta inicial termina y comienza la fase de respuesta efectiva. A partir de ese momento, el comandante del incidente (IC) está a cargo de la respuesta de emergencia en lugar de Centro de Alarma (AC), que cambia a un actor secundario en el proceso. El IC evalúa la situación y toma decisiones sobre cómo llevar a cabo de mejor manera la emergencia, teniendo en cuenta los recursos que le asigna la AC a la emergencia. Los hitos del proceso de respuesta se muestran a continuación:

Fase de respuesta inicial

- Notificación a la organización (bomberos) de la emergencia.
- Verificación de la alarma.
- Asignación y envió de recursos.
- Llegada del primer camión de bomberos en el sitio de emergencia.

Fase de respuesta efectiva

- Establecimiento de Comandante del Incidente (IC).
- Evaluación iterativa y la toma de decisiones para hacer frente con urgencia.
- Situación se controla.
- Mitigación de Emergencia y cierre.

Los Bomberos en Chile son voluntarios por lo que generalmente poseen otro empleo remunerado; cuando se produce una emergencia, ellos deciden si asisten o no dependiendo del incidente (Monares et al., 2012). Por este motivo, es importante contar con un medio donde los bomberos puedan conocer las emergencias que suceden cerca del lugar donde ellos se encuentran; esto se podría realizar a través de dispositivos móviles que ayudarían a saber el lugar de la emergencia y el tipo de emergencia, además del personal que asiste en ese momento.

2.2. Reuso

El pensar en reuso en el desarrollo de software tal vez no es una idea tan clara, esto es, debido a que el software es algo intangible. Una vez terminado un software se puede tomar el archivo ejecutable y copiarlo muchas veces y obtener el mismo producto replicado pero esta no es la idea del reuso. El reuso consiste en poder tomar un programa o componente de software y buscar que funciones o procedimientos se pueden reutilizar para la generación de un nuevo programa. Una definición de reuso es la utilización de software existente o conocido (Frakes, 2005), este software existente es el que sirve para generar nuevos productos a partir de una base común. Esto no solo implica la utilización del código fuente, sino que también implica la utilización de los requisitos, modelos y toda la documentación existente sobre el software.La Tabla 1 muestra una comparativa entre lo que es reuso y lo que no.

Tabla 1: Tabla reuso vs no reuso

REUSO	***NO ES REUSO***
Utilizar modelos y documentos creados en otro proyecto software.	Utilizar la metodología de desarrollo de otro proyecto sin usar sus modelos o documentación.
Utilizar código fuente creados en otro proyecto software.	Replicar programa ejecutable en distintos dispositivos.

2.2.1. Línea de productos de software

Para facilitar el reuso es conveniente incluir con anticipación este enfoque en el desarrollo de software. Una forma de hacer esto es la generación de líneas de productos de software.

Por ejemplo, tiene un conjunto de teléfonos móviles que permiten llamar, mandar mensajes, escuchar música, conectarse a internet y poseen una pantalla táctil y cámara integrada. Uno de estos teléfonos posee unos parlantes en cada extremo que incorporan la capacidad de escuchar música en estero, mientras que otro posee un teclado qwerty que facilita la entrada de texto y finalmente un tercer modelo incorpora una luz trasera que funciona como linterna integrada. En este ejemplo se observa una línea de productos, donde se identifican los elementos comunes de los teléfonos (llamadas, envío de mensajes, cámara integrada, etc) y algunas variaciones de ellos, donde cabe destacar que estas variaciones permiten la personalización de cada producto adaptada a necesidades distintas. De la misma forma que en el ejemplo anterior, el objetivo de las líneas de producto de software es poder utilizar los elementos comunes y gestionar la variaciones de manera eficaz (Díaz & Trujillo, 2010), donde estos elementos se asocian al dominio del problema.

Una línea de productos de software se define como un conjunto de sistemas, que comparten un conjunto de características comunes, las que satisfacen las necesidades

específicas de un dominio particular de mercado, y que se desarrollan a partir de un sistema común de activos base de manera preestablecida (Clements & Northrop, 2002). Las características comunes que comparten estos sistemas se pueden obtener observando con detalle el dominio de aplicación para encontrar lo que se repite en cada sistema y generar los activos base. Así estos activos (componentes) comunes deben generarse pensando en el reuso, creando interfaces para que éstos puedan acoplarse con nuevos componentes creados para el dominio.

Existen distintas técnicas para producir líneas de productos de software que pueden encontrarse en el trabajo de Frakes (2005), las que se describen a continuación:

- Family-Oriented Abstraction, Specification and Translation (FAST), que define patrones de procesos de ingeniería que son comúnmente usados en líneas de productos de software.

- Domain analysis and reuse environment (DARE) estudia cómo muchos análisis de dominio pueden ser basados en procesos repetibles y cómo muchos pueden ser automatizados. Los procesos de DARE se toman de tres fuentes de información: código, documentos y expertos del conocimiento como la base de modelos del dominio.

- Product Line UML-Based Software Engineering (PLUS), que proporciona varias técnicas de modelado y notaciones para línea de productos de software como casos de uso para los requisitos, diagramas de clases para el análisis de la línea de productos de software, etc.

- Feature-Oriented Reuse Method FORM es un método sistemático que busca y captura lo que se parece y las variabilidades en términos de características en la línea de productos de software. Éste método se utiliza para soportar la reutilización de activos (componentes) en una línea de productos de software o en el desarrollo de productos software que utilicen componentes.

- Komponentbasierte Anwendungsentwicklung (KobrA) es un método de línea de productos de software basado en componentes abordado con UML; provee un enfoque de

desarrollo de activos genéricos que pueden adaptar variaciones en la línea de productos a través de un marco de ingeniería.

2.2.2. CBSR (Ingeniería de Software Basada en componentes)

Cuando se construye software pensando en la reutilización, una buena idea es construir software basado en componentes. Se puede pensar en un componente de software como en una pieza de automóvil que puede ser cambiada cuando ésta falla o puede incluso ser reemplazada por una mejor, debido a que posee un conector estándar que permite el fácil ensamblaje de la pieza. Con esto, se puede tener una idea de qué es un componente de software, aunque se puede definir como una pieza de software funcional que es liberada independientemente de otras y que proporciona acceso a sus servicios a través de sus interfaces (Montilva, 2006).

La construcción de componentes de software puede ser la base para una línea de productos de software. Un componente puede ser usado como una pieza que sea fácil de reemplazar cuando se detecte una falla o necesite una actualización, aunque el interés de este proyecto es utilizar el desarrollo basado en componentes para el reuso de software.

Los componentes son unidades de software que se rigen por los mismos principios presentes en el paradigma de orientación a objetos: unificación de datos y comportamiento, identidad y encapsulamiento. A éstos se le agrega el uso obligatorio de interfaces (Vignaga & Perovich, 2003).

2.3. Java para aplicaciones Android

Java es un lenguaje de programación potente se encuentra hoy en día en 3 mil millones de teléfonos móviles (Java, 2014) y en distintos dispositivos como blue-ray, televisores, etc. Este lenguaje de programación fue creado en Sun Microsystems, Inc. en 1991 por James

Gosling, Patrick Naughton, Chris Warth, Ed Frank, y Mike Sheridan aunque inicialmente se llamaba Oak, su nombre se cambió a Java en 1995. Este lenguaje de programación se caracteriza por ser un lenguaje fácil de aprender y que pueden funcionar en varios sistemas operativos instalando la máquina virtual de Java en el sistema operativo y distintos dispositivos que ejecutan Java.

Android es un sistema operativo de código abierto bajo licencia Apache, perteneciente a Google desde julio de 2006 (Nolasco Valenzuela, 2014). Este sistema operativo se basa en la versión 2.6 de Linux (Saha, 2012) para la programación de aplicaciones es necesario Android SDK que incluye las herramientas de desarrollo, emulador de Android y las bibliotecas necesarias para la creación de una aplicación en Android. Para programar en este entorno de desarrollo es necesario obtener Eclipse un IDE al cual se le debe agregar en plug-in llamado Android Development Tools (ADT). Con esto es posible crear aplicaciones en el lenguaje de programación Java.

2.4. Generador de código

El desarrollo de aplicaciones generalmente lleva consigo en alguna proporción código que se repite; esto ha llevado a la utilización de funciones y bibliotecas que pueden utilizarse para utilizar el mismo código dentro de un programa sin la necesidad de tener que repetirlo.

Esta reutilización de código ha llevado a la idea de poder tomar los elementos comunes de un dominio en particular para la generación automática de aplicaciones tomando como base estos elementos. Un generador de código básicamente toma la especificación de un sistema o componentes y retorna el sistema o componente final (Czarnecki & Eisenecker, 1999), esto quiere decir que a partir de la especificación se deben tomar los componentes que tomará el generador de código. Se debe tener en cuenta que los componentes de software son comunes entre las aplicaciones que se pretenden generar y se deben crear interfaces que permitan el acoplamiento adecuado de estos componentes.

El generador de código toma la especificación a través de valores predeterminados, luego comprueba si el sistema se puede construir, completa la especificación y monta los componentes restantes. Aunque existe una comprobación para determinar si el sistema se puede construir, es recomendable no dar configuraciones que no sean compatibles al momento de especificar. Si se piensa en un ejemplo de la vida real, puede ver que en un automóvil no se puede montar una rueda bajo el asiento del conductor. Aunque puede resultar algo obvio en este ejemplo, esto es más difícil de observar cuando se trata de software al ser algo intangible. Para evitar realizar estas combinaciones es conveniente generar un modelo de características que es un conjunto jerarquizado de funciones con relación entre un padre y las características secundarias.

La programación de un generador de aplicaciones implica tener que generalizar los algoritmos; esto significa que, en la medida de lo posible, los algoritmos deben ser parametrizados al máximo y expresados de la forma más independiente posible de los detalles concretos (Lecca, 2007). Para ayudar a esta generalización es conveniente utilizar lenguajes de programación orientados a objetos que a través de la utilización de herencias puede permitir la reutilización de código.

2.5. Patrones de diseño

En el mundo de la informática es común encontrarse con un conjunto de problemas que son recurrentes. Para no tener que resolver una y otra vez estos problemas, es conveniente utilizar patrones de diseño. Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de su solución en forma tal que es posible usarla un millón de veces sin elaborarla dos veces de la misma forma (Pressman, 2006). Existen gran cantidad de patrones que se separan principalmente en tres tipos:

2.5.1. Patrones Creacionales

Estos se centran en la creación, composición y representación de objetos y ofrecen mecanismos que hacen más fácil la formación de las instancias de los objetos dentro de un sistema y establecen restricciones en el tipo y número de objetos que es posible crear dentro de un sistema (Pressman, 2006). Dentro de los patrones creacionales se encuentran Abstract Factory, Builder, Prototype, Singleton.

2.5.2. Patrones Estructurales

Se centran en problemas y soluciones asociados con la manera en la que se organizan e integran las clases y objetos para construir una estructura más grande, es decir, ayudan a establecer relaciones entre entidades dentro de un sistema (Pressman, 2006). Dentro de los patrones estructurales se encuentran Adapter, Bridge, Composite, Decorator, Facade, Proxy.

2.5.3. Patrones Conductuales

Se enfocan a problemas asociados con la asignación de responsabilidad entre los objetos y la manera en la que se efectúa la comunicación entre ellos. Dentro de estos patrones se pueden mencionar Chain of Responsibility, Command, Iterator, Mediator, entre otros.

2.5.4. Descripción de Patrones

Gamma, Helm, Johnson, & Vlissides (1996) son los que mejor describen los patrones de diseño por lo que se han escogido algunos de los patrones descritos por ellos para presentarlos en este proyecto.

Builder

Patrón creacional que separa la construcción de un objeto complejo de su representación de manera que el mismo proceso de construcción puede crear diferentes representaciones. Este patrón debe ser utilizado cuando el algoritmo para la creación de un objeto complejo debe ser independiente de las partes que componen el objeto y la forma en que están montados. El proceso de construcción debe permitir diferentes representaciones del objeto que se está construyendo.

Singleton

Patrón creacional que asegura que una clase solo tiene una instancia, y proporcionar un punto de acceso global a la misma. Este patrón se utiliza cuando hay exactamente una instancia de una clase, y debe ser accesible a los clientes de un punto de acceso conocido.

Cuando la única instancia debería ser extensible por subclases, y los clientes deben ser capaces de utilizar una instancia extendida sin modificar su código.

Bridge

Patrón estructural que desacopla una abstracción de su aplicación de manera que los dos pueden variar independientemente. Este patrón se utiliza cuando se quiere evitar una unión permanente entre una abstracción y su implementación.

Decorator

Patrón estructural que fija responsabilidades adicionales a un objeto dinámicamente. Proporcionan una alternativa flexible a la subclasificación para extender la funcionalidad. Se utiliza para añadir responsabilidades a objetos individuales de forma dinámica y trasparente, es decir, sin afectar a otros objetos

Command

Patrón de tipo conductual que encapsula una petición como un objeto, permitiendo de ese modo que parametrizar clientes con diferentes peticiones, en cola o iniciar peticiones, y apoyar las operaciones que se pueden deshacer. Se utiliza para parametrizar los objetos mediante una acción a realizar, Command es un reemplazo orientado a objetos para las devoluciones de llamada.

2.6. UML

UML, siglas de Unified Modeling Lenguage, es un lenguaje de modelado cuyo principal objetivo es proporcionar a expertos en el área del desarrollo de software herramientas para el análisis, diseño e implementación de sistemas basados en software, también permite modelar procesos empresariales y similares (Group, 2010). Fue creado a mediados de la década de los 90 cuando se fusionaron los elementos de modelado de Rumbaugh, Booch y Jacobson apara crear el proceso unificado de modelado que poco tiempo después se eliminó el proceso de la especificación para convertirse en UML (Kimmel, 2006).

Al ser un lenguaje de modelado, UML posee una cantidad bastante extensa de modelos tanto para el análisis y el diseño. Aquí solo se prestará atención a los modelos utilizados en este proyecto.

2.6.1. Diagrama de casos de uso

Los diagrama de casos de uso, también conocidos como cajas de uso, son una manera de capturar los requisitos del sistema que se desea realizar. Éstos poseen dos conceptos claves que son el actor y el caso de uso (Group, 2010), estos se relacionan entre sí mediante distintos tipos de conectores para representar el sistema y su interacción con actores internos y externos.

Actores

Un actor es una figura de palillos como lo muestra la Figura 2 y representa participantes en los casos de uso, estos pueden ser personas o cosas, las cosas pueden ser subsistemas o subprograma y son descubiertos mediante el análisis del sistema (Kimmel, 2006), este actor recurre a los servicios del sistema para cumplir un objetivo (Larman, 1999).

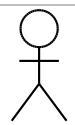


Figura 2: Símbolo de un Actor

Casos de uso

Para representar los requisitos del sistema se utilizan los casos de uso, la mejor definición es la de los mismos autores de UML, que dicen que un caso de uso especifica una secuencia de acciones, incluyendo variantes, que el sistema puede llevar a cabo, y que producen un resultado observable de valor para un actor concreto (Jacobson et al., 2000). Éstos se representan en los diagramas de caso de uso con un óvalo al que se le da un nombre y una descripción mediante texto como lo muestra la Figura 3.

Figura 3: Símbolo de un Caso de Uso

Conectores

Existen distintos tipos de conectores, estos sirven para indicar la manera en que cajas de uso y actores están asociados, diferentes estilos de conector transmiten distinta información acerca de esta relación. Los tipos de conectores se encuentran indicados en la Figura 4.

El primer conector indica una asociación bidireccional entre dos elementos de un diagrama, esto quiere decir que si se tiene un elemento A en un extremo y un elemento B en el otro extremo, A es parte de B y B es parte de A.

El segundo conector de la Figura 4 que tiene forma de flecha indica una asociación en un solo sentido, esto quiere decir que si se tiene un elemento A en el extremo que comienza la flecha y un elemento B en el extremo donde esta la punta de la flecha se dice que A es parte de B, pero B no es parte de A.

El tercer conector de la Figura 4 indica una relación de dependencia, si en el extremo izquierdo de la figura se tiene un elemento A y en el otro extremo un elemento B, se dice que A usa a B.

El cuerto conector indica una relación de generalicación o herencia. Esto quiere decir que si en el extremo izquierdo de la figura existe un elemento A y en otro extremo un elemento B, se dice que A es un B ó que A es un subtipo de B.

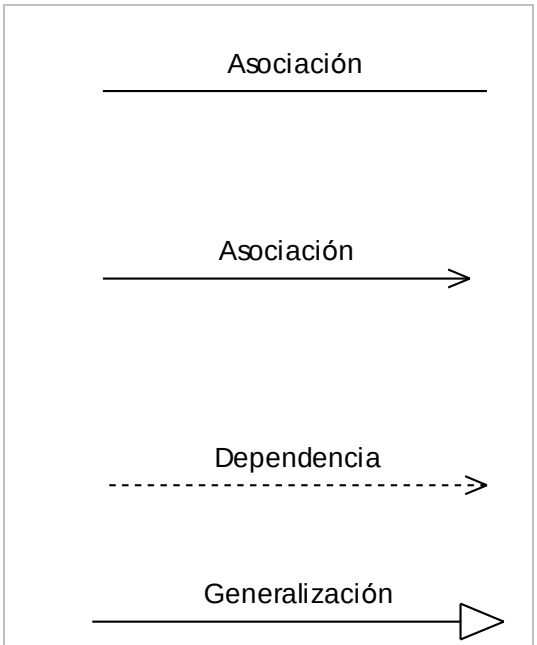


Figura 4: Conectores básicos en UML

Es importante destacar que los diagramas de caso de uso son una herramienta en UML que permite de manera visual describir un caso de uso, donde un caso de uso no es un diagrama sino un documento de texto que define los requisitos del sistema (Larman, 1999). Por esto se debe prestar mayor atención en la escritura de los casos de uso y luego en crear un diagrama simple, donde se visualicen los actores del sistema.

Inclusión y extensión

Dentro de la relación de dependencia existen dos estereotipos comunes que son el de incluir(include) o extender (extend).

El estereotipo incluir en la relación de dependencia, indica que un caso de uso dependiente necesita los servicios del otro caso de uso. El caso de uso del que se depende representa una entidad completa y distinta que no debe depender del caso de uso dependiente (Kimmel, 2006).

El esteretipo excluir en la relación de dependencia, indica que un caso de uso extiende a otro para dar una funcionalidad extra, esto quiere decir que se puede agregar más

capacidades a un caso de uso básico creando un caso de uso que se llamará caso de uso de extensión donde la flecha en la relación de dependencia debe terminar en el caso de uso básico (Kimmel, 2006).

En el ejemplo de la Figura 5 se muestra un actor llamado jugador que desea realizar un movimiento, esta acción se representa en la caja de uso "Realizar movimiento"; para realizar este movimiento siempre se muestran las cartas en juego, es por eso que "Realizar movimiento" incluye el caso de uso "listar cartas en juego", de manera opcional puede buscar una carta en la baraja, es por esto que el caso de uso "Buscar en Baraja" extiende a "Realizar movimiento".

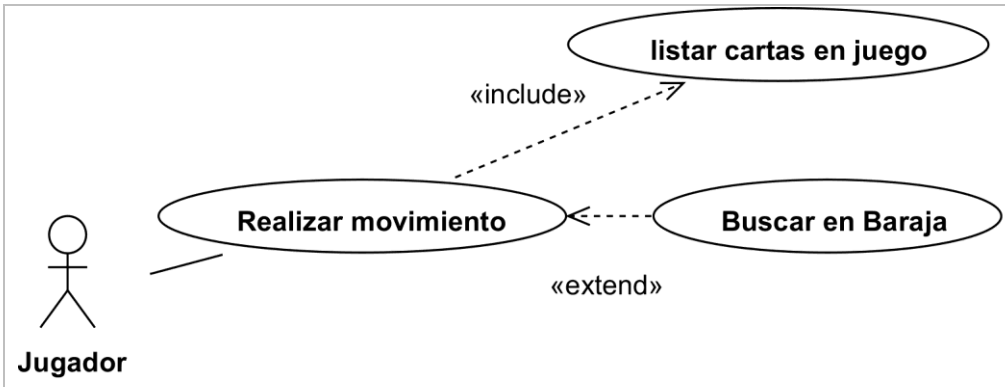


Figura 5: Ejemplo de Caso de Uso

2.6.2. Diagrama de actividades

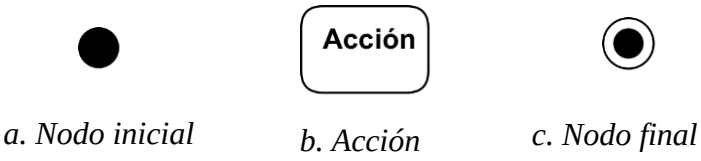
Un diagrama de actividades es similar a un diagrama de flujo; se usa para analizar procesos, y constituye un puente progresivo que conduce del análisis al diseño (Kimmel, 2006).

Una activadad es un tipo de comportamiento que se especifica como un gráfico de nodos interconectados por aristas (Group, 2010). Es similar a un diagrama de flujo, la principal diferencia es que los diagramas de actividad pueden modelar comportamiento paralelo.

Los principales símbolos utilizados en los diagramas de actividad se presentan en esta sección.

Nodo inicial

Es donde comienza todo diagrama de actividades. Se representa por un círculo relleno como lo muestra la Figura 6-a, puede tener una línea de transición saliendo de él la cual es llamada flujo de control.



a. Nodo inicial *b. Acción* *c. Nodo final*

Figura 6: Nodo inicial, de acción y nodo final

Acción

Estos nodos representan las acciones que se llevan a cabo en el diagrama de actividades; pueden tener más de un flujo de entrada, pero un solo flujo de salida; llevan un texto que describe la acción. En la Figura 6-b se muestra un nodo de acción.

Nodo Final

Indica el término de las acciones en el diagrama, se representa por un círculo que contiene otro círculo relleno como lo muestra la Figura 6-c.

Flujo de Control

El flujo de control es una flecha dirigida, que comienza en un nodo inicial o en una acción y puede terminar en otro nodo. Su función es conectar nodos en el diagrama de actividad, siendo posible añadir al flujo de control un guardián que es un texto entre corchetes. Se utiliza comúnmente para señalar una condición antes de continuar al nodo siguiente. La Figura 7-a muestra un flujo de control simple, mientras que la Figura 7-b un flujo de control con guardián.

a. Flujo de Control *b. Flujo de Control con guardián*

Figura 7: Flujo de Control

Nodo de decisión y fusión

Un nodo de decisión tiene la forma de un diamante como lo muestra el ejemplo en laFigura 8, transmiten la ramificación y la fusión condicionales y se usa un guardián (ver Figura 7-b) en los flujos de control que salen de un nodo de decisión para representar la lógica.

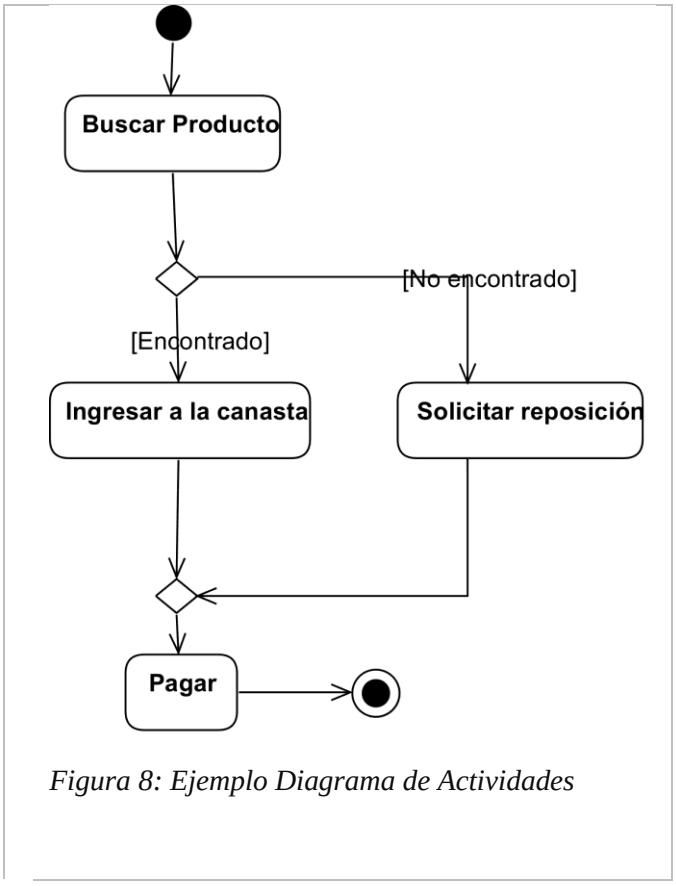


Figura 8: Ejemplo Diagrama de Actividades

Nodos conectores

Se utilizan cuando en un diagrama es difícil conectar dos acciones, los nodos conectores se representan por un círculo con un identificador como muestra la Figura 9, un nodo conector recibe el flujo de control de una acción y el otro nodo conector se conecta con un flujo de control hacia la siguiente acción.

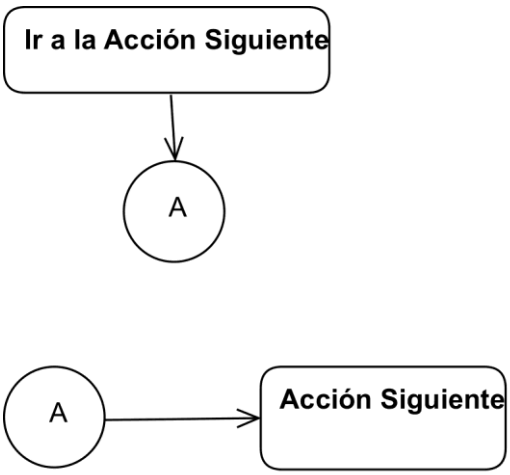


Figura 9: Los Nodos etiquetado con la letra "A" representan los nodos conectores

Bifurcaciones y uniones de transición

Describen comportamiento paralelo. Esto no necesariamente representa las acciones ocurriendo en forma simultánea, también podrían ocurrir de manera intercalada, etc., solo representa las acciones bifurcadas en el transcurso de un intervalo compartido y concurrente. La Figura 10 muestra un ejemplo de bifurcación.

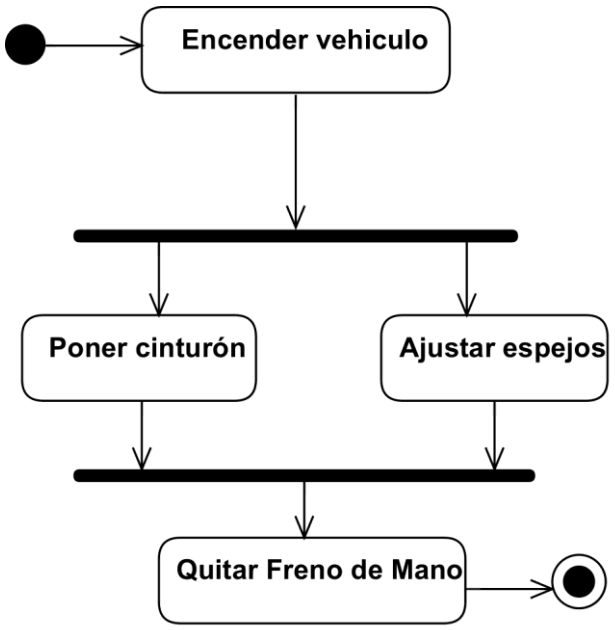


Figura 10: Ejemplo Bifurcación y unión de transición.

2.6.3. Diagrama de clases

Seguramente uno de los elementos más usados y conocidos de UML son los diagramas de clases. Un diagrama de clase describe las diferentes entidades del sistema llamadas clases, como también las relaciones estáticas. También muestra los componentes de una clase (Muñoz Caro et al., 2003).

Dentro de los elementos básicos de un diagrama de clase se encuentra el clasificador y los conectores que se describen a continuación.

2.6.3.1. Clasificador

Un Clasificador es un rectángulo que puede representar a una clase o un ejemplar de una clase; este puede llevar el nombre de la clase o ejemplar, seguido de sus atributos y comportamientos (métodos, procedimientos) (Kimmel, 2006). La Figura 11 muestra un ejemplo sencillo de un clasificador mostrando la clase Carta.

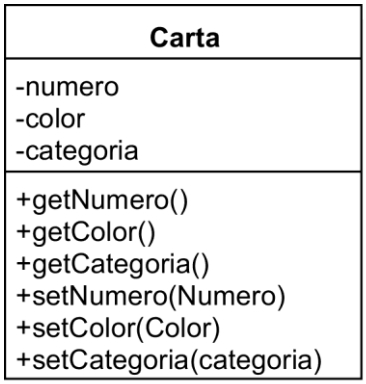


Figura 11: Clasificador

2.6.3.2. Relaciones entre clases

Las clases se relacionan entre ellas mediante conectores, que son los mostrados en la Figura 4, éstos sirven para representar asociaciones, jerarquía etc.

Generalización

La generalización, también llamada herencia se presenta cuando una clase es un subtipo de otra; a la primera clase se le llama superclase o clase padre y a la clase que es un subtipo de esta se llama subclase o clase hija (Caro et al., 2002). La clase hija hereda los atributos y procedimientos del padre, pudiendo añadir los propios. La relación se presenta

mediante una flecha de punta vacía que apunta hacia la clase padre como muestra el ejemplo que se ilustra en la Figura 12.

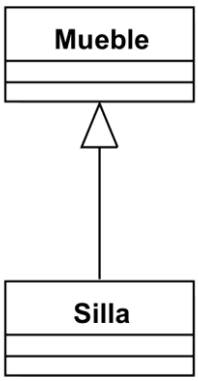


Figura 12: Ejemplo de generalización, se lee "Silla" es un/a "Mueble"

Asociación

Es cuando una clase está estructuralmente compuesta de otras clases. Para lograr esto, se usa algún objeto de una de las clases como atributo de la clase compuesta. Es posible indicar cuántos objetos están conectados en una relación de asociación (Muñoz Caro et al., 2003). Esta relación se representa con una línea continua como lo muestra el ejemplo de la Figura 13. Para representar multiplicidad se indica numéricamente el valor como lo muestra la Figura 14, si el número de objetos por el que está compuesta una clase es indefinido se indica con un asterisco como lo indica la Figura 15.

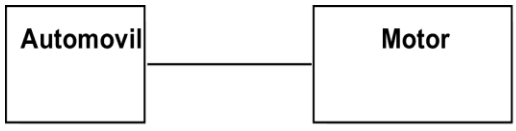


Figura 13: Ejemplo de asociación, se lee "Automovil" tiene un "Motor".

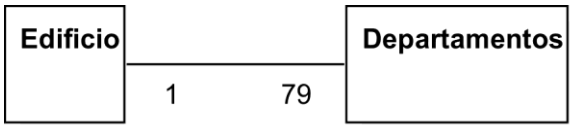


Figura 14: Ejemplo de asociación n-aria, se lee "Edificio" tiene '79' "Departamentos".

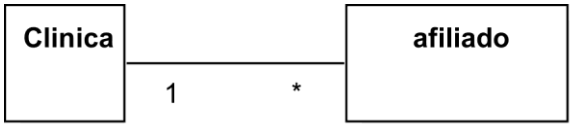


Figura 15: Ejemplo de asociación, se lee "Clinica" tiene indefinidos "afiliado"

Dependencia

Es una relación de utilización, donde un cambio en el estado de un objeto afecta al estado del otro, pero no a la inversa. Esta relación aparece cuando una clase se relaciona con otra a través de los mensajes que le envía, es decir, que se pasa un ejemplar de la clase independiente como uno de los parámetros del método invocado (clase dependiente) (Muñoz Caro et al., 2003). En la Figura 16 se muestra un ejemplo de la relación de dependencia.

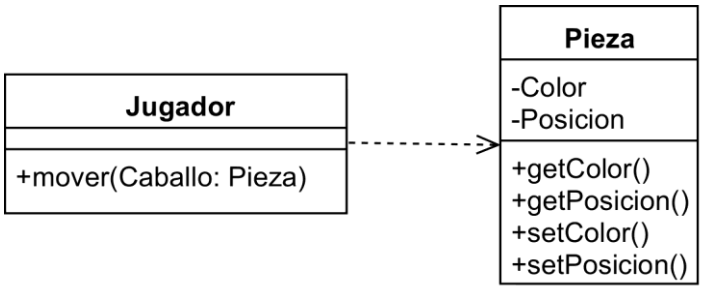


Figura 16: Ejemplo de dependencia, se lee "Jugador" usa a "Pieza"

2.7. Mapa Móvil

Mapa Móvil es una aplicación desarrollada por Erika Ormeño, la cual es una modificación de una aplicación desarrollada inicialmente en la Universidad de Chile. El aporte realizado por Erika Ormeño fue a partir del modelo de características crear una aplicación basada en componentes.

Para lograr la separación de la aplicación en componentes se aislaron las funcionalidades de la aplicación en componentes reutilizables, lo cual fue logrado utilizando patrones de diseño. La principal tarea de estos componentes es servir a futuro como una base para la creación de múltiples aplicaciones en el dominio de la administración de emergencia. La aplicación está basada en el modelo de características de la Figura 17, que muestra de manera jerarquizada las funciones de la aplicación.

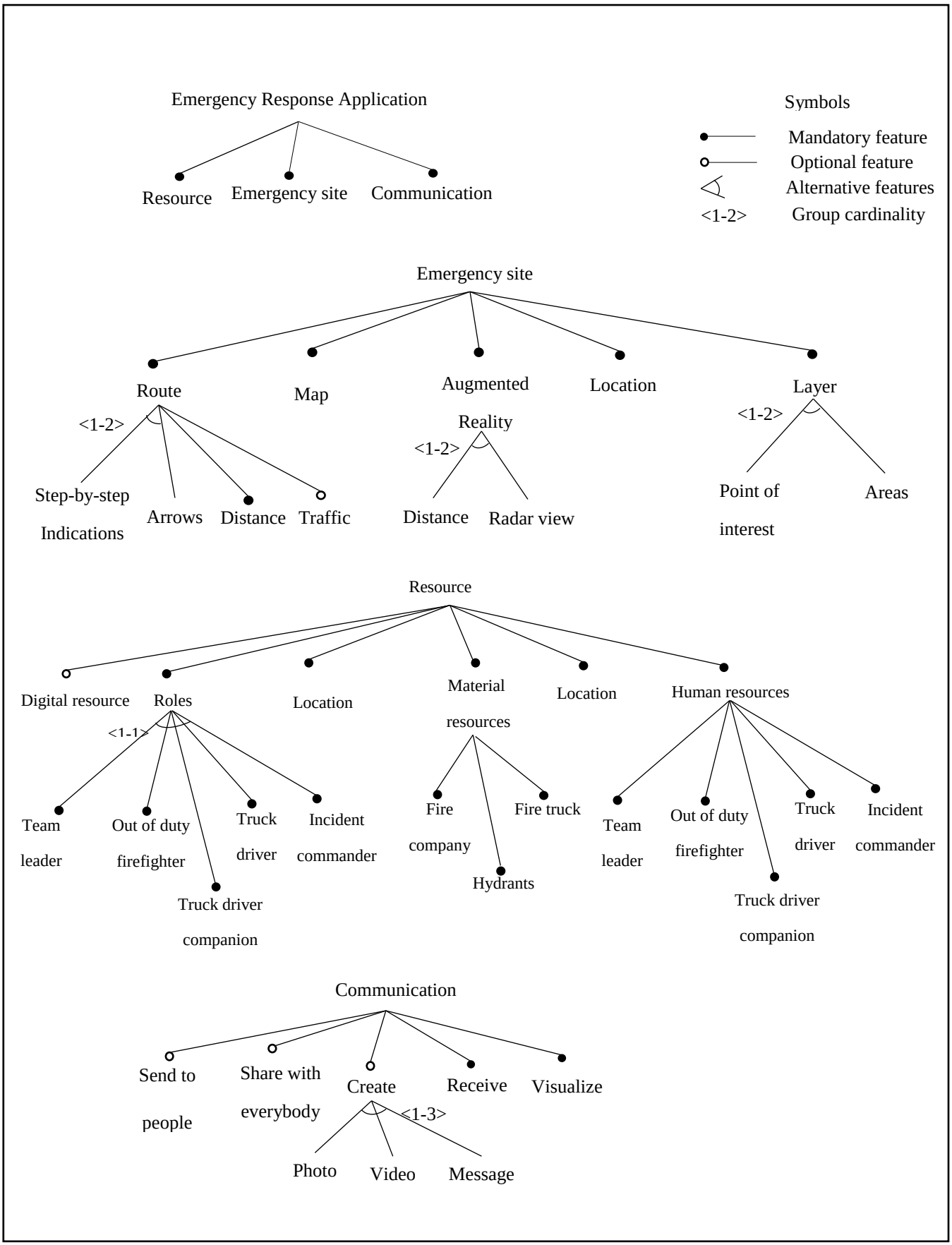


Figura 17: Modelo de características (Rossel et al., 2016)

Las características del modelo que fueron implementadas en Mapa Móvil se muestran en 3 tablas que corresponden a la característica Resource (Tabla 2), Emergency site (Tabla 3) y Communication (Tabla 4). La lista total de componentes de Mapa Móvil puede ser revisada en el ANEXO F.

Tabla 2: Resource

Resource	
Feature	***Componentes que implementan la característica***
Material resources	
-Fire company	-ConcreteStaticResourceTypes
-Hydrants	-ConcreteStaticResourceTypes
-Fire truck	-ConcreteDynamicResourceTypes
Digital resource	-ConcreteDigitalResourceTypes
Human resources	
-Team leader	……
-Out of duty firefighter	-ConcreteDynamicResourceTypes
-Truck driver companion	……
-Truck driver	……
-Incident commander	-ConcreteDynamicResourceTypes
Roles	
-Team leader	……
-Out of duty firefighter	-UserLocationLayer (mediante el uso de la actividad EmergencyMap)
-Truck driver companion	……
-Truck driver	……
-Incident commander	-UserLocationLayer (mediante el uso de la actividad EmergencyMap)
Location	-ResourcesLayer (mediante el uso de la actividad EmergencyMap)

Tabla 3: Emergency site

Emergency site	
Feature	***Componentes que implementan la característica***
Map	-GMapA (mediante el uso de la actividad EmergencyMap)
Augmented reality	
-Distance	……
-Radar view	……
Location	-EmergencyLayer (mediante el uso de la actividad EmergencyMap y el componente EmergenciesList)
Route	
-Step-by-step indications	……
-Arrows	-EmergencyDirectionLayer* -UserLocationLayer* *Mediante el uso de la actividad EmergencyMap.
-Distance	-ConcreteCoordinateUtilities* -ConcreteDistanceUtilities* -ConcreteUserLocationReaderA* *Mediante el uso de EmergenciesList y EmergencyMap.
-Traffic	……
Layer	
-Point of interest	ResourcesLayer* UserLocationLayer* *Mediante el uso de EmercencyMap
-Areas	

Tabla 4: Communication

Communication	
Feature	***Componentes que implementan la característica***
Create	
Photo	ConcretePictureMakerA (mediante DigitalResourcesMaker)
Video	ConcreteVideoMakerA (mediante DigitalResourcesMaker)
Message	UserStatusForm
Send to people	Photo/Video: ConcreteDigitalResourceSender (mediante DigitalResourcesForm)
Share with everybody	Photo/Video: ConcreteDigitalResourceSender (mediante DigitalResourcesForm) Message: UserStatusSender (mediante UserStatusForm)
Receive	ConcreteLocalRepositorySynchronizer
Visualize	ConcreteDigitalResourceOpener ConcreteLocalRepositoryReader ResourcesLayer

Dada la importancia de los componentes en esta aplicación se realizará un resumen de los tipos de componentes que conforman Mapa Móvil.

Los componentes creados para esta aplicación se separaron en tres grandes grupos que se resumen en la Tabla 5.

Tabla 5: Grupo de componentes

GRUPO COMPONENTE		DESCRIPCIÓN
LÓGICA DE NEGOCIO	:	Entrega funcionalidades de la Línea de Producto de software.
COMPONENTES GENERALES	:	Administran a los componentes de la lógica de negocio, encargando de conectar la mayoría de las interfaces que requieren los componentes que administran.
INTERFAZ DE USUARIO	:	Hacen uso de las funcionalidades ofrecidas por lo componentes de la lógica de negocio y son llamadas por el sistema operativo.

2.7.1. Componentes de la lógica de negocio.

Estos componentes entregan las funcionalidades a la línea de productos de software como también representan a todos aquellos procedimientos no visibles. Para facilitar la creación de futuras aplicaciones, estos componentes se dividen en dos grupos que indican que componentes son independientes del sistema operativo Android.

Componentes independientes

Dependen de las bibliotecas básicas o paquetes externos que no dependen del sistema operativo en que se ejecutará la aplicación.

Componentes dependientes

Dependen del sistema operativo en el que se ejecutará la aplicación; en este caso estos componentes dependen de Android.

2.7.2. Componentes Generales

Son los encargados de administrar a los componentes de la lógica de negocio y conectan la mayoría de las interfaces de éstos. La administración es realizada en base a dos grandes componentes, éstos componentes son ConcreteBasicComponentsA y ConcreteOptionalComponentsA.

Administración de los componentes básicos

Los componentes básicos son administrados por la clase ConcreteBasicComponentsA que hereda de la clase DefaultBasicComponents. Cada una de ellas gestiona los ejemplares de los componentes básicos dependientes e independientes del sistema operativo Android respectivamente; esta última clase hereda de la clase abstracta BasicComponents los métodos abstractos como se muestra en la Figura 18.

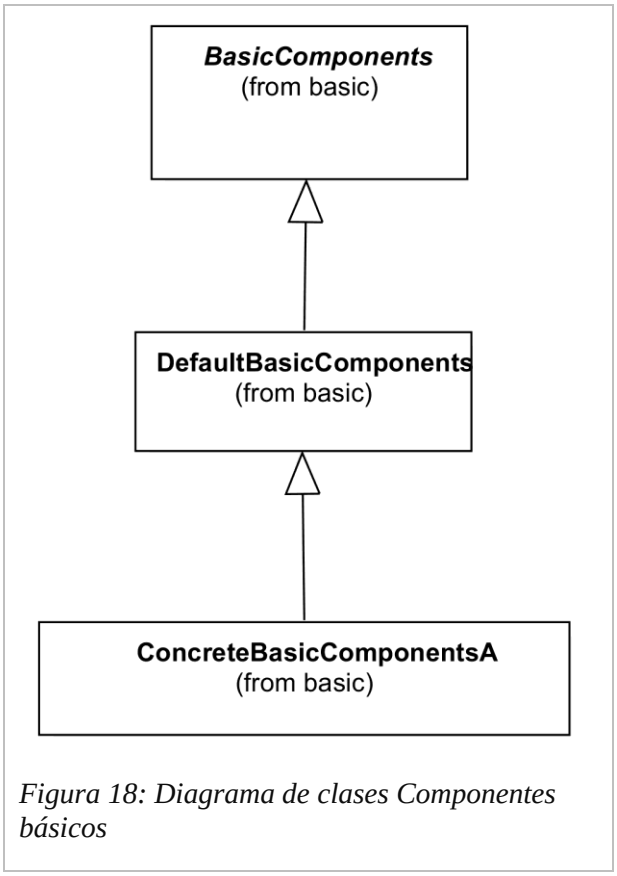


Figura 18: Diagrama de clases Componentes básicos

Administración de los componentes Opcionales

De manera equivalente los componentes Opcionales se manejar un una clase llamada ConcreteOptionalComponentsA que hereda de la clase DefaultOptionalComponents. Cada una gestiona los ejemplares de los componentes opcionales dependientes e independientes de Android respectivamente. Esta última clase a su vez hereda de la clase abstracta OptionalComponents los métodos abstractos como se muestra en el la Figura 19.

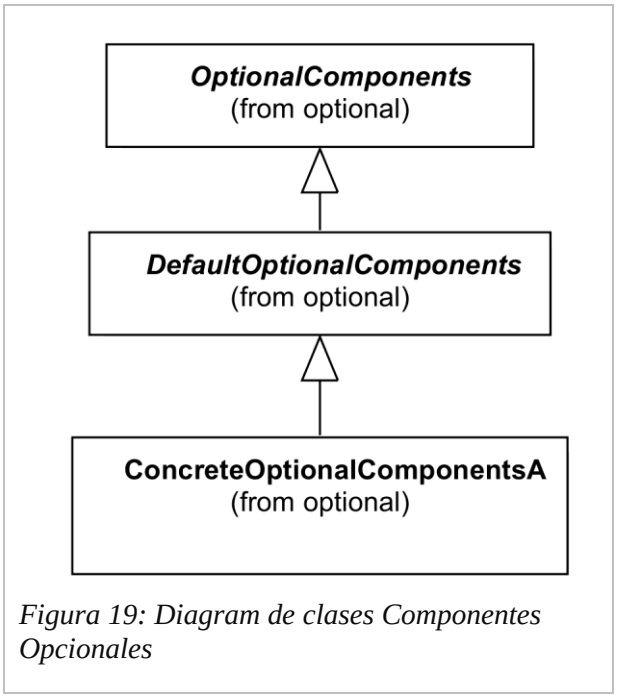


Figura 19: Diagram de clases Componentes Opcionales

2.7.3. Componentes de Interfaz de Usuario

Estos componentes son los que se conectan directamente con el sistema operativo y sirven para generar la Interfaz de usuario, en el sistema opertaivo Android la interfaz grafica se representa mediante Actividades, una actividad esta compuesta por un archivo xml donde se configura la disposición de los elementos en la pantalla y una clase asociada a este archivo.

Para una mejor comprensión de la organización de componentes en la Figura 20 se puede observar un mapa conceptual que muestra la clasificación de componentes creados en Mapa Móvil.

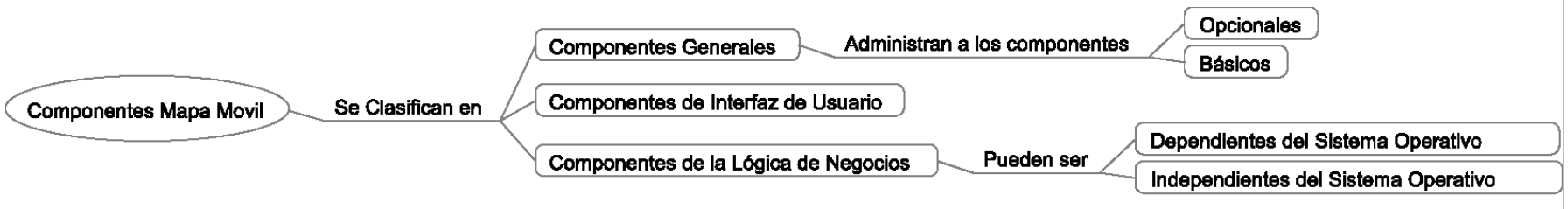


Figura 20: Mapa Conceptual con la clasificación de componentes de Mapa Móvil

3. Estado del Arte

Hoy en día se busca generar software más complejo en una menor cantidad de tiempo, lo que implica la utilización de nuevos métodos de desarrollos y técnicas que permitan este tipo de desarrollo. Para esto se estudió la generación automática de código, con la ayuda de técnicas como línea de productos de software que permite buscar las características comunes en el dominio de aplicación (Díaz & Trujillo, 2010).

El trabajo de Díaz Ríos (2012) está centrado principalmente en la generación de código y línea de productos de software en el dominio de la generación de mallas geométricas. Este trabajo resulta de gran interés para lo que se pretende lograr en este proyecto ya que toma una aplicación llamada Meshing Tool Generator y la utiliza para crear una herramienta que permite al usuario final la creación automática de productos, que corresponden a generadores de mallas con distintas funcionalidades Se observa un desarrollo de actividades tradicional, donde están implicadas las etapas de especificación de requisitos, diseño, implementación y pruebas, aunque pudo haber sido reemplazado por un método de desarrollo moderno como UP que se basa en UML donde cada fase se puede representar de un modo iterativo con los resultados desarrollados incrementalmente (Sommerville, 2005).

El trabajo mencionado fue desarrollado usando el lenguaje de programación C++, que al ser un lenguaje orientado a objetos permite la reutilización de código apoyada por la herramienta makefile.El trabajo que se desarrollará en esta tesis será en JAVA al tratarse de aplicaciones móviles así que se buscará una herramienta similar a la utilizada en el trabajo de Díaz Ríos (2012), pero orientada al lenguaje de programación JAVA. Aunque el trabajo está enfocado a un dominio distinto, se pueden utilizar los conceptos de línea de productos de software y generador de código para el presente proyecto.

En el trabajo de Vicente, Toledo, Fernández, & Sánchez (2006) se muestra un enfoque de desarrollo para sistemas mixtos Software/Hardware, donde se utiliza el desarrollo basado en componentes y la generación automática de software para el dominio de los Sistemas de Procesamiento de Información Visual (VIPS). En este trabajo se muestra el desarrollo de la herramienta IP-CoDER que permite la construcción incremental de VIPS. Aquí se propone el desarrollo de sistemas más flexibles con diseños reutilizables usando una metodología en espiral y fácilmente adaptable, que es parte de lo que se pretende alcanzar en este proyecto de título aunque abordado con una metodología distinta. Se puede observar en este trabajo que es de gran importancia la separación del Hardware con el Software, debido a que se busca una independencia entre ambos creando algoritmos sencillos de bajo nivel para la plataforma Hardware, y algoritmos más complejos para el procesamiento de Software. Esta es una buena idea si se piensa en la reutilización de código pensando en la variedad de fabricantes distintos de dispositivos. El proyecto actual solo se centra en dispositivos móviles Android; estos dispositivos al tener un desarrollo en Java permite no tener que dar mucha relevancia al fabricante de dispositivo que se utilice sino a la versión de Android sobre la cual se trabaje. La herramienta que se describe IP-CoDER es un buen ejemplo de reutilización debido a que se construye más como un entorno integrador de distintas herramientas existente para el desarrollo de las VIPS.

4. Desarrollo

4.1. Introducción

El proyecto fue desarrollado utilizando el método UP que se describe en la sección 1.6.2, donde fueron elegidas dos iteraciones sobre las que se fue desarrollando la aplicación Generador App. Previo al desarrollo se realizó el estudio del dominio del tema, es decir, todo lo relacionado con reuso, LPS, generador de aplicaciones y administración de emergencia, mientras que el estudio de la aplicación Mapa Móvil fue realizado en paralelo al desarrollo del proyecto. Uno de los puntos más importantes y que es necesario mencionar, es la búsqueda de una herramienta que permitiera la construcción de aplicaciones para Android, lo que es la base para la aplicación.

4.2. Herramienta para la construcción de aplicaciones.

4.2.1. Búsqueda de la herramienta para la construcción.

Lo más esencial en este proyecto fue buscar la forma de generar automáticamente aplicaciones, para esto se buscaron herramientas que permitieran mediante comandos generar programas, la primera idea fue la herramienta makefile que había conocido programando en lenguaje C bajo un sistema operativo Linux que permite determinar automáticamente que piezas o módulos de un programa necesitan ser recompilados, y ejecutar las órdenes apropiadas para éste (Aburruzaga García, 2011), aunque puede ser utilizado para otros lenguajes diferentes a C se descartó al encontrar otras herramientas más apropiadas para construir aplicaciones en el lenguaje Java para Android, entre estas herramientas esta Gradle y Apache Ant.

4.2.2. Apache Ant vs Gradle

Apache Ant

Ant (Another build tool) es una herramienta poderosa para la construcción de aplicaciones; está basada en java a diferencia de otras herramientas como Make o jam, esto hace que la herramienta sea portable y no dependa del sistema operativo donde se utilice (Bailliez et al., 2015). Ant es fácil de utilizar debido a que necesita un archivo de configuración XML con una estructura simple para definir los tareas que desea lograr con Ant; además, estas tareas pueden ser agrupadas fácilmente usando distintos objetivos (targets) para definir dependencias (Serrano & Ciordia, 2004).

Esta herramienta es la que utiliza ADT, que es un plugin para el IDE Eclipse que provee una interfaz para varias de las herramientas de Android SDK.

Gradle

Gradle, o más bien Gradle Build Tool, es una herramienta de construcción de aplicaciones moderna que poco a poco se abre paso para la construcción de aplicaciones. Es tan flexible como Ant y permite la importación de cualquier proyecto basado en Apache Ant convirtiendo un proyecto Ant en tareas Gradle nativas (Gradle User Guide Version 3.4.1, n.d.), el corazón de gradle es su lenguaje DSL (Domain Specific Language) que está basado en Groovy y permite crear Script para configurar la construcción de un proyecto en vez de archivos de configuración XML; además, previa configuración, puede ser llamado mediante la línea de comandos para su ejecución.

Esta herramienta es la que utiliza Android Studio, IDE oficial de para la creación de aplicaciones en esta plataforma, aunque Gradle no incluye por defecto las funcionalidades para Android, Google provee un plug-in para dar soporte lo cual permite la fácil configuración de un proyecto Android (Kousen, 2016).

La Tabla 6 muestra una comparación entre las herramientas de construcción estudiadas para el desarrollo de aplicaciones Android mediante línea de comandos.

Tabla 6: Comparación entre herramientas de contrucción.

	ANT	**GRADLE**
Flexibilidad	Proyectos creados en Apache ant pueden ser fácilmente exportados a Gradle.	Los proyectos Gradle no pueden ser exportados a Apache Ant.
Facilidad de uso	Archivos de configuración se crean con XML.	Archivos de configuración se crean con lenguaje propio basado en Groovy.
Provee interfaz para Android	Si, accediendo al archivo de configuración de ADT.	Se encontraba en estado beta al momento de elegir la herramienta.
Documentación Disponible	Si, en inglés	Si, en inglés

4.2.3. Elección de la herramienta

Ambas herramientas sirven para el desarrollo de aplicaciones Android, aunque la herramienta elegida fue Apache Ant por las siguientes razones:

1) La aplicación Mapa Móvil fue construida mediante ADT lo que permite una mayor compatibilidad al momento de re-construir los componentes.

2) Se puede modificar mediante un archivo de propiedades los atributos del archivo buil.xml que viene incluido con el plugin ADT para eclipse. Gradle trabaja con un lenguaje propio de script.

3) No existe soporte nativo de Gradle para las funcionalidades de Android, esto lleva a tener que usar Android Studio, herramienta que en el momento de la elección se encontraba en versión beta.

4) Debido a tener en 2010 una versión estable ADT (Android Developers, 2014). existe mayor información con respecto a utilizar Ant mediante los archivos de configuración de ADT.

Aunque la elección fue Ant, Gradle permite convertir proyectos Ant en Gradle; esta opción también está disponible en Android Studio, por lo que se puede migrar fácilmente de Ant a Gradle.

4.3. Resumen de iteraciones

Ya seleccionada la herramienta de construcción (Apache Ant), el siguiente paso consistió en buscar la manera de construir una aplicación mediante los componentes de Mapa Móvil con esta herramienta.

Se tomó la decisión de realizar dos iteraciones de UP para el desarrollo de la aplicación Generador de Aplicaciones, aunque en este documento solo se pondrá un mayor énfasis en la última iteración debido a que el ciclo de vida iterativo se basa en la ampliación y refinamiento sucesivos del sistema mediante las múltiples iteraciones (Larman, 1999), por lo que la iteración final contendrá el resultado mejor refinado. La ventaja principal de UP es que al ser un desarrollo por iteraciones se reduce los riesgos altos, ya que hay un progreso visible en las primeras etapas y en una retroalimentación temprana (Larman, 1999). Por esto, la primera iteración se centró en construir una aplicación capaz de construir la aplicación

Mapa Móvil con todos los componentes base (componentes básicos y opcionales) de ésta y como segundo objetivo principal el poder a partir de un mismo código lograr construir más de una aplicación; luego en la segunda iteración se logró conectar estos dos objetivos pudiendo construir a partir del código base de la aplicación Mapa Móvil tres aplicaciones: la primera es la aplicación Base que está compuesta por todos los componentes que constituyen Mapa Móvil, la segunda es la aplicación Basic Only que se componen solo por los componentes Bases y finalmente la última aplicación es la aplicación Original. Esta última se constituye por los componentes básicos de la aplicación, pero tiene la característica de utilizar como base de datos un componente alternativo llamado XMLLocalRepositoryA en lugar del componente DBLocalRepositoryA. Además de lograr esta conexión, en la segunda iteración se implementó la Interfaz de Usuario y se solucionaron errores dejados en la primera iteración.

4.3.1. Marco de Desarrollo

Aunque en este documento se muestren secuencialmente las etapas de requisitos, diseño e implementación, es importante mencionar que estas actividades no fueron llevadas a cabo secuencialmente, solo se escriben de esta manera para dar una mejor estructura al documento. La Tabla 7 muestra el Marco de Desarrollo donde están los artefactos elegidos en cada una de las disciplinas desarrollas tanto en el primer y segundo ciclo de desarrollo, y servirá para estructurar gran parte del desarrollo en este documento.

Tabla 7: Marco de Desarrollo del proyecto Generador de Aplicaciones

Disciplina	Artefacto
Modelado del Negocio	Lista Actor Objetivo
Requisitos	Modelo de Casos de Uso Glosario Diccionario de Datos Especificación complementaria
Diseño	Modelo de Diseño con diagramas de clase
Implementación	Construcción de Aplicación

4.4. Recolección de Requisitos

La necesidad que se intenta resolver es la de construir aplicaciones móviles para el dominio de Administración de Emergencias, usando un Generador de Código basado en los componentes reutilizables de la aplicación Mapa Móvil. Es por esto que pensando en las necesidades del proyecto surgieron los primeros requisitos que se modelaron en casos de usos, otros requisitos surgieron debido al diseño de la aplicación Mapa Móvil, que permite modificar ciertos parámetros los cuales fueron incluidos y finalmente surgieron requisitos relacionados con la configuración del proyecto. Esto se debe a que de manera complementaria se podrá en el futuro modificar la aplicación Generador App, para crear otra aplicación o conjunto de aplicaciones Android usando como base este Generador pudiendo cargar otro proyecto Android configurando previamente sus dependencias.

Los requisitos surgen de las necesidades de los actores; por esto es que en la Tabla 8 muestra la lista Actor-Objetivo, donde en el lado derecho se muestran 4 actores. Los primeros dos actores son actores principales que satisfacen sus necesidades a través de los objetivos que están a su la izquierda, mientras que los dos actores que siguen representan sistemas externos.

Lista Actor Objetivo

Tabla 8: Lista Actor Objetivo

Actor	Objetivo
Administrador	Seleccionar App Configurar parámetros App Generar App Obtener App Configurar Proyecto
Usuario	Seleccionar App Obtener App
Sistema de Construcción	Generar App
Sistema de Instalación de App	Obtener App

4.4.1. Actores

Administrador

Es un actores principal del sistema; él es quien administra la construcción de aplicaciones móviles Android. El principal interés de este actores es poder realizar la selección precisa y clara de los componentes de la aplicación, que éstos se encuentren disponibles al momento de construir la aplicación móvil, pudiendo configurar de manera sencilla los parámetros de la aplicación y que en un tiempo razonable pueda obtener la App correcta.

Usuario

El usuario representa al actor que le dará uso a las aplicaciones creadas mediante en generador de aplicaciones, no es su responsabilidad conocer sobre el procedimiento de construcción de su aplicación, aunque sí debe conocer las limitaciones. Su principal interés

es obtener de manera sencilla la aplicación construida y si es posible que esta aplicación pueda ser instalada en su dispositivo móvil.

Sistema de Construcción

Este actor representa el sistema externo encargado de construir la aplicación móvil, en este caso el sistema es Apache Ant con ADT, aunque puede ser otro. Su principal interés es que al momento de construir la aplicación estén disponibles los componentes que necesarios para generar la aplicación como los parámetros que solicite.

Sistema de instalación de la App

Este actor representa un sistema externo encargado de instalar la aplicación en el dispositivo móvil; le interesa que la aplicación ya se encuentre generada y esté disponible para la instalación. Este actor se considera de apoyo debido a que sus intereses no representan una prioridad, sino que dan un valor agregado.

4.4.2. Casos de Uso

En esta sección se realizará una breve descripción de los casos de uso que surgen de los objetivos de los actores involucrados. Aquí solo se muestran los actores involucrados y el escenario principal de éxito. Para una descripción detallada de los casos de uso, incluyendo los casos de uso de subfunción, revise el ANEXO A.

Seleccionar App

Describe el proceso de selección de los componentes que generar la aplicación móvil, la Tabla 9 muestra un resumen del caso de uso con los actores involucrados.

Tabla 9: Resumen Caso de Uso Seleccionar App.

Seleccionar App		
Actor Principal	:	Administrador
Personal involucrado e intereses:		
Personal Involucrado	**Interés**	
Administrador:	Desea que la selección sea precisa y clara para evitar el error humano al seleccionar la aplicación	
Usuario:	Desea que la selección de los componentes sea de acuerdo a su necesidad particular	
Afectado:	Que la aplicación facilite el control de la emergencia y la rápida llegada del personal al foco del incendio.	
Escenario Principal o de éxito (o flujo Básico) :		
1. El sistema despliega las App que es posible generar. 2. El Administrador selecciona la App que el Bombero desea generar. 3. El Administrador finaliza la selección. 4. El sistema Guarda de manera temporal la App seleccionada.		

Configura parámetros App

Presenta la configuración de los parámetros de la App que se generará. Estos parámetros pueden ser el servidor (HOST y PORT) con el que se conectará a la base de datos externas, la versión de Android del dispositivo donde se instalará la App y el modo de construcción, ya sea de Debug o Release. La Tabla 10 muestra un resumen del Caso de Uso.

Tabla 10: Resumen Caso de Uso Configurar Parametros de la aplicacion

Configura parámetros App		
Actor Principal	:	Administrador
Personal involucrado e intereses:		

Personal Involucrado	**Interés**
Administrador:	La información de los parámetros de la aplicación debe presentarse en forma clara y sin ambigüedades; le interesa poder configurar el HOST y PORT para que la App se pueda conectar a la base de datos externa, la versión del dispositivo móvil que se utilizará y si la App generada será de Release o Debug.
Usuario:	Le interesa que la App generada pueda utilizarse en su dispositivo móvil.

Escenario Principal o de éxito (o flujo Básico) :

1. El sistema despliega los parámetros de la App a configurar.
2. El Administrador ingresa el HOST utilizado para que la App se conecte a la base de datos externa.
3. El Administrador ingresa el PORT utilizado para que la App se conecte a la base de datos externa.
4. El Administrador elige la versión del dispositivo para el que se genera la App que puede ser “superior o igual a Android 14” o “inferior a Android 14”.
5. El Administrador selecciona el modo de construcción que puede ser debug o release.
6. El Administrador indica mediante una acción que ha finalizado la configuración de la App.
7. El sistema recolecta los parámetros de la App.

Generar App

Describe el proceso que se realizará para que la aplicación se conecte con el sistema de construcción de aplicaciones y genere un App; esto se muestra en la Tabla 11.

Tabla 11: Resumen Caso de Uso Generar App

Generar App		
Actor Principal	:	Administrador
Personal involucrado e intereses:		
Personal Involucrado		**Interés**
Administrador:		Le interesa obtener la App correcta, esto quiere decir que los parámetros seleccionados y la llave utilizada sea la correcta, información sobre la App que se generará e indicaciones de cuando el proceso de construcción ha finalizado.
Usuario :		Le interesa que el tiempo de generación de la aplicación móvil sea reducido y que la App que se genere sea la que solicitó.
Sistema de Construcción:		Sistema encargado de construir la aplicación móvil, le interesa generar la aplicación según los parámetros enviados por el Generador App y dar respuesta del resultado de la construcción.
Escenario Principal o de éxito (o flujo Básico) :		
1. El Administrador indica al sistema que se desea generar la App. 2. El sistema muestra la información relevante de la aplicación a generar; esta información puede incluir la App seleccionada, y los parámetros con los que se generará la aplicación móvil. 3. El sistema excluye los componentes (UC6) que no fueron seleccionados para generar la App. 4. El sistema añade los parámetros de la aplicación Móvil a la construcción. 4.1.El sistema añade el HOST. 4.2.El sistema añade el PORT.		

4.3.El sistema añade la versión de Android.
5. El sistema llama al Sistema de Construcción para construir la App enviándole el modo de construcción.
6. El sistema espera la confirmación del Sistema de Construcción que la App se terminó de construir.
7. El sistema recibe la confirmación de que la construcción ha finalizado con éxito.
8. El sistema de construcción envía la ruta de la aplicación generada y la ruta del resultado de la construcción al sistema.
9. El sistema muestra al Administrador información que la construcción finalizó.
10. El Administrador finaliza la generación de la aplicación.
11. El sistema incluye los componentes (UC7) que fueron excluidos para una futura generación de aplicaciones.

Obtener App

Describe el escenario de obtención de la aplicación mediante la instalación directa al dispositivo o manualmente como se describe en la Tabla 12.

Tabla 12: Resumen de Caso de Uso Obtener App

Obtener App		
Actor Principal	:	Administrador
Personal involucrado e intereses:		
Personal Involucrado		**Interés**
Administrador:		Le interesa obtener de manera sencilla la App construida.
Usuario :		Le interesa poder obtener la App e instalarla en el dispositivo.
Escenario Principal o de éxito (o flujo Básico) :		
1. El sistema despliega las opciones de postConstrucción, las cuales son “Mostrar APK en carpeta”, “Instalar en Dispositivo Conectado”, “Resultado de la Construcción”. 1.1.El sistema despliega la opción Mostrar APK en carpeta. 1.2.El sistema despliega la opción Instalar en Dispositivo Conectado. 1.3.El sistema despliega la opción Resultado de la Construcción. 2. El sistema carga la ruta de la App.		

3. El sistema carga la ruta del archivo que contiene el resultado de la construcción. 4. El Administrador selecciona la opción Mostrar APK en carpeta. 4.1.El Administrador entrega la aplicación móvil al bombero que solicito la App. 5. El Administrador selecciona la opción Resultado de la Construcción. 5.1.El Sistema muestra el resultado de la construcción en pantalla. 6. El Administrador finaliza la aplicación.

Configurar Proyecto

Describe el escenario de configuración del proyecto Android que contiene los componentes bases para la construcción de aplicaciones móviles. Ver Tabla 13.

Tabla 13: Resumen Caso de Uso Configurar Proyecto

Configurar Proyecto		
Actor Principal	:	Administrador
Personal involucrado e intereses:		
Personal Involucrado		*Interés*
Administrador	:	Realizar la configuración inicial del proyecto Android que contiene los componentes que generar las App; esto implica exportar el proyecto junto con las bibliotecas pertenecientes al proyecto, ya sean internas o externas en el caso de necesitarlas, la llave de debug y llave de release junto con los parámetros pertenecientes a la llave, y la ubicación del Android SDK.
Escenario Principal o de éxito (o flujo Básico) :		
1. El sistema despliega las opciones de configuración inicial. 2. El Administrador selecciona la ruta donde se encuentra la carpeta principal del proyecto “rutaProyecto” que contiene los componentes que generar las App . 3. El Administrador selecciona la ruta donde se encuentra la llave de debug “rutaLlaveDebug”. 4. El Administrador selecciona los datos de la llave de release. 4.1.El Administrador selecciona la ruta de la llave de release. 4.2.El Administrador ingresa el Alias de la llave de release.		

4.3. El Administrador ingresa el Password del Alias “AliasPass” de la llave de release.
4.4. El Administrador ingresa el Store de la llave de release.
4.5. El Administrador ingresa el Password del Store “StorePass” de la llave de release.

5. El Administrador ingresa la ruta de una biblioteca interna del proyecto.

El Administrador repite el paso 5 hasta que ingresen todas las bibliotecas internas del proyecto.

6. El Administrador ingresa la ruta de una biblioteca externa del proyecto.

El Administrador repite el paso 6 hasta que ingresen todas las bibliotecas externas del proyecto.

7. El Administrador ingresa la ruta del Android SDK.
8. El Administrador indica al sistema que ha terminado de ingresar los datos de configuración del proyecto.
9. El Sistema exporta los archivos del proyecto al directorio “proyecto”.
10. El Sistema exporta las Bibliotecas externas e internas.

La Figura 21 muestra el diagrama de casos de uso para la aplicación generador App; a la izquierda se muestran los actores principales y a la derecha los actores de apoyo.

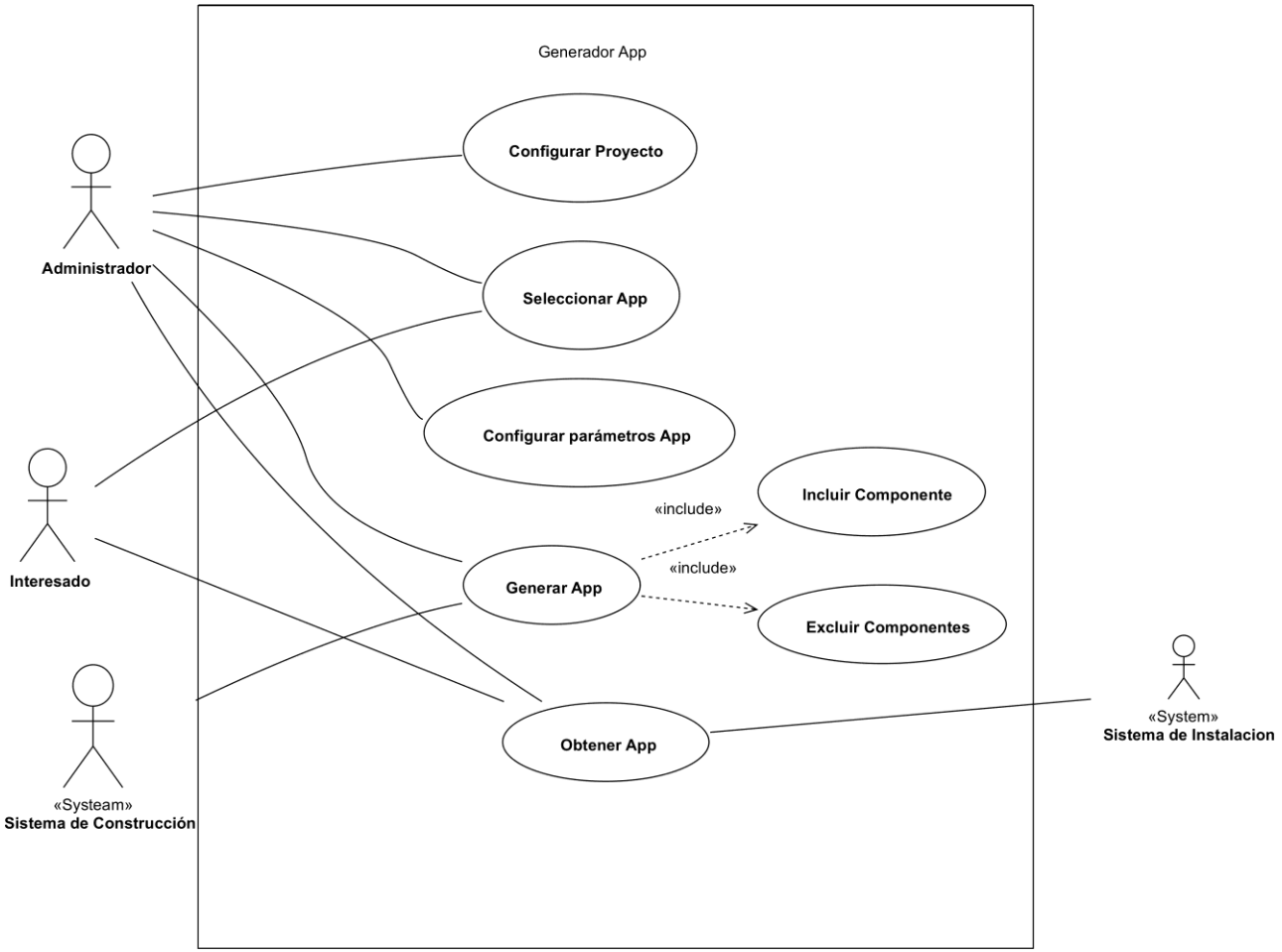


Figura 21: Diagrama de Casos de Uso de aplicación Generador App

4.5. Modelado del Diseño

Los diagramas de diseño se obtienen a partir de modelar los requisitos de la aplicación Generador App. Para una descripción más precisa, los diagramas aquí presentados fueron obtenidos con ingeniería inversa sobre la aplicación construida.

4.5.1. Organización de paquetes

La aplicación fue organizada en paquetes. En la Figura 22 se muestra el diagrama de paquetes de la aplicación; los paquetes principales se describen a continuación:

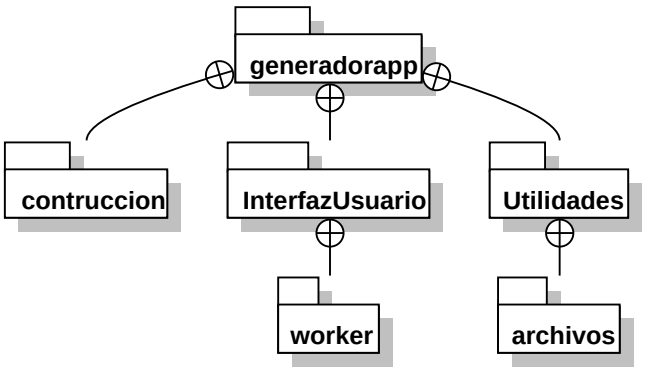


Figura 22: Diagrama de paquetes aplicación Generador App

generadorapp: Es el paquete principal. Aquí se encuentran los 3 subpaquetes que componen la aplicación Generador de aplicaciones. Contiene también las clases principales que construyen los objetos necesarios para cargar los componentes y los datos de la aplicación.

construcción: Este paquete contiene todas las clases que se preocupan de realizar tareas de construcción, ya sea configuraciones previas, la construcción misma o funciones de post-construcción. Gran parte de las clases que contiene este paquete realizan los procedimientos de conexión con los sistemas externos que se encuentran en la lista actor objetivo de la Tabla 8 en la sección 4.4.

InterfazUsuario: Como su nombre lo indica, este paquete se encarga de construir la interfaz de usuario. También posee un subpaquete encargado del manejo de eventos.

Utilidades: Paquete construido para todas las clases que contengan procedimientos de soporte. Por el momento contiene el paquete encargado de realizar todas las operaciones con archivos.

En la Tabla 14, se muestra la descripción de paquetes extraída de la documentación interna del proyecto en la que se da una descripción breve y precisa de las clases que contiene cada paquete.

Tabla 14: Descripción de paquetes extraídos de la documentación en javadoc de Generador App.

Paquetes	
Paquete	**Descripción**
generadorapp	Contiene los paquetes del proyecto, además de las clases principales que construyen los objetos necesarios para cargar los componentes y los datos de la aplicación.
generadorapp.contruccion	Contiene las clases con los procedimientos necesarios para realizar el proceso de construcción del proyecto Android, además de la clase que permite realizar la configuración inicial del proyecto.
generadorapp.InterfazUsuario	Contiene todas las clases que permiten la construcción de la interfaz de usuario del generador de aplicaciones.
generadorapp.InterfazUsuario.worker	Contiene todas las clases del proyecto que heredan de SwingWorker, clase que permite realizar las tareas más pesadas dentro de uno de sus procedimientos para liberar el EDT

	(Event Dispatch Thread) mientras se realiza dicha tarea.
generadorapp.Utilidades.archivos	Este paquete contiene las clases necesarias para trabajar con los distintos tipos de archivos en el proyecto, como son archivos de texto, archivos XML o archivos de propiedades.

4.5.2. Diagramas de clase

Las clase se organizaron en paquetes para representar la función dentro de la aplicación como se explican en la sección 4.5.1.

A continuación se presenta la descripción de las clases contenidas en cada paquete para luego mostrar el diagrama de clases correspondiente; estos diagramas omiten de manera intencional los atributos y procedimientos. Para una descripción detallada de cada clase puede revisar el ANEXO D.

Clases en el paquete generadorapp

El paquete generadorapp contiene las clases que realizan las principales tareas. En la Tabla 15 se muestra la descripción de estas clases, mientras que en la Figura 23 se muestra el diagrama de clases.

Tabla 15: Resumen de clases en el paquete generadorapp.

generadorapp	
Clase	**Descripción**
app	Esta clase contiene los atributos de la aplicación que se construirá.
archivoAsociado	Esta clase es utilizada para cargar las propiedades del componente.
biblioteca	Posee los atributos y procedimientos para el manejo de bibliotecas internas o externas de un proyecto Android.

componente	Clase que define un componente mediante sus atributos, como su identificador y componentes asociados y sus procedimientos, como son el de excluir o incluir el componente, etc...
datosApp	Esta clase contiene los datos asociados a la aplicación, como son los archivos y carpetas del proyecto, archivos de propiedades, rutas XPATH de archivos XML, codificación, entre otros.
GeneradorApp	Es la clase que llama a la ventana principal del proyecto y es la que se ejecuta para iniciar la aplicación.
llaveRelease	Clase que contiene los atributos de la llave de Release.
Ruta	Esta clase contiene las rutas de archivos utilizadas en este proyecto.

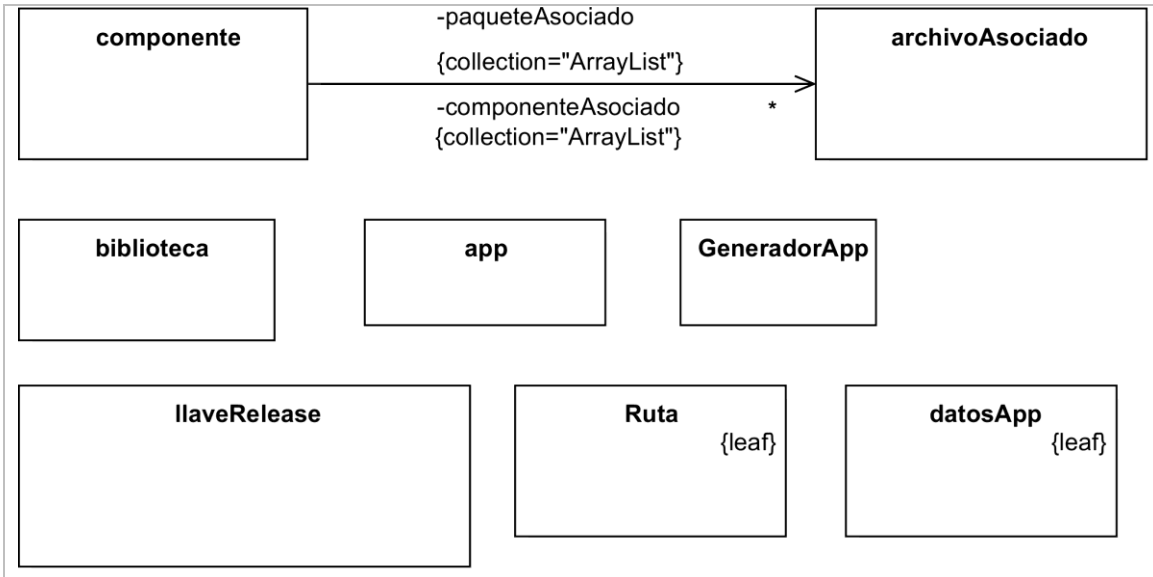


Figura 23: Diagrama de Clases para el paquete generadorapp

Clases en el paquete generadorapp.construccion

El paquete generadorapp.construccion contiene las clases que se encargan de la configuración y construcción de la App. En la Tabla 16 se describe cada una de las clases, mientras que en la Figura 24 se muestra el diagrama de clases de este paquete. Se puede observar que para realizar la configuración inicial, ésta depende de la llave de liberación de la App y de las bibliotecas externas y externas asociadas.

Tabla 16: Descripción de las clases del paquete generadorapp.construccion.

generadorapp.contruccion	
Clase	**Descripción**
ConstruirApp	Esta clase permite la configuración y construcción de la App (aplicación móvil).
ConfiguracionInicial	Contiene los procedimientos que permiten realizar la configuración inicial de la aplicación. Esta configuración solo se realiza una vez.
PostConstruccion	Contiene los procedimientos que realizan las tareas de post-construcción. Estas tareas pueden ser mostrar la carpeta de la apk, instalarla en un dispositivo, etc.

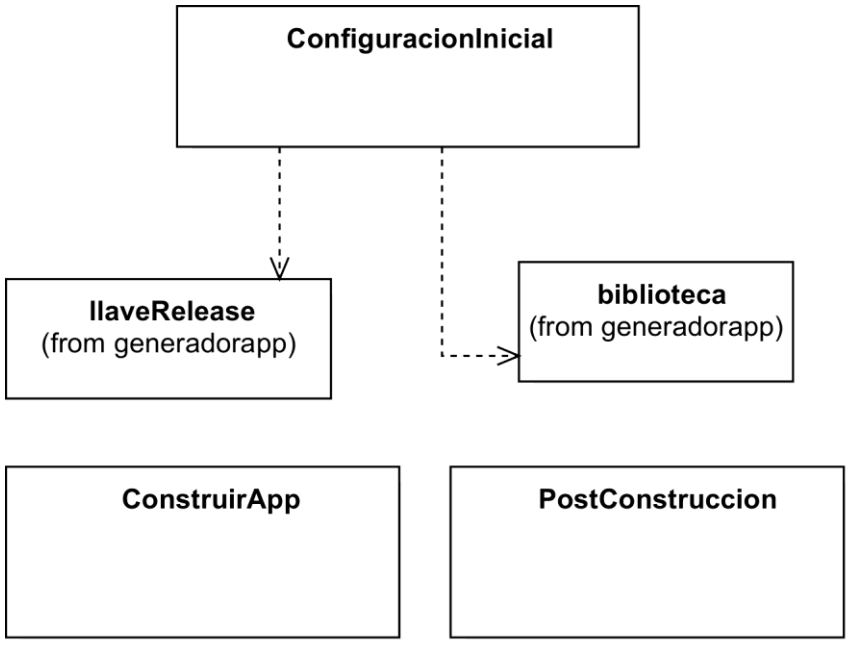


Figura 24: Diagrama de clases del paquete generadorapp.construccion.

Clases en el paquete generadorapp.InterfazUsuario

El paquete generadorapp.InterfazUsuario contiene las clases encargadas de crear la Interfaz de Usuario y recolectar los datos necesarios para la generación de la aplicación móvil. La descripción se muestra en la Tabla 17, mientras que el diagrama de clases se muestra en la Figura 25.

Tabla 17: Descripción de las clases del paquete generadorapp.InterfazUsuario

generadorapp.InterfazUsuario	
Clase	**Descripción**
ConfiguracionAppUI	Este JFrame se encarga de realizar configuraciones especificas a la aplicación móvil, estas configuraciones son: - El Servidor y Puerto con el que se conectará la aplicación

ConfiguracionInicialUI	Este JFrame está encargado de realizar la configuración inicial al proyecto Android.
GenerandoAppUI	Genera la aplicación Móvil (App) con los datos recolectados en los JFrame anteriores.
PantallaPrincipalUI	Muestra en pantalla las distintas App que se pueden generar y el botón que permite abrir el JFrame con la configuración inicial del proyecto.
postConstruccion	Este JFrame está encargado de mostrar las opciones de PostConstruccion de la App.

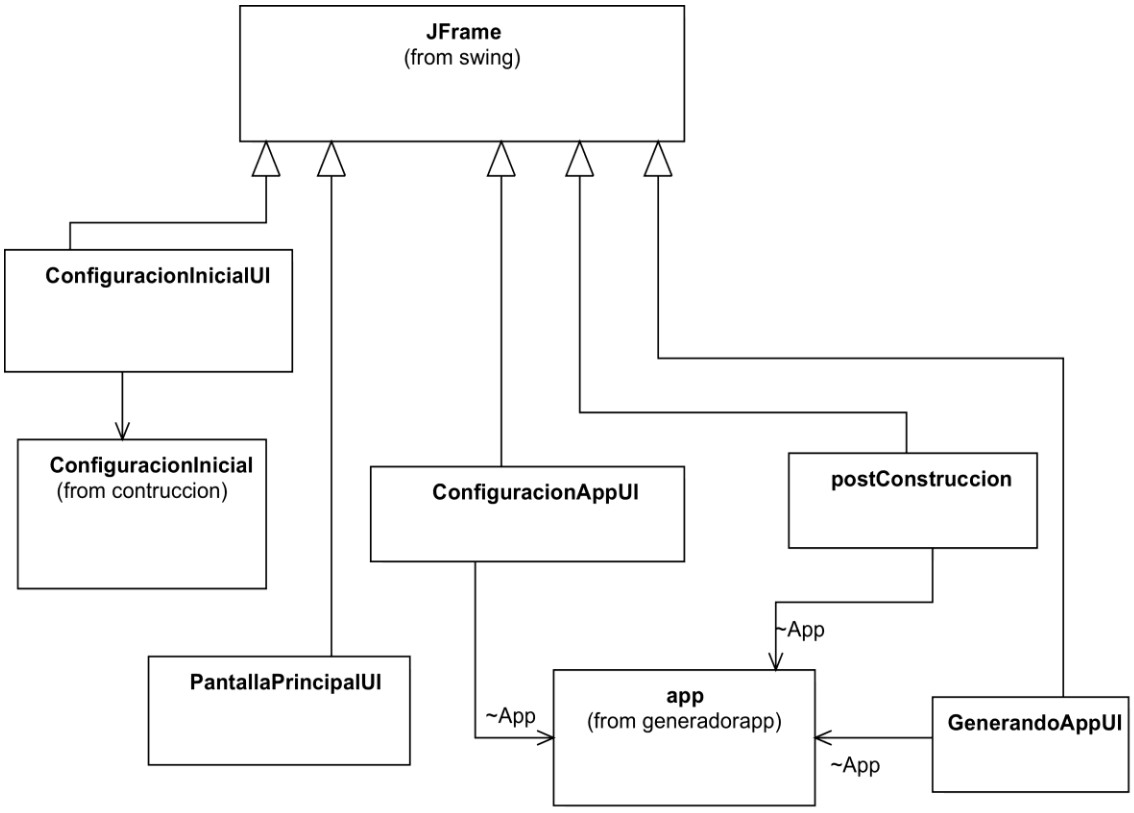


Figura 25: Diagrama de clases del paquete generadorapp.InterfazUsuario

Clases en el paquete generadorapp.InterfazUsuario.worker

El paquete generadorapp.InterfazUsuario contiene las clases encargadas de llamar internamente a las funciones de construyen y configuran la aplicación. De esta manera se libera el EDT, el cual se encarga de administrar la interfaz gráfica. Ver descripción de las clases en la Tabla 18 y el diagrama de clases en la Figura 26.

Tabla 18: Descripción de las clases en el paquete generadorapp.InterfazUsuario.worker

generadorapp.InterfazUsuario.worker	
Clase	**Descripción**
aplicarConfiguracionInicial	Esta clase hereda de SwingWorker y se utiliza para realizar la tarea de aplicar la configuración Inicial y liberar el EDT (Event Dispatch Thread) mientras la tarea se realiza.
construirApp	Inicializa los atributos necesarios para construir la aplicación. Además, carga una barra de progreso y un label que se utilizarán para informar del progreso de la aplicación en el JFrame que lo llamó; también carga dos botones que se habilitarán terminada la ejecución de la tarea.

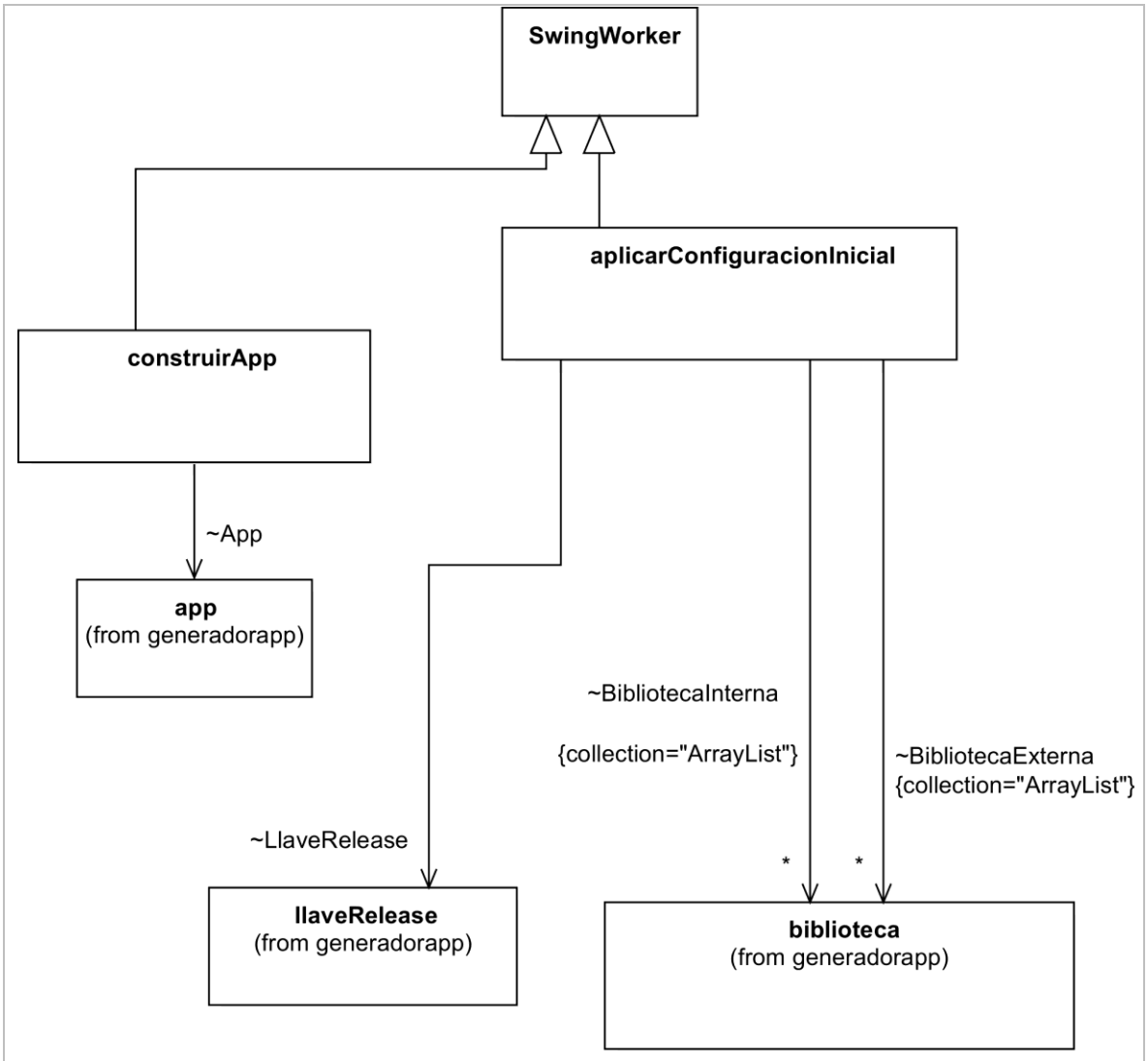


Figura 26: Diagrama de clases del paquete generadorapp.InterfazUsuario.worker

Clases en el paquete generadorapp.Utilidades.archivos

El paquete generadorapp.Utilidades.archiovos contiene todas las clases de soporte para el manejo de los distintos archivos usados en este proyecto, entre los cuales destaca el manejo de archivos XML y de propiedades que permiten rescatar las configuraciones del proyecto. Se muestran una descripción de las clases en la Tabla 19 mientras que el diagrama de clases se muestra en la Figura 27.

Tabla 19: Descripción de clases en el paquete generadorapp.Utilidades.archivos

generadorapp.Utilidades.archivos	
Clase	**Descripción**
Archivo	Esta clase permite realizar operaciones con archivos y carpetas; para esto utiliza objetos de la clase File, además de llamadas a script para realizar operaciones mediante el sistema operativo.
Atributo	Esta clase permite crear un atributo, un atributo esta compuesto por un nombre y un valor, es utilizado en archivos de configuración, propiedades o archivos XML.
etiquetaXML	Clase creada para trabajar con etiquetas XML; las etiquetas XML tienen un nombre, un conjunto de atributos, contenido y un nodo padre.
Propiedad	Esta clase es utilizada para creación y modificación de archivos de propiedad.
texto	Esta Clase contienen los procedimientos que permiten realizar modificaciones a archivos de texto.
xml	El objetivo de esta clase es poder realizar operaciones sobre documentos XML; estas operaciones consisten en eliminar, agregar o modificar.

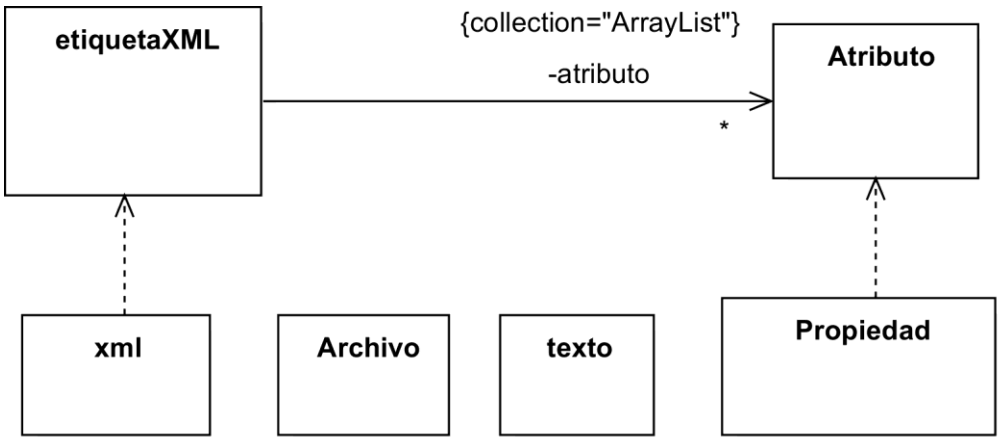


Figura 27: Diagrama de clases del paquete generadorapp.Utilidades.archivos

4.6. Implementación

4.6.1. Conexión de la aplicación Generador App con el sistema de construcción

El sistema de construcción se forma en base a dos componentes, el primero de ellos es Apache Ant que realiza el proceso de construcción de la aplicación y el segundo componente es ADT que cumple con dos labores. La primera labor es la de crear el archivo XML utilizado por Ant donde se realizan las tareas de construcción, este archivo es llamando por defecto Build.xml. La segunda labor es dar soporte a la construcción, esto mediante la llamadas realizadas desde el archivo Build.xml del proyecto Android al archivo de construcción ubicado en el directorio de ADT, donde se encuentran ubicadas las tareas de construcción, limpieza, entre otras. En el diagrama de actividad de la Figura 28 se muestran las acciones realizadas para realizar la construcción de la aplicación.

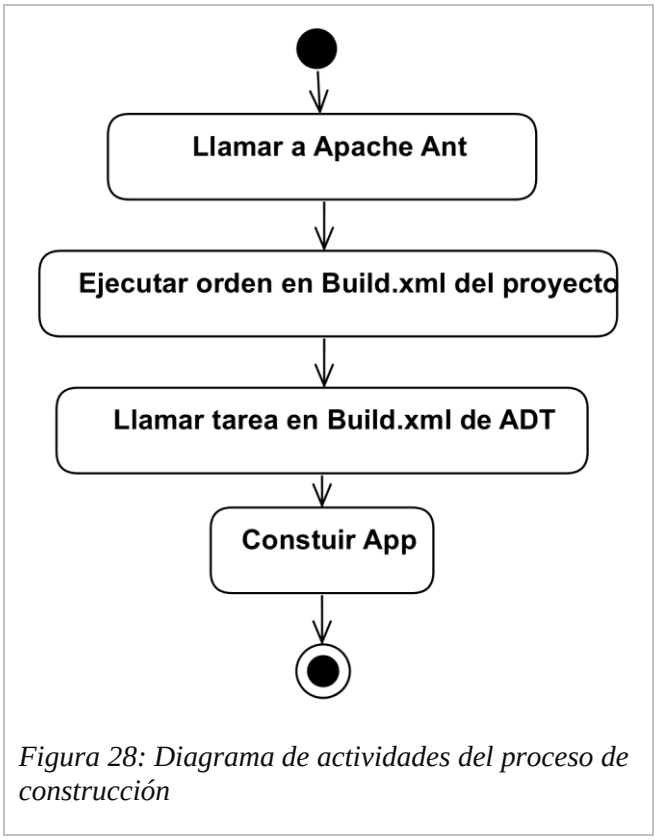


Figura 28: Diagrama de actividades del proceso de construcción

La herramienta utilizada (Ant) permite la construcción aplicaciones mediante la línea de comando, y para lograrlo utiliza un archivo de configuración XML (build.xml); en este archivo se crean las tareas (task) que desee realizar, en el caso de Android, se puede crear un archivo de configuración de manera automática sobre un proyecto utilizando ADT. Para estas configuraciones es necesario añadir a la variable de entorno PATH del sistema operativo de Ant y el SDK de Android. Para mayor información vea el ANEXO E que muestra el manual de instalación de la aplicación para el desarrollador.

Es necesario antes de continuar recordar que las conexiones externas y configuraciones se realizan automáticamente por la aplicación Generador App, por lo que aquí se presenta de manera explicativa cómo se realizaron estas conexiones. Los archivos que se presentan en esta sección no representan a los archivos reales del proyecto, sino que fueron reducidos para ser más explicativos.

Configuración Inicial

Antes de realizar cualquier tarea de construcción, es necesario que ADT genere el archivo de configuración Inicial. Esto se realiza mediante un comando en cual se muestra en la Figura 29; éste se encarga de actualizar el proyecto creando los archivos build.xml y local.properties necesarios para ejecutar el proyecto mediante Ant. Este comando debe aplicarse en las carpetas raíz de todas las bibliotecas internas del proyecto y en la carpeta raíz del proyecto. El archivo build.xml que se genera se muestra en la Figura 30, donde se han omitido los comentarios automáticos generados intencionalmente para mostrar con más claridad las tareas que realiza este archivo. Para una explicación mas detalla revise los comentarios que se encuentran en la misma Figura 30.

```
::Este comando crea un archivo de configuracion build.xml y el archivo
::local.properties si no existe.
android update project --path .
```

Figura 29: Comando para crear archivos de configuración del proyecto

```
<!-- Se indica la versión del programa y la codificación del archivo.
Por defecto la codificación es UTF-8. -->
<?xml version="1.0" encoding="UTF-8"?>

<!-- El nombre por defecto del programa representa a la clase principal,
mientras que la tarea por defecto (si no se indica otra tarea) es la de
mostrar la ayuda. -->
<project name="EmergenciesList" default="help">

<!-- Se indica el nombre de los archivos donde se buscarán las
propiedades de la construcción personalizadas por el usuario -->
    <property file="local.properties" />

    <property file="ant.properties" />

    <property environment="env" />
    <condition property="sdk.dir" value="${env.ANDROID_HOME}">
        <isset property="env.ANDROID_HOME" />
    </condition>

<!-- Se indica el archive desde donde se cargaran las propiedades del
proyecto, estas propiedades pueden ser la versión de Android utilizada
para construir el proyecto, sus bibliotecas internas,etc. -->
    <loadproperties srcFile="project.properties" />

    <!-- quick check on sdk.dir -->
    <fail
            message="sdk.dir is missing. Make sure to generate
local.properties using 'android update project' or to inject it through
the ANDROID_HOME environment variable."
            unless="sdk.dir"
    />

    <import file="custom_rules.xml" optional="true" />
    <!-- version-tag: 1 -->

<!-- Se importa el archive build.xml en el directorio del sdk de
Android, este archivo tiene las tareas de construcción, limpieza de
archivos,etc. necesarias para generar un archivo APK. -->

    <import file="${sdk.dir}/tools/ant/build.xml" />

</project>
```

Figura 30: Build.xml

Para terminar con la configuración, solo basta agregar la versión de Android y bibliotecas internas utilizadas al archivo project.properties. Este archivo se crea

manualmente, o en este caso particular mediante la aplicación generador App. En la Figura 31 se muestra el archivo de configuración generado para este proyecto.

```
#Fri Mar 10 13:43:04 CLST 2017
target=android-19
android.library.reference.2=../google-play-services_lib
android.library.reference.1=../android-support-v7-appcompat
```

Figura 31: project.properties

Llamada a apache Ant

Para realizar la llamada a Apache Ant desde la aplicación Generador App se realiza una llamada al sistema operativo. Esta llamada se encarga de ejecutar un script el cual realiza la tarea de construcción. En la Figura 32 se puede ver la estructura del script para la construcción del programa en modo debug, mientras que la Tabla 20 muestra la descripción de los comandos básicos para construir aplicaciones.

Tabla 20: Comandos básicos para la construcción de aplicaciones.

Comando		Descripción
ant clean		Esta orden se encarga de limpiar la carpeta bin que contiene todos los archivos generados en compilaciones anteriores.
ant debug		Se encarga de llamar a la tarea debug que construye la aplicación Android en modo Debug.
ant release		Se encarga de llamar a la tarea release que construye la aplicación Android en modo Release.
errorlevel		Comando del sistema operativo, indica la orden se ejecutó correctamente.

```
::Se eliminan los archivos binarios de la compilación anterior
call ant clean

::Se construye el proyecto en modo debug
call ant debug>debug.txt

if errorlevel 1 exit 1
```

Figura 32: Comandos para construir App en modo debug

Archivo de configuración

Cuando Ant ejecuta la orden en el directorio del proyecto, se busca en el archivo por defecto build.xml la tarea a realizar; si esta tarea no se encuentra, se busca en los archivos que han sido importados. En el caso de este proyecto, build.xml importa el archivo desde ADT, lo que se puede ver en la Figura 30, donde se puede ver el elemento `<import file="${sdk.dir}/tools/ant/build.xml" />`; esto indica que se exportarán todas las tareas ubicadas en este archivo, lo que quiere decir que las tareas de construcción, limpieza de directorios, etc. se buscarán en este archivo y se ejecutarán.

4.6.2. Construcción de las aplicaciones

Con la base para construir una aplicación Android, el siguiente paso es la generación de múltiples aplicaciones. Aquí cobra importancia el diseño de software basado en componentes.

La aplicación Mapa Móvil fue construida basada en componentes, lo que permite tomar una pieza de software y separarla del resto. La aplicación base es la aplicación que contiene tanto los componentes básicos como los opcionales. Cada componente opcional tiene asociadas clases y paquetes de clases que pueden ser incluidos o excluidos en la generación de una nueva aplicación, lo que quiere decir que los archivos asociados a estos componentes deben ser removidos de la aplicación base si no se desean usar. Esto puede

parecen un procedimiento de borrar o agregar archivos y carpetas, pero estos componentes poseen una interfaz de donde se deben desconectar antes de borrarlos. Esto se logra quitando de las interfaces las llamadas a dichos componentes, es decir, se deben identificar las interfaces asociadas a dichos componentes y borrar de estas interfaces todos las líneas de código donde se llame a alguna clase asociada al componente. Con esta idea se crearon las clases componente y archivoAsociado.

La clase componente se encarga de incluir o excluir el componente antes de que se construya la aplicación, esto lo logra identificando todos los archivos y carpetas asociadas a este componente.

La clase archivoAsociado es la que posee el nombre del archivo asociado, el paquete donde se encuentra este archivo y la acción que debe realizar con él. Esta acción puede ser la de mover el archivo en caso de tener que incluirlo o excluirlo de la aplicación móvil que se desea generar, o modificar alguna línea de código del archivo en caso de que éste represente una interfaz donde necesite removerse la utilización del componente.

La Figura 33 muestra la clase componente. A ésta se le asocian dos atributos del tipo archivoAsociado, los que son arreglos que guardan los componentes y paquetes asociados con el fin de realizar los procedimientos de incluir y excluir un componente.

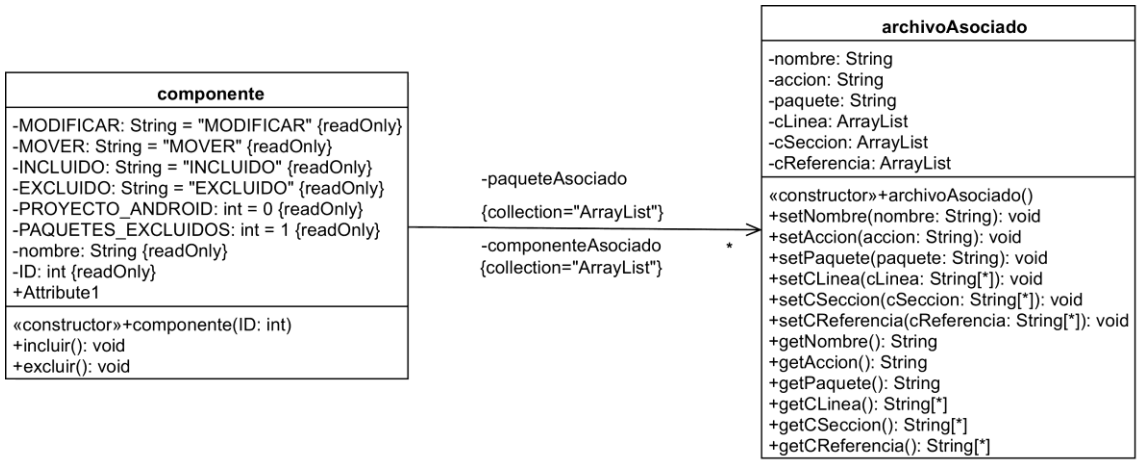


Figura 33: Clase componente

La lista de archivos asociados al componente se encuentra en un archivo de propiedades. Este archivo contiene un identificador del componente, la lista de archivos y sus atributos como lo muestra la Figura 34 que es un extracto del archivo de propiedades que excluye todos los componentes opcionales, al excluirlos, se genera la aplicación que solo posee los componentes básicos. Esta aplicación tiene por nombre Basic Only.

```
#Este archivo Contiene las clases y paquetes asociadas al componente
#que posee el mismo nombre del archivo.
clases.cantidad=12
paquetes.cantidad=7


clase.1.nombre=DigitalResourceTypes.java
clase.1.accion=MODIFICAR
clase.1.paquete=cl.ucsc.projectoftitle.newmobilemap.components.entities.t
ypes
clase.1.cs.cantidad=2
clase.1.cl.cantidad=0
clase.1.comentarioSeccion.1=PICTURE_SEC
clase.1.comentarioSeccion.2=VIDEO_SEC

paquete.1.nombre=digitalmakers
paquete.1.accion=MOVER
paquete.1.paquete=cl.ucsc.projectoftitle.newmobilemap.components

paquete.2.nombre=digitalopeners
paquete.2.accion=MOVER
paquete.2.paquete=cl.ucsc.projectoftitle.newmobilemap.components
```

Figura 34: Extracto del archivo de propiedades opcionales.properties

Para realizar los cambios en el archivo asociado al componente fue necesario identificar las interfaces dentro de cada archivo, lo que se logró creando identificadores, que indican la línea de código o la sección que se debe remover o agregar para incluir o excluir el componente respectivamente. Para esto se agregó una etiqueta en la interfaz, que contiene el nombre del componente seguido del sufijo “SEC” como se muestra en la Figura 35.

```
……..
import java.io.File;

public interface DigitalResourceTypes extends ResourceTypes{
……..
// /* VIDEO_SEC
    public interface Video{
        ……..
    }
//   VIDEO_SEC_FIN */
}
```

Figura 35: Ejemplo de etiquetado de interfaces.

4.6.3. Organización del generador de aplicaciones

Debido al manejo de archivos externo para guardar las configuraciones del proyecto, ejecutar la construcción, etc., se organizó el proyecto en carpetas, las que no representan la estructura de paquetes presentada en la sección 4.5.1, sino que representan la organización de los archivos asociados a la construcción de aplicaciones móviles. La Figura 36 muestra la estructura de carpetas del proyecto.

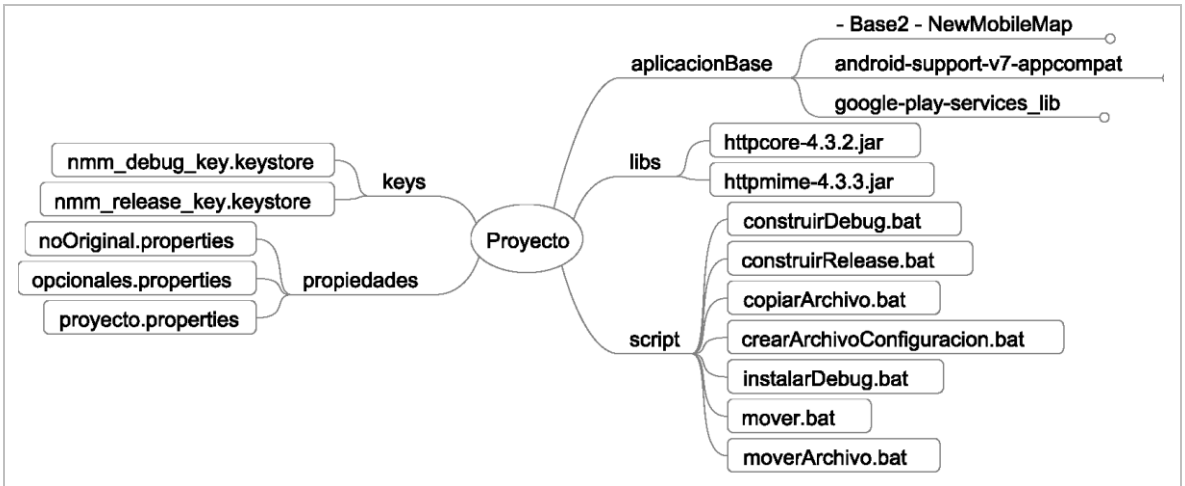


Figura 36: Estructura de Carpetas

La Tabla 21 muestra una breve descripción de la función de cada carpeta del proyecto. Cabe destacar que las primeras tres carpetas mostradas (aplicacionBase, keys y libs) son las carpetas que guardan todos los archivos del proyecto Android que se importa.

Tabla 21: Descripción de carpetas del proyecto

Carpeta	Descripción
aplicacionBase	Es la carpeta que contiene la aplicación base, es decir, la aplicación que contiene todos los componentes (básicos y opcionales); además se encuentran en esta carpeta las bibliotecas internas del proyecto.
keys	Esta carpeta guarda las claves de release y debug del proyecto.
libs	Guarda las bibliotecas externas necesarias para generar el proyecto.
propiedades	Se encuentran los archivos de propiedades del proyecto. El archivo proyecto.properties contiene las configuraciones generales del proyecto, mientras que los demás archivos contienen la información de paquetes y clases asociadas a un componente o una aplicación particular que se pueda generar a partir de éstos.
script	Contiene los archivos con secuencias de comandos.

4.6.4. Interfaz Gráfica

A continuación se presenta la interfaz gráfica del proyecto, la que fue creada con el paquete Swing de Java. Se implementó el aspecto Nimbus, lo que permite dar un aspecto multiplataforma a la aplicación.

La Figura 37 muestra la pantalla de configuración inicial del proyecto. A continuación se dará una breve explicación de su funcionamiento:

Importar Proyecto: En esta área se encuentra el botón "importar" que permite importar un proyecto Android generado mediante ADT a la aplicación Generador App. Aunque para este caso particular el proyecto importado es la aplicación Mapa Móvil con sus componentes base, podría ser importada otra aplicación Android y ser generada como

aplicación base. La carpeta del proyecto importado se mostrara en la zona que se encuentra sobre el botón.

Llave de Debug: Al presionar el botón "buscar" se abre una ventana que permite buscar la ubicación de la llave de debug. La ubicación se mostrará en la zona que esta sobre el botón.

Llave de Release: Aquí se ingresa la información correspondiente a la llave de release, es decir su Alias, Stores y sus respectivas contraseñas. Luego de ingresarlas se debe presionar el botón "Guardar" para que los datos ingresados sean guardados. El la zona inferior se encuentra el botón "buscar" que permite buscar la ubicación de la llave de release.

Bibliotecas Internas del Proyecto: Al presionar el botón "Añadir" se abre una ventana que permite buscar la carpeta raíz de la biblioteca interna, si existe más de una, entonces se presiona nuevamente el botón para ingresar otra biblioteca. La información del directorio desde donde se importará la biblioteca se muestra en la zona que esta arriba del botón.

Biblioteca Externa del Proyecto: Al presionar el botón "Añadir" se abre una ventana que permite buscar archivos de biblioteca, si el proyecto necesita más de una biblioteca externa entonces se debe presionar nuevamente el botón y añadir otra. La información del nombre del archivo y el directorio desde donde se importará, se muestra en la zona que esta arriba del bótón.

ANDROID SDK: Al presiornar el botón "Buscar" se indica en que directorio se encuentra el SDK de Android.

El botón "Aplicar Configuración" se encarga de importar el proyecto y todas sus bibliotecas, y generar los archivos de configuración inicial del proyecto, entre estos archivos se encuentra el archivo build.xml y archivos de propiedades del proyecto.

El botón "Salir" permite salir de esta ventana.

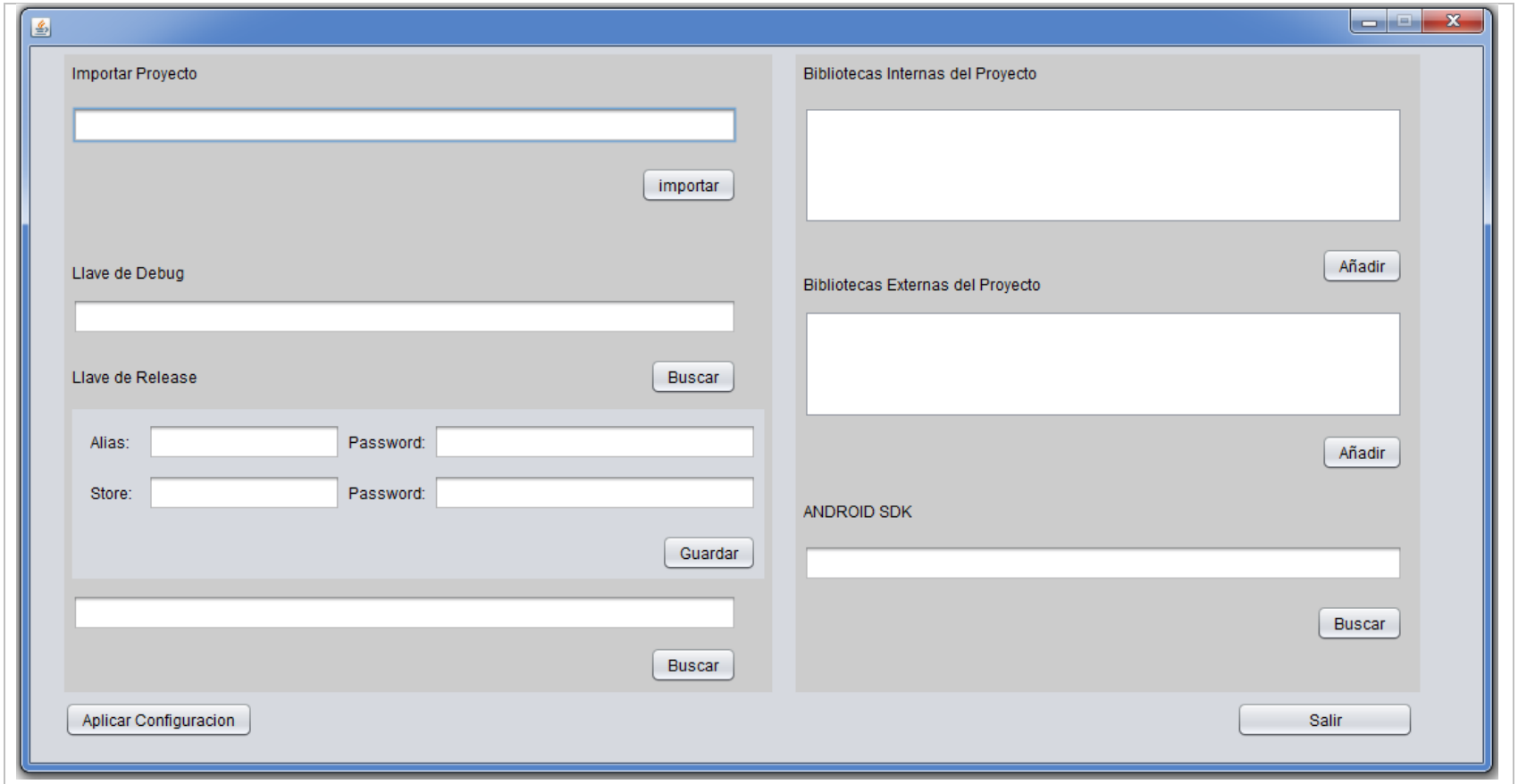


Figura 37: ConfiguracionInicialUI

La Figura 38 muestra la pantalla principal, que es la pantalla que se visualiza al iniciar la aplicación. Aquí es posible seleccionar las aplicaciones que pueden generarse. A continuación se dará una breve explicación de su funcionamiento:

APK Base: Permite generar la aplicación base, es decir, aquella que tiene tanto los componentes básicos como los opcionales.

Apk Only Basic: Permite generar la aplicación que solo posee los componentes básicos .

Apk Original: Permite generar la aplicación original, esta aplicación es similar a la aplicación "Onlyu Basic" con la diferencia que utiliza como alternativa al componente llamado XMLLocalRepositoryA en lugar del componente DBLocalRepositoryA .

El botón "Configuración Inicial" lleva a la ventana de la Figura 37 mientras que el botón "Siguiente" lleva a la ventana de la Figura 39.

Figura 38: PantallaPrincipalUI

La Figura 39 muestra la pantalla de configuración de la aplicación, aquí se seleccionan los parámetros específicos que pueden seleccionarse en mapa móvil, además de esto, el modo de construcción de la aplicación. A continuación se dará una breve explicación de su funcionamiento:

Elegir Servidor: Permite elegir el servidor al que se conectará la aplicación móvil generada para la conexión a la base de datos externa. Al hacer click en "servidor" se permite ingresar el host y puerto del servidor, mientras que si selecciona "Elegir servidor por defecto" se elegirá el servidor por defecto configurado en la aplicación.

Elegir versión de Android: Esta opción es heredada de los componente base de la aplicación Mapa Móvil, permite seleccionar si la aplicación generada será para dispositivos con Android Inferior a la versión 14 o Android con la versión 14 o Superior. Solo se a probado con éxito en dispositivos con Android 14 o Superior.

Modo de Construcción: Permite elegir entre construir la aplicación en modo Debug o Release.

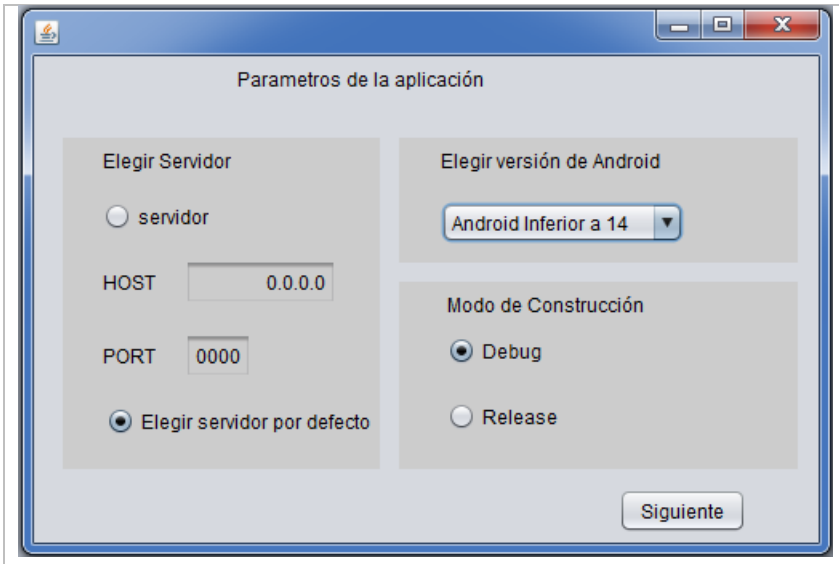


Figura 39: ConfiguracionAppUI

La pantalla que permite generar la App se muestra en la Figura 40, esta lanza una ventana donde se muestra el progreso de la construcción. A continuación se dará una breve explicación de su funcionamiento:

Generar App: Al presinar el botón “Generar App” la aplicación Generador App comienza con el proceso de construcción, en este momento se excluyen los componentes que no se desean utilizar en la aplicación y se muestra la información de la aplicación que se genera.

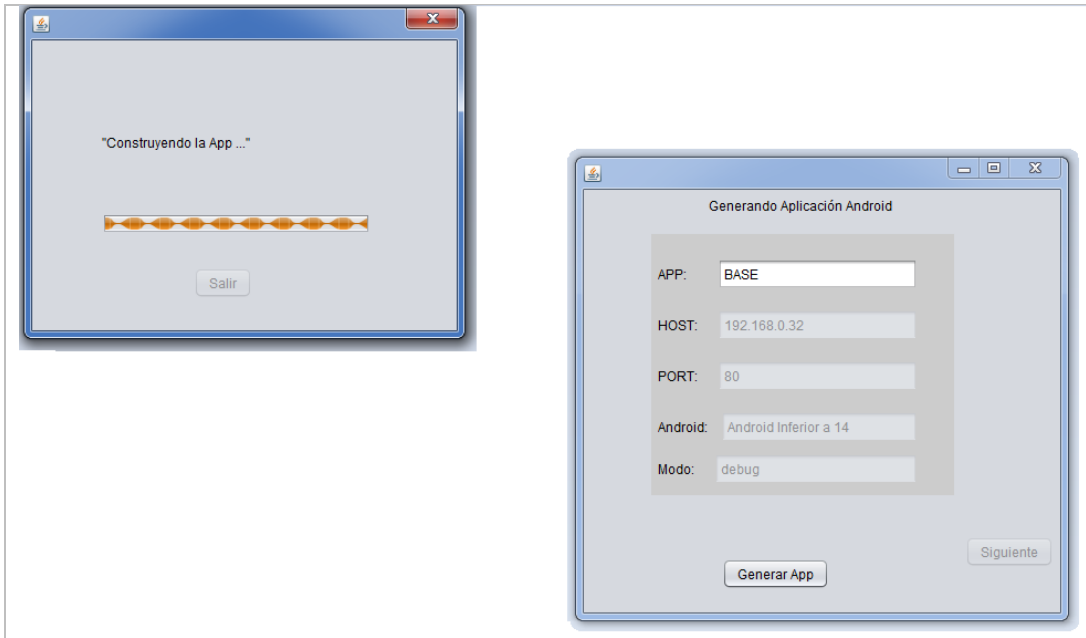


Figura 40: GenerandoAppUI

La Figura 41 muestra las opciones luego de finalizada la construcción. A continuación se dará una breve explicación de su funcionamiento:

Mostrar Apk en Carpeta: al presionar este botón se mostrará la carpeta con los archivos binarios generados, donde se encuentra el archivo de instalación con extensión apk que puede instalarse en el disposito.

Instalar en Dispositivo: Permite la instalación de la aplicación en modo debug en un dispositivo que se encuentre conectado al computador donde se ejecuta la aplicación Generador App.

Resultado Construcción: Muestra un archivo con el resultado de la construcción, en caso de existir un error al construir la aplicación, este archivo no mostrará la información de construcción exitosa.

Salir: Permite salir de la aplicación.

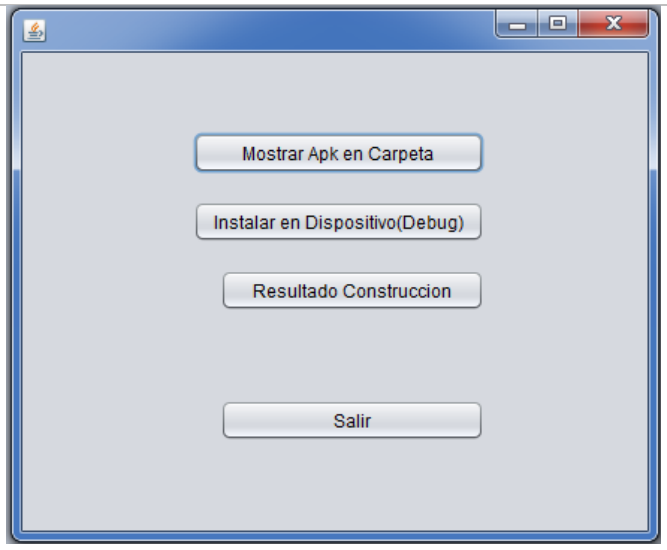


Figura 41: postConstruccion

4.7. Prueba de aplicaciones generadas.

Las aplicaciones fueron probadas en los dispositivos que se muestran en la Tabla 22, donde se realizaron las siguientes pruebas.

Prueba de instalación

Se comprobó que las aplicaciones generadas pudieran ser instaladas en los dispositivos señalados.

Prueba de funcionamiento

Se revisaron aleatoriamente funcionalidades ofrecidas por las aplicaciones en los distintos dispositivos.

Concordancia de la aplicación

Se comprobó que la aplicación móvil generada por Generador App correspondiera con la aplicación que se instaló en el dispositivo.

Tabla 22: Dispositivos utilizados para las pruebas

Marca	Modelo	Android
Motorola	Moto E	4.4.2
Alcatel	7055a	4.4.2
LG	LG-D335	4.4.2
SONY	C1504	4.1.1
Motorola	Moto G 1	5.1
Azumi	A50c+	4.4.2

Es importante destacar que las pruebas se realizaron para verificar el correcto funcionamiento del generador, por lo cual, se omitieron en las pruebas los errores de comportamiento producidos por algún componente, como la falla de la aplicación cuando no existe ninguna emergencia en curso, que se producía en la aplicación base cuando se

construía directamente de ADT sin utilizar el generador. El "✓" representa que la aplicación funciono con éxito y el "–" represeta que no pudo ser probada la aplicación en el dispositivo.

Tabla 23: Dispositivo Motorola Moto E

Dispositivo				
Marca	:	Motorola		
Modelo	:	Moto E		
Tipo	:	Smartphone		
Versión de Android	:	4.4.2.		
Pruebas		Instalación	Concordancia	Funcionamiento
Base		✓	✓	✓
Basic Only		✓	✓	✓
Original		✓	✓	✓

Tabla 24: Dispositivo Alcatel One Touch Hero 2C

Dispositivo				
Marca	:	Alcatel		
Modelo	:	7055a		
Tipo	:	Smartphone		
Versión de Android	:	4.4.2.		
Pruebas		Instalación	Concordancia	Funcionamiento
Base		✓	✓	✓
Basic Only		✓	✓	✓
Original		✓	✓	✓

Tabla 25: Dispositivo LG L Bello Dual

Dispositivo				
Marca	:	LG		
Modelo	:	LG-D335		
Tipo	:	Smartphone		
Versión de Android	:	4.4.2.		
Pruebas		Instalación	Concordancia	Funcionamiento
Base		✓	✓	✓
Basic Only		✓	✓	✓
Original		✓	✓	✓

Tabla 26: Dispositivo Sony Xperia E

Dispositivo				
Marca	:	SONY		
Modelo	:	C1504		
Tipo	:	Smartphone		
Versión de Android	:	4.1.1.		
Pruebas		Instalación	Concordancia	Funcionamiento
Base		✓	✓	✓
Basic Only		✓	✓	✓
Original		✓	✓	✓

Tabla 27: Dispositivo Motorola G

Dispositivo				
Marca	:	Motorola		
Modelo	:	Moto G 1		
Tipo	:	Smartphone		
Versión de Android	:	5.1.		
Pruebas		Instalación	Concordancia	Funcionamiento
Base		✓	✓	✓
Basic Only		-	-	-
Original		-	-	-

Tabla 28: Dispositivo Azumi A50C+

Dispositivo				
Marca	:	Azumi		
Modelo	:	A50c+		
Tipo	:	Smartphone		
Versión de Android	:	4.4.2.		
Pruebas		Instalación	Concordancia	Funcionamiento
Base		✓	✓	✓
Basic Only		✓	✓	✓
Original		✓	✓	✓

Las primeras aplicaciones construidas por Generador App presentaban problemas con los caracteres especiales, como tildes o la letra ñ. Este error fue corregido buscando la codificación usada por los archivos de código fuente de los componentes e indicándola al momento de generar las aplicaciones.

Finalmente se muestran algunas capturas de las aplicaciones construidas mediante el programa Generador App para los dispositivos Motorola Moto E y Azumi A50c+.

La Figura 42 y la Figura 43 muestran capturas de la aplicación base enviando un recurso digital en el dispositivo Motorola Moto E.

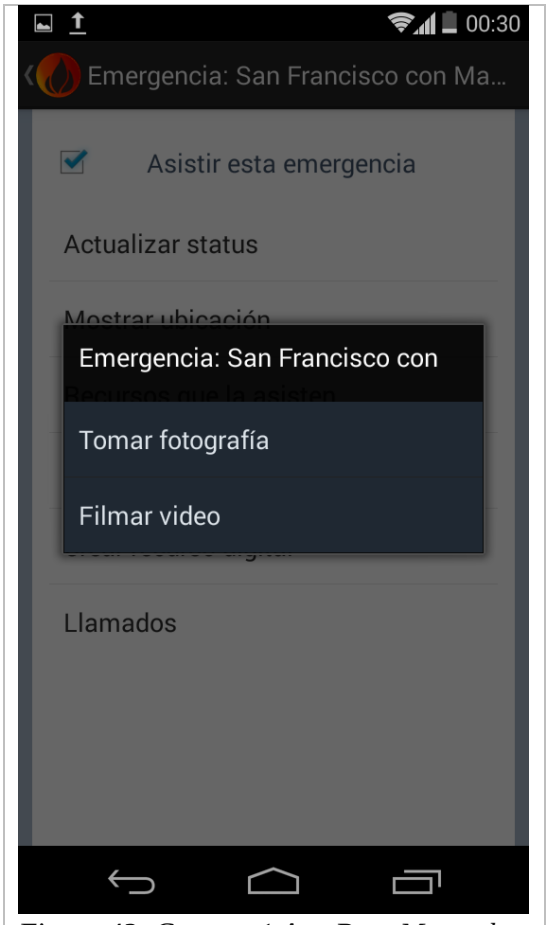


Figura 42: Captura 1 App Base Motorola Moto E

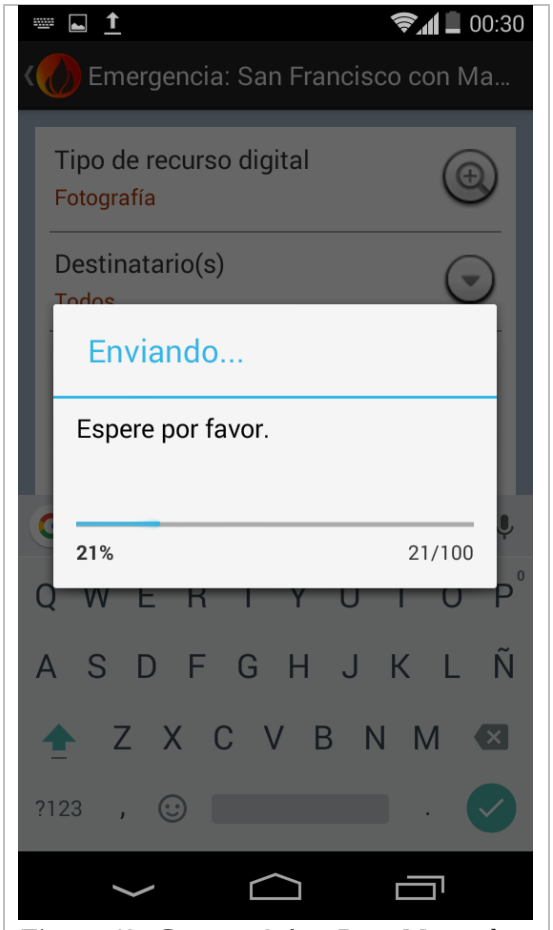


Figura 43: Captura 2 App Base Motorola Moto E

En la Figura 44, Figura 45 y Figura 46 se muestran capturas de la aplicación Base construida por el generador en el dispositivo Azumi A50c+.

Figura 44: Captura 1 App Base en Azumi A50c+

Figura 45: Captura 2 App Base en Azumi A50c+

Figura 46: Captura 3 App Base en Azumi A50c+

En la Figura 47, Figura 48 y Figura 49 se muestran capturas de la aplicación Basic Only construida por el generador en el dispositivo Azumi A50c+.

Al comparar las Figura 44 y Figura 47 se observa que en la primera existe un menú más grande que contiene las opciones “Crear recurso digital”, “Actualizar status”, “Puntos de Interés” y “Recursos digitales” que no se encuentran disponibles en la segunda Figura mencionada. Esto se debe a que estas opciones representan componentes opcionales de la aplicación.

Figura 47: Captura 1 App Basic Only en Azumi A50c+

Figura 48: Captura 2 App Basic Only en Azumi A50c+

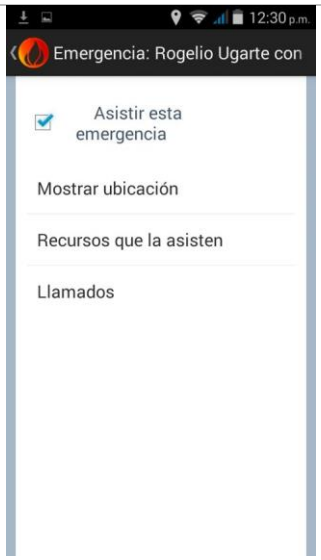


Figura 49: Captura 3 App Basic Only en Azumi A50c+

En la Figura 50, Figura 51 y Figura 52 se muestran capturas de la aplicación Original construida por el generador en el dispositivo Azumi A50c+.

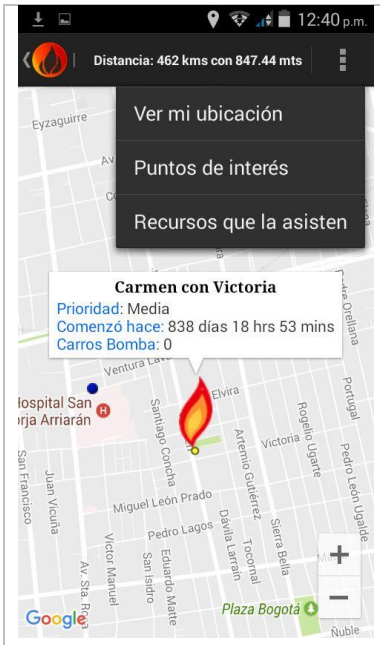


Figura 50: Captura 1 App Original en Azumi A50c+

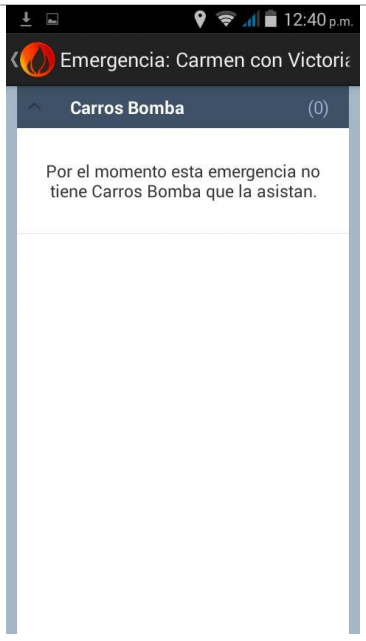


Figura 51: Captura 1 App Original en Azumi A50c+

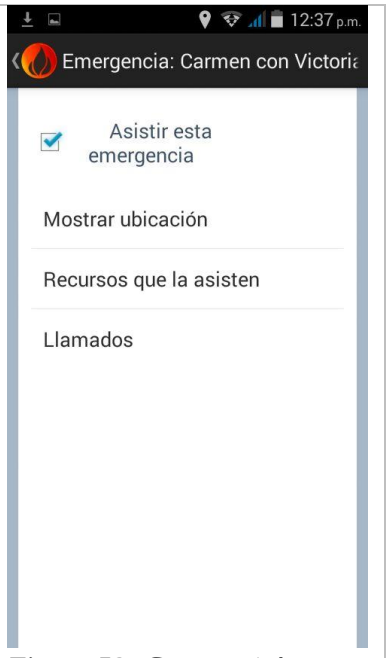


Figura 52: Captura 1 App Original en Azumi A50c+

Se observa de la Figura 47 a la Figura 52 la aplicación Basic Only y la aplicación Original, aunque son similares, la gran diferencia entre ellas es que la aplicación Original sólo posee el recurso Carro de Bomba; además, la aplicación original no posee la capacidad de conectarse a una base de datos externa debido a que posee una base de datos interna desde donde obtiene las emergencias previamente ingresadas.

5. Conclusiones

El objetivo principal de este proyecto consistió en desarrollar un generador de aplicaciones móviles para el dominio de la administración de emergencia, lo cual fue posible lograr mediante componentes previamente desarrollados.

A continuación se muestran las conclusiones y recomendaciones obtenidas del desarrollo de los objetivos específicos señalados en la sección 1.3.

El estudio de los componentes y la aplicación Mapa Móvil permitió conocer la manera en que estaban construidos los componentes identificando principalmente los componentes bases y los opcionales que permitieron establecer las bases para la creación de las aplicaciones; además, con esto se pudo buscar la herramienta más adecuadas para construirlos.

Aunque la construcción de los componentes es robusta, lo cual permitió identificar las interfaces de los componentes para poder agregarlos o excluirlos de la creación de una aplicación, fue difícil identificar qué componentes implementaban alguna de las características del modelo presentado en la Figura 17, esto debido a que el nombre del componente que lo implementa tiene un nombre que lo identifica para su implementación, pero no como una característica. Un ejemplo de esto es el manejo de recursos, que se realizó separándolos en recursos estáticos y dinámicos. Esto no facilitaba el poder identificarlos como un recurso humano o material, que son las características presentes en el modelo.

El desarrollo de la aplicación Generador App muestra que es posible construir un generador de aplicaciones móviles a partir de componentes creados para una línea de producto de software. Para esto, los componentes software deben tener en su documentación identificadas las interfaces que conectan a estos componentes, sobre todo en los componentes opcionales, debido a que éstos deben ser agregados o removidos durante la creación de una nueva aplicación.

Se puede afirmar que el desarrollo de los componentes software puede ser totalmente independiente del desarrollo del generador de aplicaciones, aunque se necesita primero tener el desarrollo de los componentes, esto debido a que existen distintas herramientas que se pueden utilizar para la creación de generador, dependiendo de cómo fueron construidos y debido a que en el desarrollo del generador se necesita tener identificados como mínimo los componentes bases y algún componente opcional que sirva para estudiar cómo se realizará el acoplamiento de este componente en la aplicación.

El probar las aplicaciones generadas, muestra que éstas no presentaron problemas de funcionalidad visible durante las pruebas, sólo un error de codificación en los archivos fuentes por parte del generador, se observa que es difícil producir este tipo de errores al crear el generador; éstos son más habituales en la construcción de los componentes.

Finalmente, se observó que las aplicaciones generadas funcionan en dispositivos de distintas marcas y modelos para versiones de Android desde la 4.1.1 a la 5.1.

6. Referencias bibliográficas

Aburruzaga García, G. (2011). Make. Un programa para controlar la recompilación. Retrieved from http://hdl.handle.net/10498/13276

Alonso, A., Artime, I., & Rodríguez, M. (2011). Dispositivos móviles. *Universidad de Oviedo*. Retrieved from http://isa.uniovi.es/docencia/SIGC/pdf/telefonia_movil.pdf

Android Developers. (2014). ADT Plugin. Retrieved March 5, 2017, from https://developer.android.com/studio/tools/sdk/eclipse-adt.html

Bailliez, S., Ken Barozzi, N., Bergeron, J., Bodewig, S., Chanezon, P., Duncan Davidson, J., … Strong, C. (2015). Apache Ant™ 1.9.6 Manual. Retrieved from https://ant.apache.org/manual/

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (1997). Extensible markup language (XML). *World Wide Web Journal*, *2*(4), 27–66. Retrieved from http://www.w3pdf.com/W3cSpec/XML/2/REC-xml11-20060816.pdf

Build Script Basics - Gradle User Guide Version 3.4.1. (n.d.). Retrieved March 5, 2017, from https://docs.gradle.org/3.4.1/userguide/userguide.html

Cambridge Dictionary, C. (2016). Cambridge Dictionary. *Meaning*, (entry 124), 138–138. Retrieved from http://dictionary.cambridge.org/

Caro, C. M., Ramos, A. N., & Barceló, A. V. (2002). *Introducción a la programación con orientación a objetos*. Prentice Hall. Retrieved from https://www.researchgate.net/profile/Alfonso_Nino/publication/31737252_Introduccion_a_la_programacion_con_orientacion_a_objetos_C_Munoz_Caro_A_Nino_Ramos_A_Vizcaino_Barcelo/links/55a3d63a08aed99da24d2ed2.pdf

Christensson, P. (2010). SDK (Software Development Kit) Definition. Retrieved April 8, 2017, from https://techterms.com/definition/sdk

Clements, P. C., & Northrop, L. M. (2002). *Framework for Software Product Line Practice*. Retrieved from http://www.sei.cmu.edu/productlines/tools/framework/

Czarnecki, K., & Eisenecker, U. W. (1999). Components and Generative Programming. In O. Nierstrasz & M. Lemoine (Eds.), *Software Engineering --- ESEC/FSE '99: 7th European Software Engineering Conference Held Jointly with the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering Toulouse, France, September 6--10, 1999 Proceedings* (pp. 2–19). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-48166-4_2

Díaz, Ó., & Trujillo, S. (2010). Líneas de producto software. *Fábricas de Software: Experiencias, Tecnologías Y Organización*, *2*, 1–14.

Díaz Ríos, V. N. (2012). *Desarrollo de una línea de productos de software de generación de mallas geométricas*. Universidad de Chile. Retrieved from http://www.repositorio.uchile.cl/handle/2250/112515

Engelbrecht, A., Borges, M. R. S., & Vivacqua, A. S. (2011). Digital tabletops for situational awareness in emergency situations. *Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 669–676. https://doi.org/10.1109/CSCWD.2011.5960190

Forni, A. A., & van der Meulen, R. (2016). Gartner Says Five of Top 10 Worldwide Mobile Phone Vendors Increased Sales in Second Quarter of 2016. Retrieved February 23, 2017, from http://www.gartner.com/newsroom/id/3415117

Frakes, W. B. (2005). Software reuse research: status and future. *IEEE Transactions on Software Engineering*, *31*(7), 529–536. https://doi.org/10.1109/TSE.2005.85

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1996). Design Patterns: Elements of Reusable Software. *Addison-Wesley Professional Computing Series*, 395. https://doi.org/10.1093/carcin/bgs084

Group, O. M. (2010). OMG Unified Modeling Language TM (OMG UML), Superstructure v.2.3. *InformatikSpektrum*, *21*(May), 758. https://doi.org/10.1007/s002870050092

Jacobson, I., Rumbaugh, G., Jacobson, J., Booch, G., & Rumbaugh, J. (2000). *El proceso unificado de desarrollo de software/The unified software development process*. Pearson Educación.

Java. (2014). Acerca de Java. Retrieved April 8, 2017, from https://www.java.com/es/about/

Kimmel, P. (2006). *UML Demystified* (1st ed.). New York, NY, USA: McGraw-Hill, Inc.

Kousen, K. (2016). *Gradle Recipes for Android*. O'Reilly.

Larman, C. (1999). *UML y Patrones*. Pearson.

Lecca, E. R. (2007). Programación genérica en C ++ , usando Metaprogramación. *Industrial Data*, *10*(1), 80–87.

Monares, A., Ochoa, S. F., Pino, J. A., & Herskovic, V. (2012). Improving the initial response process in urban emergencies. In *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2012* (pp. 379–386). https://doi.org/10.1109/CSCWD.2012.6221846

Montilva, A. (2006). Desarrollo de Software Basado en Líneas de Productos de Software. *IEEE Computer Society*, 1–34.

Muñoz Caro, C., Niño Ramos, A., & Vizcaino Barceló, A. (2003). *Introducción a la programación con con Orientación a Objetos*. *Universidad de Castilla-La Mancha*. Retrieved from https://www.researchgate.net/profile/Alfonso_Nino/publication/31737252_Introduccion_a_la_programacion_con_orientacion_a_objetos_C_Munoz_Caro_A_Nino_Ramos_A_Vizcaino_Barcelo/links/55a3d63a08aed99da24d2ed2.pdf

Nolasco Valenzuela, J. S. (2014). *Desarrollo de aplicaciones móviles con Android* (Vol. 11). Macro. Retrieved from http://www.editorialmacro.com/media/custom/upload/978-612-304-244-8.pdf

Ormeño, E. (2015). *Construcción de Componentes Reutilizables para la Elaboración de Aplicaciones Móviles que Apoyen el Trabajo Colaborativo en Emergencias*.

Universidad Católica de la Santísima Concepción.

Pressman, R. S. (2006). *Ingeniería Del Software: Un Enfoque Práctico* (Séptima ed). McGraw-Hill Interamericana.

Rossel, P. O., Herskovic, V., & Ormeño, E. (2016). Creating a family of collaborative applications for emergency management in the firefighting sub-domain. *Information Systems Frontiers*, *18*(1), 69–84. https://doi.org/10.1007/s10796-015-9575-0

Saha, A. K. (2012). A developer's first look at android. *Linux For You*, *2008*(January), 48–50.

Serrano, N., & Ciordia, I. (2004). Ant: automating the process of building applications. *IEEE Software*, *21*(6), 89–91. https://doi.org/10.1109/MS.2004.33

Sommerville, I. (2005). *Ingeniría de Software* (Séptima ed). Pearson Educación.

The Open Group. (2008). Environment Variables. Retrieved April 8, 2017, from http://pubs.opengroup.org/onlinepubs/000095399/basedefs/xbd_chap08.html#tag_08_03

Vicente-Chicote, C., Toledo, A., Fernández, C., & Sánchez, P. (2006). Generación automática de aplicaciones mixtas Sw/Hw mediante la integración de componentes COTS. In *IEEE Latin America Transactions* (Vol. 4, pp. 93–99). https://doi.org/10.1109/TLA.2006.1642456

Vignaga, A., & Perovich, D. (2003). Enfoque Metodológico para el Desarrollo Basado en Componentes. Retrieved from http://www.fing.edu.uy/inco/grupos/coal/uploads/Investigaci%F3n/vp03.pdf

Villate, J. (2002). Glosario de Informática Inglés-Español.

XSL Working Group, & XML Linking Working Group. (2016). XML Path Language (XPath). Retrieved February 24, 2017, from https://www.w3.org/TR/xpath/

7. Anexos

7.1. ANEXO A

Lista Actor Objetivo y casos de uso de la aplicación Generador App

VISION Y ANALISIS DEL NEGOCIO

Lista Actor Objetivo

Actor	:	Objetivo
Administrador	:	Seleccionar App Configurar parámetros App Generar App Obtener App Configurar Proyecto
Bombero	:	Seleccionar App Obtener App
Sistema de Construcción	:	Generar App
Sistema de Instalación de App	:	Obtener App

CASOS DE USO

Antes de presentar los casos de uso se muestra la plantilla basada en el ejemplo de casos de uso del libro Uml y Patrones de Craig Larman. Ésta plantilla es utilizada para crear los casos de uso de este proyecto.

PLANTILLA CASO DE USO BASADO EN EL LIBRO UML Y PATRONES DE CRAIG LARMAN

Caso de uso <UC1>	:	<Nombre Caso de Uso>
Actor Principal	:	<Actor1>
Personal involucrado e intereses:		

Personal Involucrado		Interés
<Actor1>	:	<El actor principal que recurre a los servicios del sistema para cumplir un objetivo>
<Actor2>	:	<Interés que tiene el actor en esta característica o funcionalidad del sistema>
<Actor3>	:	
<Actor4>	:	

Precondiciones	:	<Lo que se debe cumplir antes de comenzar un escenario del caso de uso, se asume que es verdad. Normalmente implica un escenario de otro caso de uso que ha terminado con éxito. >
Garantías de éxito (Postcondiciones)	:	<Establece que debe cumplirse cuando el caso de uso se completa con éxito, debe satisfacer las necesidades de todo el personal involucrado. >
Escenario Principal o de éxito (o flujo Básico) :		

1. <Descripción escena, puede ser una interacción entre actores y/o actor y sistema, una validación (normalmente a cargo del sistema) o un cambio de estado realizado por el sistema (por ejemplo, el sistema registra o modifica algo). >

2. <Descripción escena, ponga el nombre de los actores en mayúsculas>
3. <Descripción escena, ejemplo: Actor1 introduce la identificación de un artículo>

El actor involucrado repite pasos <2-3> hasta <que se indique>

4. <Descripción escena>
5. <Descripción escena>

Extensiones (o Flujos Alternativos) :

*a. <Condición de extensión >:

<Describe una condición de Extensión que puede ser posible durante cualquiera (o al menos la mayoría) de los pasos. >

1. <Utilice las etiquetas *a,*b, etc…>
2. <Describe la condición>

*b. <Otra Condición de extensión >:

<Describe una condición de Extensión que puede ser posible durante cualquiera (o al menos la mayoría) de los pasos. >

1. <Utilice las etiquetas *a,*b, etc…>
2. <Describa la condición>

2a. <Escenario alternativo o bifurcación de la parte 2. del Escenario Principal, puede ser una bifurcación de éxito o de fracaso.>

1. <Respuesta de la condición de extensión>
2. <Respuesta de la condición de extensión>

2b. <Otro escenario alternativo o bifurcación o extensión de la parte 2. del Escenario Principal, puede ser una bifurcación de éxito o de fracaso.>

1. <Respuesta de la condición de extensión>

2. <Respuesta de la condición de extensión>

3a. <Ejemplo: La identificación del articulo no es valida.>

1. <Es sistema señala el error y rechaza la entrada>

Requisitos especiales:

- <Si un requisito no funcional, atributo de calidad o restricción se relaciona de manera específica con un caso de uso se recoge en el caso de uso.>
- <Esto incluye calidades tales como rendimiento, fiabilidad y facilidad de uso, y restricciones de diseño (a menudo, en dispositivos de entrada/salida) que son obligados o se consideran probables.>
- <Se recomienda escribir los requisitos no funcionales en una especificación complementaria para favorecer la gestión del contenido, etc…, pero queda a su criterio.>

Lista de tecnología y variaciones de datos:

2a. <Restricción técnica impuesta por el personal involucrado con respecto a tecnologías.>

2b. <Pueden ser restricciones de diseño anticipada, así que intente evitarlas.>

2c. <No debería tener múltiples pasos como las extensiones.>

3a. <Ejemplo (tecnología): La identificación de articulo debe leerse con un código de barra>

3b. <Ejemplo (datos): El identificador podría ser cualquier esquema de código UPC, EAN, JAN o SKU>

Frecuencia	:	Frecuencia de la característica (casi continuo, etc…).

Temas abiertos:		
• <> • <> • <>		

Casos de uso del proceso del negocio elementales (EBP)

Caso de uso UC01	:	Seleccionar App
Actor Principal	:	Administrador
Personal involucrado e intereses:		
Personal Involucrado		Interés
Administrador	:	Desea que la selección sea precisa y clara para evitar el error humano al seleccionar la aplicación.
Usuario	:	Desea que la selección de los componentes sea de acuerdo a su necesidad particular.
Afectado	:	Que la aplicación facilite el control de la emergencia y la rápida llegada del personal al foco del incendio.
Precondiciones	:	Los componentes que generar la App deben estar disponibles y el Generador de App debe estar configurado para los componentes que generan estas aplicaciones.

Garantías de éxito (Postcondiciones)	:	La App seleccionada debe quedar guardada de manera temporal.
Escenario Principal o de éxito (o flujo Básico) :		
1. El sistema despliega las App que son posible generar. 2. El Administrador selecciona la App que el Bombero desea generar. 3. El Administrador finaliza la selección. 4. El sistema Guarda de manera temporal la App seleccionada.		
Extensiones (o Flujos Alternativos) :		
*a. El Sistema falla en cualquier momento El Administrador debe reiniciar la aplicación. 1. El Administrador finaliza la aplicación Generador App. 2. El Administrador inicia la aplicación Generador App. 3. El Administrador vuelve a ingresar la información recolectada hasta el punto del fallo.		
Requisitos especiales:		
Lista de tecnología y variaciones de datos:		
Frecuencia	:	Una vez por cada App que se desea generar.
Temas abiertos:		

Caso de uso UC02	:	Configurar parámetros App
Actor Principal	:	Administrador

Personal involucrado e intereses:

Personal Involucrado		Interés
Administrador	:	La información de los parámetros de la aplicación debe presentarse en forma clara y sin ambigüedades; le interesa poder configurar el HOST y PORT para que la App se pueda conectar a la base de datos externa, la versión del dispositivo móvil que se utilizará y si la App generada será de Release o Debug.
Usuario	:	Le interesa que la App generada pueda utilizarse en su dispositivo móvil.

Precondiciones	:	La aplicación a generar debe estar seleccionada.
Garantías de éxito (Postcondiciones)	:	La configuración debe quedar guarda temporalmente y estar disponible hasta que se generé la aplicación.

Escenario Principal o de éxito (o flujo Básico) :

1. El sistema despliega los parámetros de la App a configurar
2. El Administrador ingresa el HOST utilizado para que la App se conecte a la base de datos externa.
3. El Administrador ingresa el PORT utilizado para que la App se conecte a la base de datos externa.

4. El Administrador elige la versión del dispositivo para el que se genera la App que puede ser “superior o igual a Android 14” o “inferior a Android 14”.
5. El Administrador selecciona el modo de construcción que puede ser debug o release.
6. El Administrador indica mediante una acción que ha finalizado la configuración de la App.
7. El sistema recolecta los parámetros de la App.

Extensiones (o Flujos Alternativos) :

*a. El Sistema falla en cualquier momento

El Administrador debe reiniciar la aplicación.

1. El Administrador finaliza la aplicación Generador App.
2. El Administrador inicia la aplicación Generador App.
3. El Administrador vuelve a ingresar la información recolectada hasta el punto del fallo.

2-3a. El Administrador selecciona el servidor por defecto

1. El sistema rescata el servidor por defecto.
2. El sistema bloquea el ingreso de cualquier otro servidor.

2b. El HOST ingresado no representa un valor valido

1. El sistema despliega un mensaje indicando que los valores validos comprenden los valores desde 1.0.0.0 hasta 223.255.255.0.
2. El Administrador ingresa nuevamente el HOST de la base de datos externa.

El Administrador repite pasos 1-2 hasta que ingrese un HOST valido.

3 b. El PORT ingresado no representa un valor valido

1. El sistema despliega un mensaje indicando que los valores validos comprenden los solo números enteros superiores a 1024 o el puerto 80. 2. El Administrador ingresa nuevamente el PORT de la base de datos externa. El Administrador repite pasos 1-2 hasta que ingrese un PORT válido.		
Requisitos especiales:		
Lista de tecnología y variaciones de datos:		
2a. El HOST se encuentra definido en los componentes que generan la App como una variable de tipo String. 2b. El HOST corresponde a la dirección IP del servidor y corresponde a una IPv4. 3a. El PORT se encuentra definido en los componentes que generan la App como una variable tipo int. 3b. El PORT corresponde al puerto que se utilizará para conectar la App con el servidor externo.		
Frecuencia	:	Una vez por cada App que se desea generar.
Temas abiertos:		

Caso de uso UC03	:	Generar App
Actor Principal	:	Administrador
Personal involucrado e intereses:		

Personal Involucrado		Interés
Administrador	:	Le interesa obtener la App correcta, esto es, que los parámetros seleccionados y la llave utilizada sea la correcta, además de la información sobre la App que se generará e indicaciones de que el proceso de construcción ha finalizado.
Usuario	:	Le interesa que el tiempo de generación de la aplicación móvil sea reducido y que la App que se genere sea la que solicitó.
Sistema de Construcción	:	Sistema encargado de construir la aplicación móvil; le interesa generar la aplicación según los parámetros enviados por el Generador App y dar respuesta del resultado de la construcción.

Precondiciones	:	Los componentes de la aplicación a generar deben estar seleccionados y disponibles, los parámetros de la App deben estar completamente configurados y el generador también.
Garantías de éxito (Postcondiciones)	:	Obtener la aplicación generada y el archivo con que contiene el resultado de la construcción.

Escenario Principal o de éxito (o flujo Básico) :
1. El Administrador indica al sistema que se desea generar la App. 2. El sistema muestra la información relevante de la aplicación a generar, esta información puede incluir la App seleccionada, y los parámetros con los que se generará la aplicación móvil. 3. El sistema excluye los componentes (UC6) que no fueron seleccionados para generar la App. 4. El sistema añade los parámetros de la aplicación Móvil a la construcción. 4.1.El sistema añade el HOST. 4.2.El sistema añade el PORT. 4.3.El sistema añade la versión de Android. 5. El sistema llama al Sistema de Construcción para construir la App enviándole el modo de construcción. 6. El sistema espera la confirmación del Sistema de Construcción que la App se terminó de construir. 7. El sistema recibe la confirmación de que la construcción ha finalizado con éxito. 8. El sistema de construcción envía la ruta de la aplicación generada y la ruta del resultado de la construcción al sistema. 9. El sistema muestra al Administrador información que la construcción finalizo. 10. El Administrador finaliza la generación de la aplicación. 11. El sistema incluye los componentes (UC7) que fueron excluidos para una futura generación de aplicaciones.
Extensiones (o Flujos Alternativos) :
*a. El Sistema falla en cualquier momento El Administrador debe reiniciar la aplicación. 1. El Administrador finaliza la aplicación Generador App.

2. El Administrador inicia la aplicación Generador App. 3. El Administrador vuelve a ingresar la información recolectada hasta el punto del fallo. 7a. El sistema recibe un mensaje desde el Sistema de Construcción indicando que la construcción de la aplicación ha fallado. 1. El sistema lanza el resultado de la construcción que indica el fallo.		
Requisitos especiales:		
Lista de tecnología y variaciones de datos:		
5a. El sistema de construcción debe permitir él envió de información relativa a la construcción a la aplicación.		
Frecuencia	:	Una vez por cada App que se desea generar.
Temas abiertos:		

Caso de uso UC4	:	Obtener App
Actor Principal	:	Administrador

Personal involucrado e intereses:

Personal Involucrado		Interés
Administrador	:	Le interesa obtener de manera sencilla la App construida.
Usuario	:	Le interesa poder obtener la App e instalarla en el dispositivo.

Precondiciones	:	La App debe haber sido generada previamente.
Garantías de éxito (Postcondiciones)	:	Obtener la aplicación generada.

Escenario Principal o de éxito (o flujo Básico) :

1. El sistema despliega las opciones de postConstrucción, las cuales son “Mostrar APK en carpeta”, “Instalar en Dispositivo Conectado”, “Resultado de la Construcción”.
   1.1.El sistema despliega la opción Mostrar APK en carpeta.
   1.2.El sistema despliega la opción Instalar en Dispositivo Conectado.
   1.3.El sistema despliega la opción Resultado de la Construcción.
2. El sistema carga la ruta de la App.
3. El sistema carga la ruta del archivo que contiene el resultado de la construcción.
4. El Administrador selecciona la opción Mostrar APK en carpeta.

4.1.El Administrador entrega la aplicación móvil al bombero que solicito la App. 5. El Administrador selecciona la opción Resultado de la Construcción. 5.1.El Sistema muestra el resultado de la construcción en pantalla. 6. El Administrador finaliza la aplicación.
Extensiones (o Flujos Alternativos) :
*a. El Sistema falla en cualquier momento El Administrador debe reiniciar la aplicación. 1. El Administrador finaliza la aplicación Generador App. 2. El Administrador inicia la aplicación Generador App. 3. El Administrador vuelve a ingresar la información recolectada hasta el punto del fallo. 4a. El Administrador selecciona la opción Instalar en Dispositivo Conectado 1. El Sistema llama al Sistema de Instalación para realizar la instalación de la App. 2. El Sistema de Instalación instala la App en el dispositivo del Interesado.
Requisitos especiales:
Lista de tecnología y variaciones de datos:

Frecuencia	:	Una vez por cada App que se desea generar.
Temas abiertos:		

Caso de uso UC5	:	Configurar Proyecto
Actor Principal	:	Administrador
Personal involucrado e intereses:		

Personal Involucrado		Interés
Administrador	:	Realizar la configuración inicial del proyecto Android que contiene los componentes que generar las App, esto implica exportar el proyecto junto con las bibliotecas pertenecientes al proyecto, ya sean internas o externas en el caso de necesitarlas, la llave de debug y llave de reléase junto con los parámetros pertenecientes a la llave, y la ubicación del Android SDK.

Precondiciones	:	El proyecto utilizado para la creación de las App debe estar disponible.
Garantías de éxito (Postcondiciones)	:	El proyecto Android Exportado al sistema Generador App.
Escenario Principal o de éxito (o flujo Básico) :		

1. El sistema despliega las opciones de configuración inicial.
2. El Administrador selecciona la ruta donde se encuentra la carpeta principal del proyecto “rutaProyecto” que contiene los componentes que generar las App
3. El Administrador selecciona la ruta donde se encuentra la llave de debug “rutaLlaveDebug”.
4. El Administrador selecciona los datos de la llave de reléase.

4.1.El Administrador selecciona la ruta de la llave de reléase.

4.2.El Administrador ingresa el Alias de la llave de reléase.

4.3.El Administrador ingresa el Password del Alias “AliasPass” de la llave de reléase.

4.4.El Administrador ingresa el Store de la llave de reléase.

4.5.El Administrador ingresa el Password del Store “StorePass” de la llave de reléase.

5. El Administrador ingresa la ruta de una biblioteca interna del proyecto.

El Administrador repite el paso 5 hasta que ingresen todas las bibliotecas internas del proyecto.

6. El Administrador ingresa la ruta de una biblioteca externa del proyecto.

El Administrador repite el paso 6 hasta que ingresen todas las bibliotecas externas del proyecto.

7. El Administrador ingresa la ruta del Android SDK.
8. El Administrador indica al sistema que ha terminado de ingresar los datos de configuración del proyecto.
9. El Sistema exporta los archivos del proyecto al directorio “proyecto”.
10. El Sistema exporta las Bibliotecas externas e internas.

Extensiones (o Flujos Alternativos) :

*a. El Sistema falla en cualquier momento

El Administrador debe reiniciar la aplicación.

1. El Administrador finaliza la aplicación Generador App.

2. El Administrador inicia la aplicación Generador App.
3. El Administrador vuelve a ingresar la información recolectada hasta el punto del fallo.

4a. El Administrador no selecciona todos los datos correspondiente a la llave de reléase.

1. El sistema indica al Administrador que no se han ingresado todos los datos correspondiente a la llave de reléase.
2. El Administrador ingresa el o los datos que no han sido seleccionados.

El Administrador repite los pasos 1-2 hasta que todos los datos correspondiente a la llave de reléase sean ingresados.

8a. El Administrador no ingreso la ruta del proyecto.

1. El sistema señala mediante un mensaje que la ruta del proyecto es de carácter obligatorio.
2. El Administrador ingresa la ruta del proyecto.

8b. El Administrador no ingreso la ruta del Android SDK.

1. El sistema señala mediante un mensaje que la ruta del Android SDK es de carácter obligatorio.
2. El Administrador ingresa la ruta del Android SDK.

Requisitos especiales:		
Lista de tecnología y variaciones de datos:		
Frecuencia	:	Frecuencia de la característica (casi continuo, etc…).
Temas abiertos:		

Casos de uso de Subfunción

Caso de uso UC6	:	Excluir componentes
Actor Principal	:	Sistema
Personal involucrado e intereses:		
Personal Involucrado		Interés
Administrador	:	Le interesa que el sistema pueda excluir el componente y todos los archivos asociados a él para generar una App.
Precondiciones	:	Los componentes excluidos deben estar disponibles e incluidos en el proyecto.
Garantías de éxito (Postcondiciones)	:	Los componentes excluidos del proyecto.
Escenario Principal o de éxito (o flujo Básico) :		

1. El Administrador solicita al sistema la exclusión de componente.
2. El sistema carga el estado del componente.
3. El sistema rescata la lista de archivos asociados al componente con el identificador del componte.
4. El sistema modifica los archivos asociados que indiquen la acción de modificar
5. El sistema excluye de la construcción los archivos asociados que indiquen la acción de mover.
6. El sistema cambia el estado del componente a EXCLUIDO.

Extensiones (o Flujos Alternativos) :		
*a. El Sistema falla en cualquier momento El Administrador debe reiniciar la aplicación. 1. El Administrador finaliza la aplicación Generador App. 2. El Administrador inicia la aplicación Generador App. 3. El Administrador vuelve a ingresar la información recolectada hasta el punto del fallo. 2a. El estado del componente es EXCLUIDO 1. El sistema finaliza la exclusión del componente.		
Requisitos especiales:		
Lista de tecnología y variaciones de datos:		
3a. El archivo asociado debe contener como atributo la ruta del archivo		
Frecuencia	:	1 o más veces por cada App que se desea generar.
Temas abiertos:		

Caso de uso UC7	:	Incluir componentes
Actor Principal	:	Sistema
Personal involucrado e intereses:		

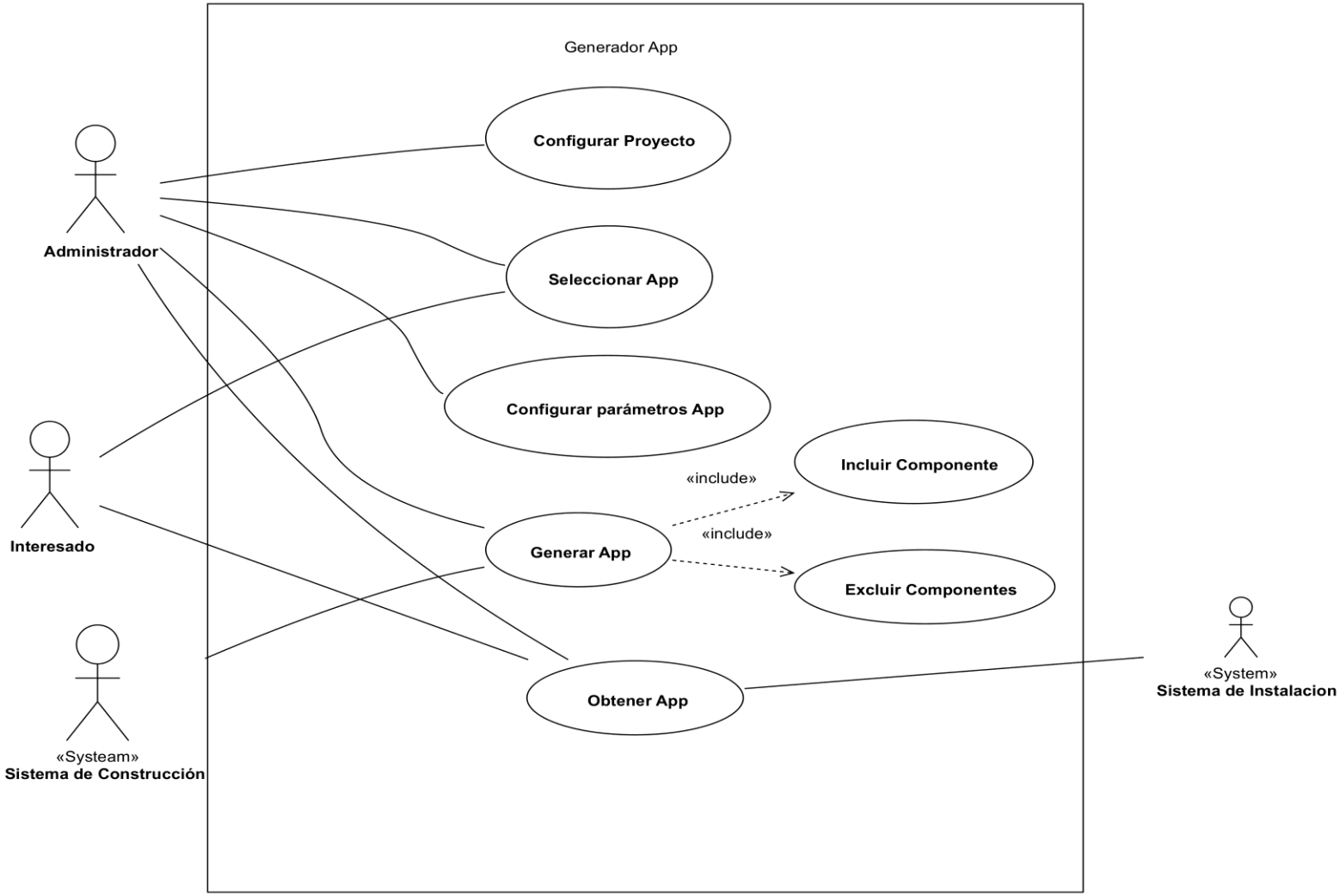
Personal Involucrado		Interés
Administrador	:	Le interesa que el sistema pueda incluir el componente y todos los archivos asociados a él para generar una App.

Precondiciones	:	Los componentes incluidos deben estar disponibles e no encontrarse previamente en el directorio del proyecto.
Garantías de éxito (Postcondiciones)	:	Los componentes excluidos del proyecto.
Escenario Principal o de éxito (o flujo Básico) :		

1. El Administrador solicita al sistema la inclusión de componente.
2. El sistema carga el estado del componente.
3. El sistema rescata la lista de archivos asociados al componente con el identificador del componte.
4. El sistema modifica los archivos asociados que indiquen la acción de modificar
5. El sistema incluye en la construcción los archivos asociados que indiquen la acción de mover.
6. El sistema cambia el estado del componente a INCLUIDO.

Extensiones (o Flujos Alternativos) :		
*a. El Sistema falla en cualquier momento El Administrador debe reiniciar la aplicación. 1. El Administrador finaliza la aplicación Generador App. 2. El Administrador inicia la aplicación Generador App. 3. El Administrador vuelve a ingresar la información recolectada hasta el punto del fallo. 2a. El estado del componente es INCLUIDO. 1. El sistema finaliza la inclusión del componente.		
Requisitos especiales:		
Lista de tecnología y variaciones de datos:		
Frecuencia	:	1 o más veces por cada App que se desea generar.
Temas abiertos:		

Diagrama de Casos de Uso de la aplicación Generado



7.2. ANEXO B

ESPECIFICACIÓN COMPLEMENTARIA.

Historia de revisiones

Versión	Fecha	Descripción	Autor
Producto v1		Primer Producto, permite la construcción de aplicaciones Android y la separación de un componente genérico.	Braulio Alberto Quiero Hernández
Producto v2	10-01-2017	Segundo Producto, este posee la GUI finalizada y permite la creación de 3 Aplicaciones	Braulio Alberto Quiero Hernández

1. Introducción

El sistema Generador App pretende satisfacer la necesidad de generar aplicaciones basadas en componentes para la Administración de emergencia, tiene como objetivo secundario ser la base para realizar un generador de aplicaciones móviles para un dominio arbitrario.

2. Funcionalidad

Dentro de las funcionalidades requeridas se solicita tener funciones utilitarias para el manejo de archivos XML, esto debido a que el desarrollo de aplicaciones Android implica el manejo de este tipo de archivos como funciones de manejo de archivos en general, esto debido a que existe una alta posibilidad de tener que leer y modificar archivos relacionados con la inclusión o exclusión de componentes.

3. Facilidad de uso

El sistema será operado por personas que participen en la administración de la emergencia por lo que no se presupone que entiendan el dominio de construcción de aplicaciones por lo cual el sistema deberá poseer un manual de usuario que explique de manera sencilla las funcionalidades del sistema.

4. Fiabilidad

El sistema debe poder recuperarse si existe un fallo en la construcción y en lo posible guardar los parámetros en los que se encontraba la aplicación antes del fallo.

5. Rendimiento

El sistema necesita como mínimo 5gb de espacio en disco para los sistemas de construcción de App y el Generador App.

6. Soporte

6.1. Adaptabilidad

El sistema debe contar con la capacidad de adaptarse a nuevos componentes, de manera opcional debe permitir la construcción de aplicaciones en otro dominio con una mínima modificación en el código fuente del generador.

7. Restricciones de Implementación

Los componentes de la aplicación están hechos en Java y la implementación de varias de sus interfaces son dependientes de Android, por lo que el sistema debe soportar la construcción de aplicaciones Java para Android.

Las aplicaciones generadas deben funcionar en dispositivos móviles más utilizados, hasta este momento son Lollipop (5.0) y KitKat (4.4), que se ajustan a la versión utilizada para construir los componentes.

https://developer.android.com/about/dashboards/index.html#Screens

Se recomienda utilizar java para la construcción de la aplicación por su portabilidad.

El sistema debe funcionar bajo sistema operativo Windows 7 o superior y de manera opcional para Ubuntu 12.04 o superior.

8. Componentes adquiridos

No se ha adquirido software para la construcción de la aplicación, aunque se recomienda:

Aplicación	Descripción
Microsoft Visio	Software comercial, constantemente se actualiza y tiene una interfaz fácil de manejar, solo se recomienda su uso en caso de necesitar alguna de sus funciones que no se encuentren dentro de los software de libre distribución. Sitio web: https://www.microsoftstore.com/store/mscl/es_CL/pdp/productID.5059135500

9. Componentes de libre distribución

No hay restricciones con los componentes de libre distribución, aunque se recomienda para la construcción del sistema un entorno de desarrollo de libre distribución compatible con java. Se recomiendan los siguientes entornos de libre distribución:

Aplicación	Descripción
Neatbeans IDE	Software gratuito, especializado para la creación de aplicaciones en Java, aunque también soporta lenguajes como JavaScript, HTML5, PHP, C/C++ entre otros. La versión más reciente de JDK se puede descargar con este IDE directamente desde su página oficial. Sitio web: https://netbeans.org/
Eclipse	Software open source muy popular, con la capacidad de instalación de plugins para aumentar sus funcionalidades. Sitio web: https://www.eclipse.org/

Para el desarrollo de los diagramas se recomienda.

Aplicación	Descripción
ArgoUML	Es OpenSource, soporta UML 1.4, y permite la ingeniería inversa. Sitio web: http://argouml.tigris.org/
Dia	Herramienta para dibujar diagramas estructurados, soporta UML, diagramas de flujo, entre otros y se encuentra disponible para Windows, Mac OS X y Linux. Sitio web: http://dia-installer.de/
StarUML	Posee una versión OpenSource y una versión comercial. La versión comercial permite la prueba sin tiempo límite, soporta Java, C# y C++, mediante la instalación de plugins permite ingeniería inversa sobre Java. Sitio web http://staruml.sourceforge.net/v1/download.php (versión libre) http://staruml.io/ (versión pago, permite prueba sin tiempo límite)

10. Interfaces

10.1.Interfaces y hardware destacables

No hay interfaces de hardware destacables, aunque se recomienda que el software permite como entrada un mouse y teclado.

10.2.Interfaces software

El sistema necesita poder conectarse con los sistemas externos que prestan servicio para la construcción de la aplicación y de manera opcional con el sistema que permite la instalación de la aplicación en dispositivos.

11. Cuestiones Legales

De los programas recomendados se pide centrar la atención en utilizar software de libre distribución para permitir el uso del software en computadores de equipos de emergencia, que en caso de bomberos, son voluntarios en su labor.

7.3. ANEXO C

7.3.1. Glosario

TERMINO	DEFINICIÓN E INFORMACIÓN	ALIAS
ADT	Siglas de Android Developer Tools, es un plugin para Eclipse que provee una interfaz gráfica para acceder a muchas de las herramientas del SDK de Android de la línea de comandos.	
Android	Sistema Operativo diseñado por Google principalmente para dispositivos móviles, pero que ha extendido su uso a dispositivos inteligentes para el hogar.	
Android SDK	Kit de desarrollo de aplicaciones Android.	
Ant	Siglas de "Another Build Tools", es una herramienta para la construcción de aplicaciones java mediante la línea de comandos del sistema operativo.	-Apache Ant
App	Aplicación Móvil que genera el sistema Generador App.	-Aplicación Móvil -Apk
Base de datos externa	Representa la base de datos externa, empleada para almacenar los datos compartidos entre las Aplicaciones móviles.	
componente	Indica un conjunto de atributos (ID, nombre, archivos asociados), que definen un componente de software para este sistema.	

Generador App	Es el nombre del sistema en construcción. También se le puedo nombrar simplemente como “Sistema”.	-Sistema -Aplicación
GUI	Siglas de graphical user interface, se hace referencia a la interfaz gráfica de usuario.	
Llave	Se refiere a un archivo que permite la creación de Aplicaciones Móviles Android, si la Llave es de Debug, permite crear una App en modo Debug y si la Llave es de Release permite crear una App en modo Release.	-Clave -Key
OpenSource	El termino se refiere no solo a programas de código abierto, sino a la libre redistribución y permitir realizar trabajos derivados con este software. Para mayor información revise el sitio web https://opensource.org/definition	
Plugin	Complemento que puede ser agregado a un programa para añadir funcionalidades adicionales.	
Resultado de la construcción	Archivo de texto que contiene información con respecto a la App que se quiere generar, esta información contiene advertencias, indica si la construcción fue generada de manera exitosa o no además de indicar cuál fue la causa del error cuando la aplicación no es generada.	
SDK	Siglas de Software development kit, se utiliza para referirse a un conjunto de herramientas que permite el desarrollo de un software.	
Servidor	Representa un par de valores HOST y PORT.	

XML	Siglas de eXtensible Markup Language o Lenguaje de Marcado extensible, es un lenguaje de texto simple y muy flexible, se caracteriza por el uso de etiquetas (par nombre, valor), tiene múltiples usos, desde formatear documentos, crear archivos de configuración, hasta servir de base para otros lenguajes.	

7.3.2. Diccionario de Datos.

<table>
<tr><th>TÉRMINO</th><th colspan="4">DEFINICIÓN E INFORMACIÓN</th></tr>
<tr><td>HOST</td><td colspan="4">Descripción:

Representa la dirección IP del tipo IPv4 de la base de datos externa a la que se conecta la Aplicación Móvil.

Alias :
Formato : String
Rango de valores : Desde 1.0.0.0 hasta 223.255.255.0.
Reglas de validación :</td></tr>
<tr><td>PORT</td><td colspan="4">Descripción:

Representa el puerto con el que la base de datos externa a la que se conecta a la Aplicación Móvil.

Alias : Sistema, Aplicación
Formato : int</td></tr>
</table>

	Rango de valores	:	80 y valores superiores a 1024	
	Reglas de validación			
App	Descripción:			
	Representa una de las posibles App que puede generar el sistema Generador App.			
	Alias	:	Apk	
	Formato	:	Puede ser definido como una enumeración o entero.	
	Rango de valores	:	{BASE=0,BASI_ONLY=1,ORIGINAL=2}	
	Reglas de validación			
Versión Android	Descripción:			
	El atributo indica si la aplicación móvil que se generará funcionara en dispositivos Android con una versión de Android 14 o superior o para una versión de Android inferior a 14.			
	Alias	:	versionAndroid	
	Formato	:	Puede ser definido como una enumeración o entero.	
	Relaciones	:		
	Rango de valores	:	{Android 14 o Superior=0, Android Inferior a 14=1}	

	Reglas de validación			
Modo Construcción	Descripción:			
	Indica el modo en que se construirá la App.			
	Alias	:	modoConstruccion, modo de construcción	
	Formato	:	Puede ser definido como una enumeración o entero.	
	Rango de valores	:	{DEBUG=0,RELEASE=1}	
	Reglas de validación			
rutaApp	Descripción:			
	Indica la ruta de la aplicación móvil generada por el sistema			
	Alias	:	ruta aplicación	
	Formato	:	String	
	Rango de valores	:	Cadenas de caracteres exceptuando los caracteres,”:”,”*”,”?”,”<”,”>”,”|”.	
	Reglas de validación		Los caracteres “\” y “/” se utilizan para separar el nombre de archivos y o directorios, utilize “\” para el sistema operativo Windows y “/” en caso contrario.	
Estado del componente	Descripción:			
	Indica si el componente está excluido o incluido en la generación de aplicaciones móviles.			

	Alias	:	estado	
	Formato	:	String	
	Rango de valores	:	{INCLUIDO,EXCLUIDO}	
	Reglas de validación			
archivoAsociado	Descripción:			
	Indica la ruta del archivo asociado a un componente.			
	Alias	:	Componente asociado	
	Formato	:	String	
	Rango de valores	:	Cadenas de caracteres exceptuando los caracteres,”:”,”*”,”?”,”<”,”>”,”|”.	
	Reglas de validación	:	Los caracteres “\” y “/” se utilizan para separar el nombre de archivos y o directorios, utilize “\” para el sistema operativo Windows y “/” en caso contrario.	
rutaProyecto	Descripción:			
	Indica la ruta donde se encuentra la carpeta principal del proyecto.			
	Alias	:	rutaProyecto	
	Formato	:	String	
	Rango de valores	:	Cadenas de caracteres exceptuando los caracteres,”:”,”*”,”?”,”<”,”>”,”|”.	
	Reglas de validación	:	Los caracteres “\” y “/” se utilizan para separar el nombre de archivos y o directorios, utilize “\” para el sistema operativo Windows y “/” en caso contrario.	

rutaLlaveDebug	Descripción:		
	Indica la ruta de la llave de debug del proyecto.		
	Alias	:	rutaLlaveDebug
	Formato	:	String
	Rango de valores	:	Cadenas de caracteres exceptuando los caracteres,”:”,”*”,”?”,”<”,”>”,”|”.
	Reglas de validación	:	Los caracteres “\” y “/” se utilizan para separar el nombre de archivos y o directorios, utilize “\” para el sistema operativo Windows y “/” en caso contrario.
rutaLlaveRelease	Descripción:		
	Indica la ruta de la llave de Release del proyecto.		
	Alias	:	rutaLlaveRelease
	Formato	:	String
	Rango de valores	:	Cadenas de caracteres exceptuando los caracteres,”:”,”*”,”?”,”<”,”>”,”|”.
	Reglas de validación	:	Los caracteres “\” y “/” se utilizan para separar el nombre de archivos y o directorios, utilize “\” para el sistema operativo Windows y “/” en caso contrario.
StorePass	Descripción:		
	Indica la contraseña del Store, esta contraseña es solicitada para construir una aplicación en modo Release.		
	Alias	:	Store Password
	Formato	:	String

	Rango de valores	:	Cadenas de caracteres exceptuando los caracteres,”:”,”*”,”?”,”<”,”>”,”|”,”/”,”\”.	
	Reglas de validación	:	Sitio oficial de desarrollo no indica reglas de validación, para mayor información visite https://developer.android.com/studio/publish/app-signing.html	
Alias	Descripción: Indica el Alias solicitado para construir una aplicación móvil Android en modo Release.			
	Alias	:	Alias	
	Formato	:	String	
	Rango de valores	:	Cadenas de caracteres exceptuando los caracteres,”:”,”*”,”?”,”<”,”>”,”|”,”/”,”\”.	
	Reglas de validación	:	Sitio oficial de desarrollo no indica reglas de validación, para mayor información visite https://developer.android.com/studio/publish/app-signing.html	
AliasPass	Descripción: Indica la contraseña del Alias, esta se solicita para construir una aplicación móvil Android en modo Release.			
	Alias	:	Alias Password	
	Formato	:	String	
	Rango de valores	:	Cadenas de caracteres exceptuando los caracteres,”:”,”*”,”?”,”<”,”>”,”|”,”/”,”\”.	

	Reglas de validación	:	Sitio oficial de desarrollo no indica reglas de validación, para mayor información visite https://developer.android.com/studio/publish/app-signing.html	
rutaBibliotecaInterna	Descripción:			
	Variable usada para almacenar la ubicación de las bibliotecas internas del proyecto.			
	Alias	:	rutaBibliotecaInterna	
	Formato	:	Arreglo o lista de String	
	Rango de valores	:	Los elementos del arreglo deben ser cadenas de caracteres exceptuando los caracteres,”:”,”*”,”?”,”<”,”>”,”|”.	
	Reglas de validación	:	Los caracteres “\” y “/” se utilizan para separar el nombre de archivos y o directorios, utilize “\” para el sistema operativo Windows y “/” en caso contrario.	
rutaBiblitecaExterna	Descripción:			
	Indica la ruta de la llave de Release del proyecto.			
	Alias	:	rutaBiblitecaExterna	
	Formato	:	Arreglo o lista de String	
	Rango de valores	:	Los elementos del arreglo deben ser cadenas de caracteres exceptuando los caracteres,”:”,”*”,”?”,”<”,”>”,”|”.	
	Reglas de validación	:	Los caracteres “\” y “/” se utilizan para separar el nombre de archivos y o directorios, utilize “\” para el sistema operativo Windows y “/” en caso contrario.	

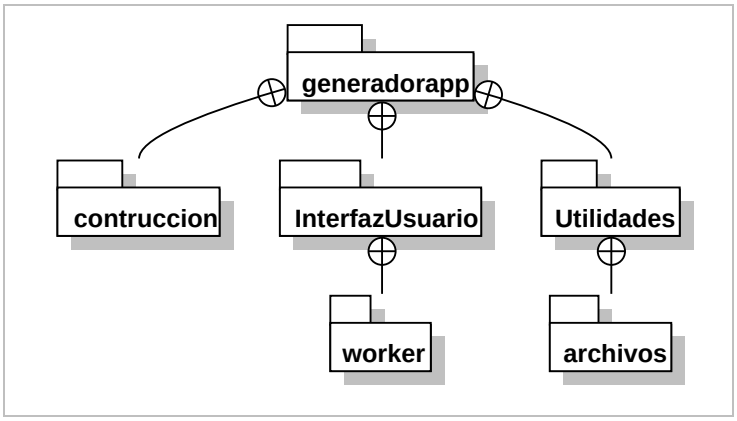
7.4. ANEXO D

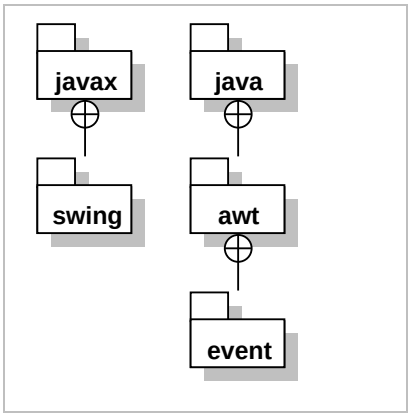
Diagramas de Clases

Los diagramas de clases se organizaron según el paquete donde se encuentra ubicado, las clases necesarias en las relaciones que no pertenecen al paquete se encuentran solo con su nombre en el clasificador, para ver sus atributos diríjase al paquete correspondiente.

7.4.1. Estructura de paquetes

Se mostrará la estructura de paquetes de la aplicación y los paquetes externos utilizados para la interfaz gráfica.



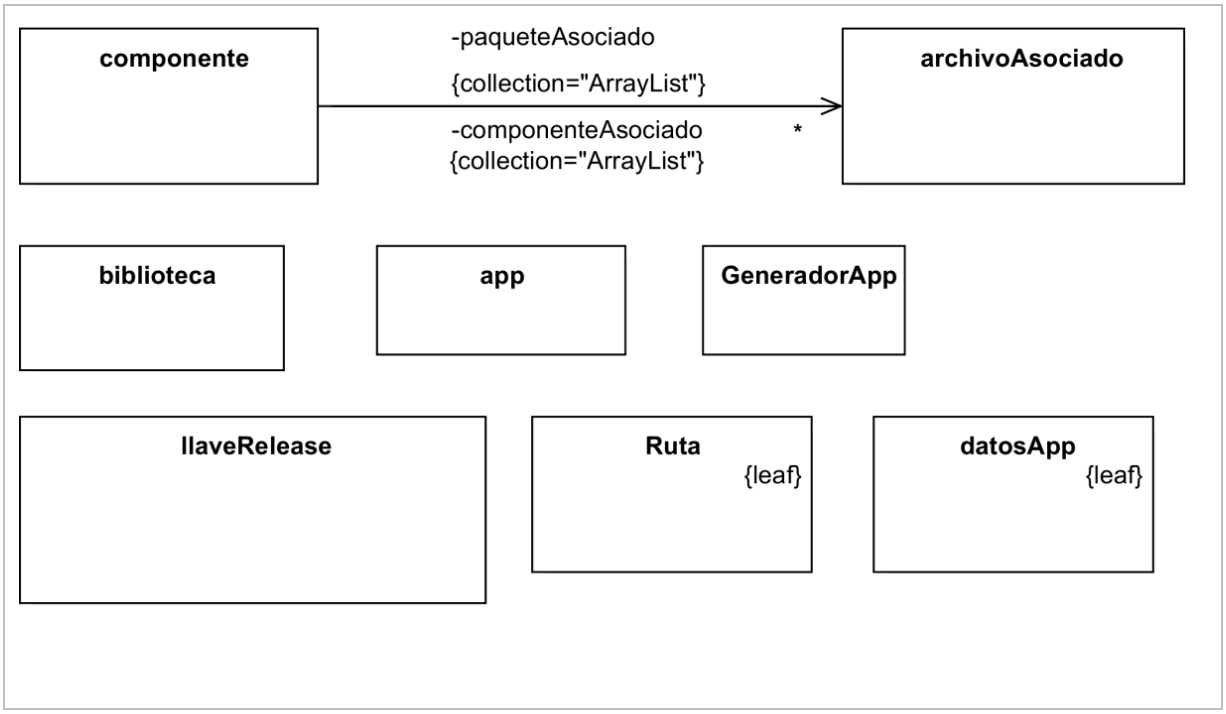


7.4.2. Diagramas de Clases

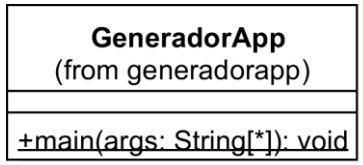
7.4.2.1. Paquete *generadorapp*

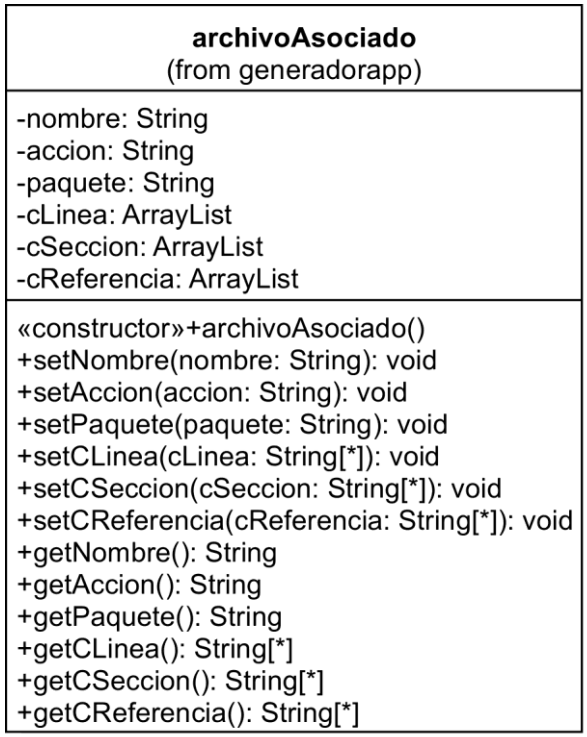
El diagrama de clases del paquete *generadorapp*. Solo se mostraran las clases en el diagrama sin atributos y procedimientos, esto para una mayor comprensión del diagrama. En las figuras siguientes se muestra el detalle de cada clase.

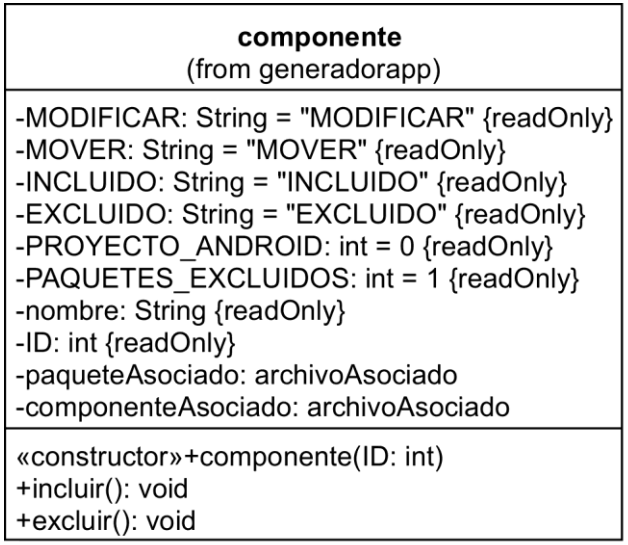
Diagrama de clases

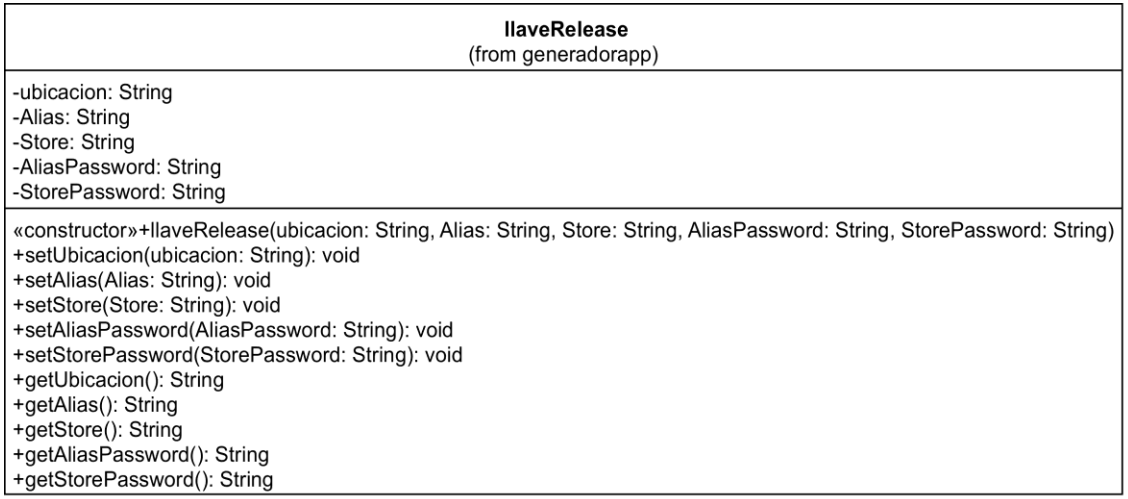


Detalle de clases









datosApp
(from generadorapp)

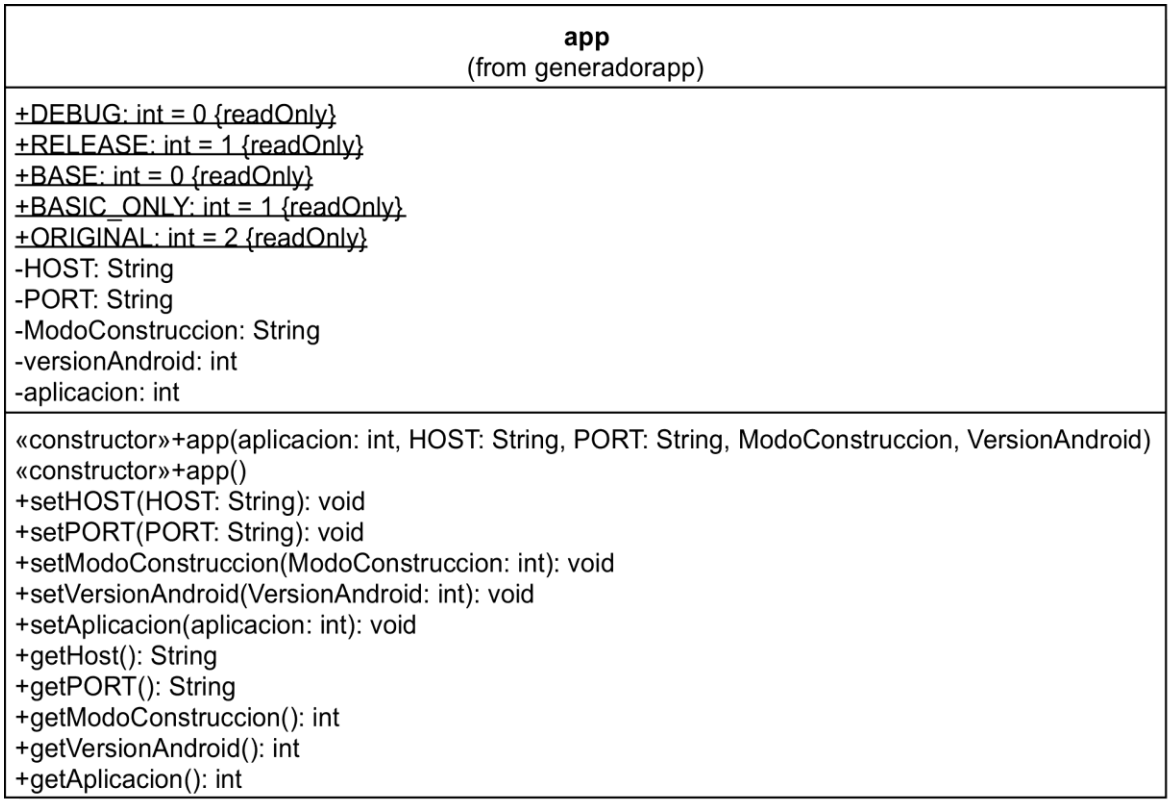
{leaf}

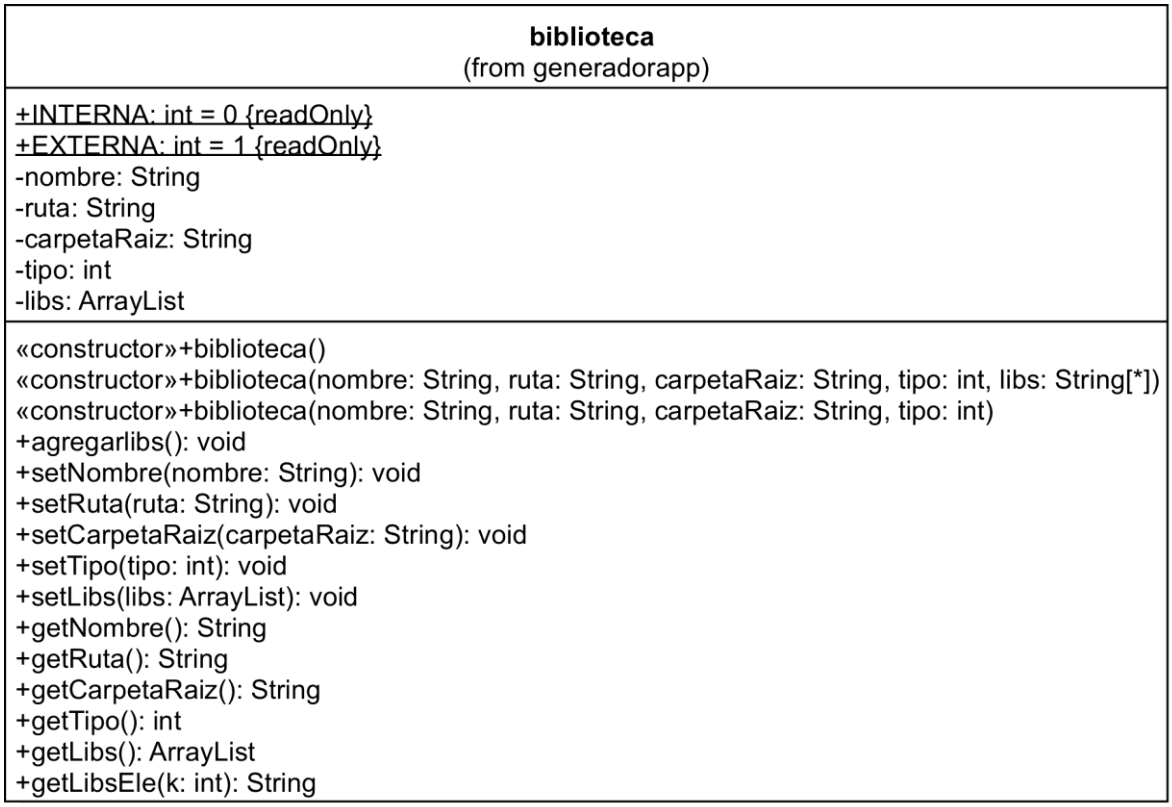
+BUILD: String = "build.xml" {readOnly}
+MANIFEST: String = "AndroidManifest.xml" {readOnly}
+ANT_PROPERTIES: String = "ant.properties" {readOnly}
+LOCAL_PROPERTIES: String = "local.properties" {readOnly}
+PROJECT_PROPERTIES: String = "project.properties" {readOnly}
+PROYECTO_PROPERTIES: String = "proyecto.properties" {readOnly}
+SRC: String = "src" {readOnly}
+BIN: String = "bin" {readOnly}
+PROYECT_NAME: String = "proyecto.raiz" {readOnly}
+LIBRARY_NAME: String = "android.library.reference." {readOnly}
+TARGET_NAME: String = "target" {readOnly}
+TARGET_VALUE: String = "android-19" {readOnly}
+PRIVATE_LIBRARY: String = "Android Private Libraries.libraryclasspath" {readOnly}
+REFERENCE_LIBRARY: String = "Referenced Libraries.libraryclasspath" {readOnly}
+ANDROID_LIBRARY: String = "Android 4.4.2.libraryclasspath" {readOnly}
+PRIVATE_LIBRARY_PATH: String {readOnly}
+REFERENCE_LIBRARY_PATH: String {readOnly}
+ANDROID_LIBRARY_PATH: String {readOnly}
+APP_CLASSPATH: String = "project/path[normalize-space(@id)='app.classpath']" {readOnly}
+APP_CLASSPATH_LOCATION: String = "bin/classes" {readOnly}
+REFERENCE_LIBS_LOCATION: String = "../../libs/" {readOnly}
+ANDROID_LIBRARY_LOCATION: String = "\\platforms\\android-19" {readOnly}
+PLAY_SERVICES_VERSION_NAME: String = "com.google.android.gms.version" {readOnly}
+PLAY_SERVICES_VERSION_VALUE: String = "@integer/google_play_services_version" {readOnly}
+PLAY_SERVICES_VERSION_PATH_NAME: String {readOnly}
+PLAY_SERVICES_VERSION_PATH_VALUE: String {readOnly}
+API_KEY_NAME: String = "com.google.android.maps.v2.API_KEY" {readOnly}
+API_KEY_DEBUG_VALUE: String = "AIzaSyADkHALHvR1fHsn3SWaJv6-jlw5tVusdW0" {readOnly}
+API_KEY_RELEASE_VALUE: String = "AIzaSyCkR9lOtq5HirLVdYHWBs1WgjGpvbvLBog" {readOnly}
+API_KEY_NAME_PATH: String {readOnly}
+API_KEY_DEBUG_PATH: String {readOnly}
+API_KEY_RELEASE_PATH: String {readOnly}
+UTF_8: String = "utf-8" {readOnly}
+CP1252: Charset {readOnly}
+DEFAULT_HOST: String = "192.168.0.32" {readOnly}
+DEFAULT_PORT: String = "80" {readOnly}
+SERVER_FILE: String = "\\src\\cl\\ucsc\\projectoftitle\\newmobilemap\\components\\edbconnections\\ExternalDBData.java" {readOnly}
+HOST_VAR: String = "SERVER_HOST" {readOnly}
+PORT_VAR: String = "SERVER_PORT" {readOnly}
+LINEA_HOST: String {readOnly}
+LINEA_PORT: String {readOnly}
+COMENTARIO_LINEA: String = "//" {readOnly}
+COMENTARIO_SECCION_INICIO: String = "/*" {readOnly}
+COMENTARIO_SECCION_FIN: String = "*/" {readOnly}

Ruta
(from generadorapp)

{leaf}

+RUTA_PAQUETES_EXCLUIDOS: String = "proyecto\\excluir" {readOnly}
+RUTA_PROYECTO_ANDROID: String = "proyecto\\aplicacionBase" {readOnly}
+RUTA_SCRIPT: String = "proyecto\\script" {readOnly}
+RUTA_LIBS: String = "proyecto\\libs" {readOnly}
+RUTA_KEYS: String = "proyecto\\keys" {readOnly}
+RUTA_PROPIEDADES: String = "proyecto\\propiedades" {readOnly}
+CONFIGURACION_INICIAL: String = "proyecto\\script\\crearArchivoConfiguracion.bat" {readOnly}
+CONSTRUIR_DEBUG: String = "proyecto\\script\\construirDebug.bat" {readOnly}
+CONSTRUIR_RELEASE: String = "proyecto\\script\\construirRelease.bat" {readOnly}
+RESULTADO_DEPURACION: String = "proyecto\\aplicacion\\Separacion\\debug.txt" {readOnly}
+CARPETA_APLICACION: String = "proyecto\\aplicacion\\Separacion\\bin" {readOnly}
+INSTALAR_DISPOSITIVO_DEBUG: String = "proyecto\\script\\instalarDebug.bat" {readOnly}
+MOVER_ARCHIVO: String = "proyecto\\script\\moverArchivo.bat" {readOnly}
+COPIAR_ARCHIVO: String = "proyecto\\script\\copiarArchivo.bat" {readOnly}
+pruebabat: String = "proyecto\\script\\prueba.bat" {readOnly}

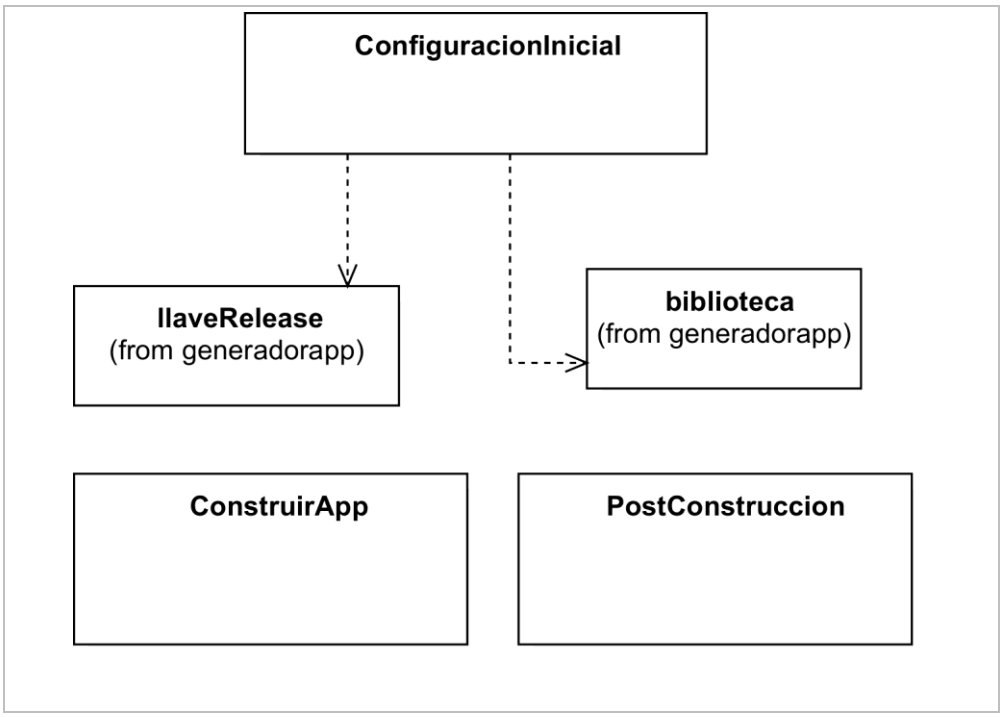

app
(from generadorapp)
+DEBUG: int = 0 {readOnly}
+RELEASE: int = 1 {readOnly}
+BASE: int = 0 {readOnly}
+BASIC_ONLY: int = 1 {readOnly}
+ORIGINAL: int = 2 {readOnly}
-HOST: String
-PORT: String
-ModoConstruccion: String
-versionAndroid: int
-aplicacion: int
«constructor»+app(aplicacion: int, HOST: String, PORT: String, ModoConstruccion, VersionAndroid)
«constructor»+app()
+setHOST(HOST: String): void
+setPORT(PORT: String): void
+setModoConstruccion(ModoConstruccion: int): void
+setVersionAndroid(VersionAndroid: int): void
+setAplicacion(aplicacion: int): void
+getHost(): String
+getPORT(): String
+getModoConstruccion(): int
+getVersionAndroid(): int
+getAplicacion(): int


biblioteca
(from generadorapp)
+INTERNA: int = 0 {readOnly}
+EXTERNA: int = 1 {readOnly}
-nombre: String
-ruta: String
-carpetaRaiz: String
-tipo: int
-libs: ArrayList
«constructor»+biblioteca()
«constructor»+biblioteca(nombre: String, ruta: String, carpetaRaiz: String, tipo: int, libs: String[*])
«constructor»+biblioteca(nombre: String, ruta: String, carpetaRaiz: String, tipo: int)
+agregarlibs(): void
+setNombre(nombre: String): void
+setRuta(ruta: String): void
+setCarpetaRaiz(carpetaRaiz: String): void
+setTipo(tipo: int): void
+setLibs(libs: ArrayList): void
+getNombre(): String
+getRuta(): String
+getCarpetaRaiz(): String
+getTipo(): int
+getLibs(): ArrayList
+getLibsEle(k: int): String

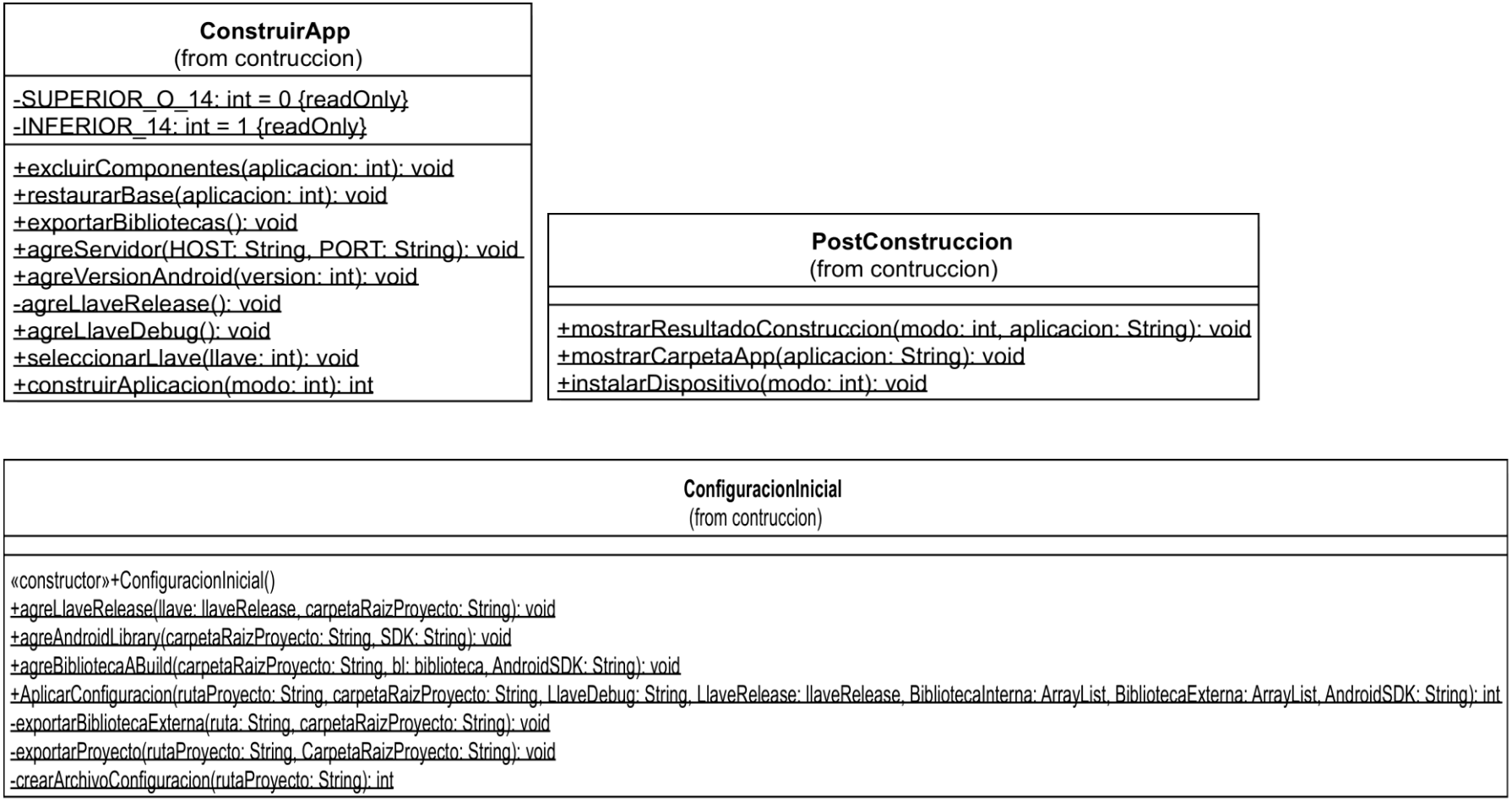
7.4.2.2. Paquete *generadorapp :: construccion*

Diagrama de clases

Se muestra el diagrama de clases de las clases en el paquete *generadorapp :: construccion*.



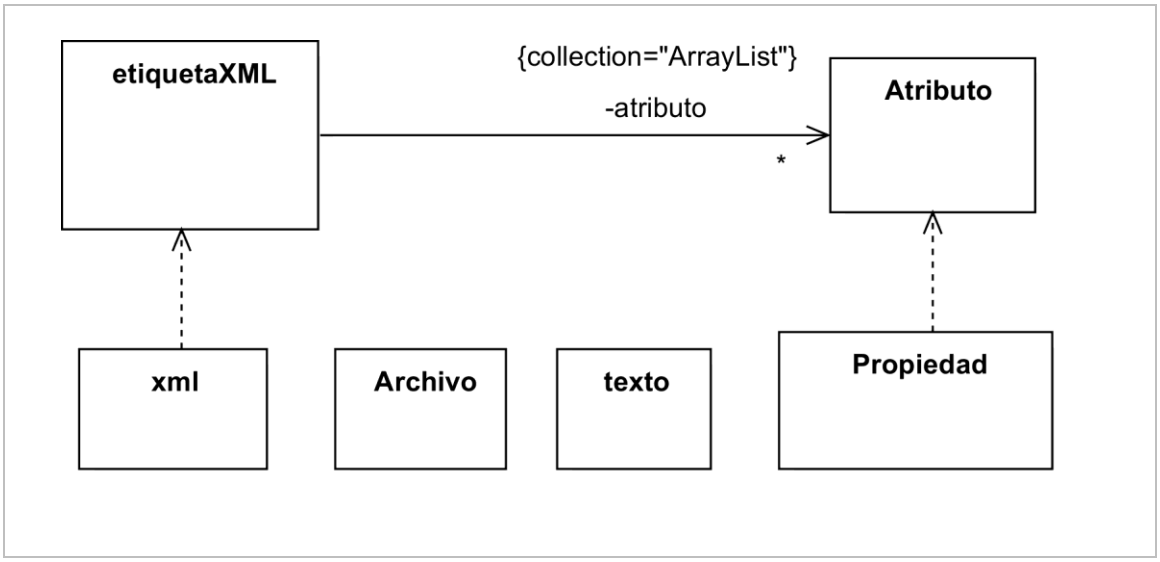
Detalle de clases



7.4.2.3. Paquete $generadorapp :: Utilidades :: archivos$

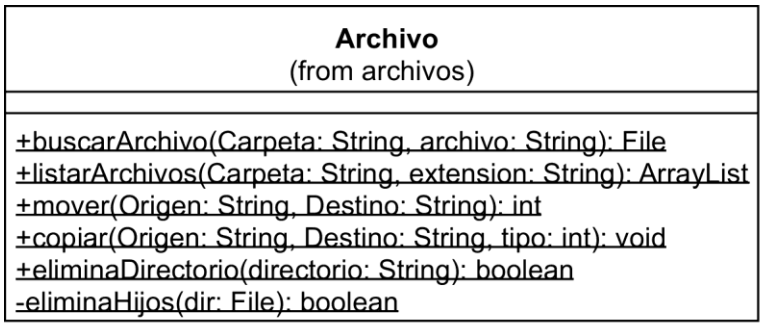
Diagrama de Clases

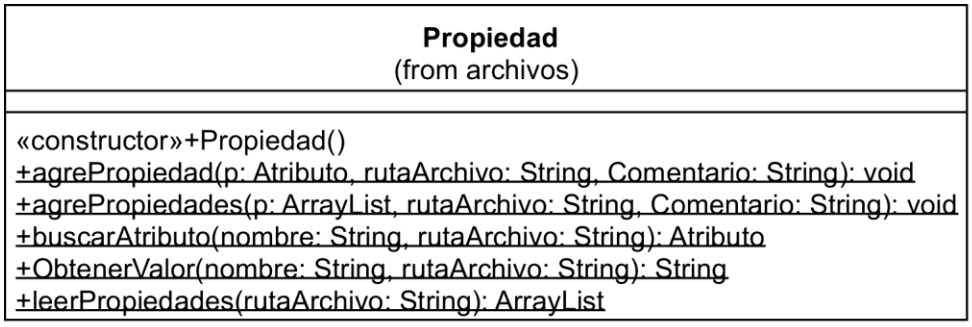
Se muestra el diagrama de clases del paquete $generadorapp :: Utilidades :: archivos$.

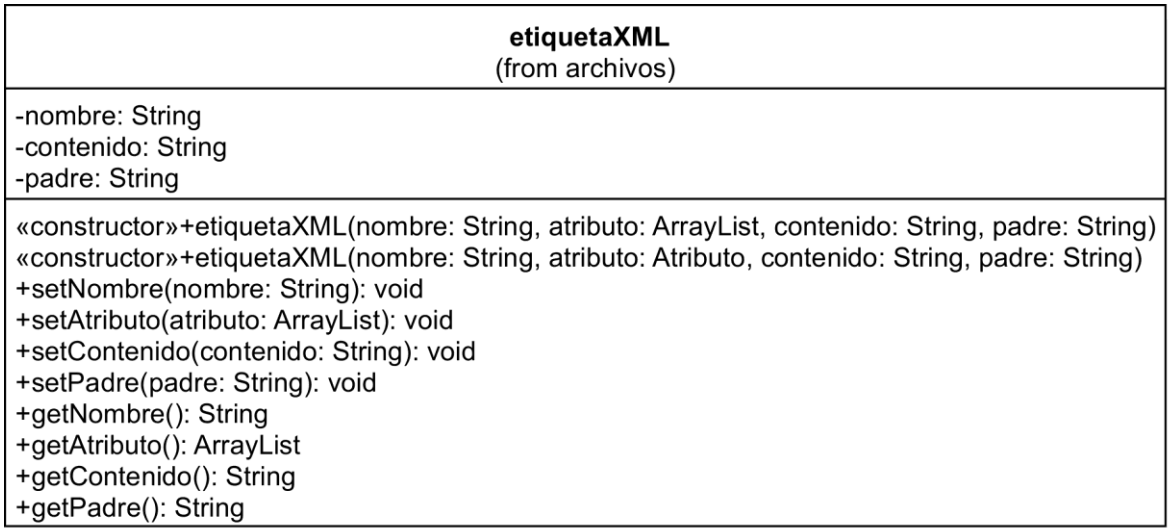


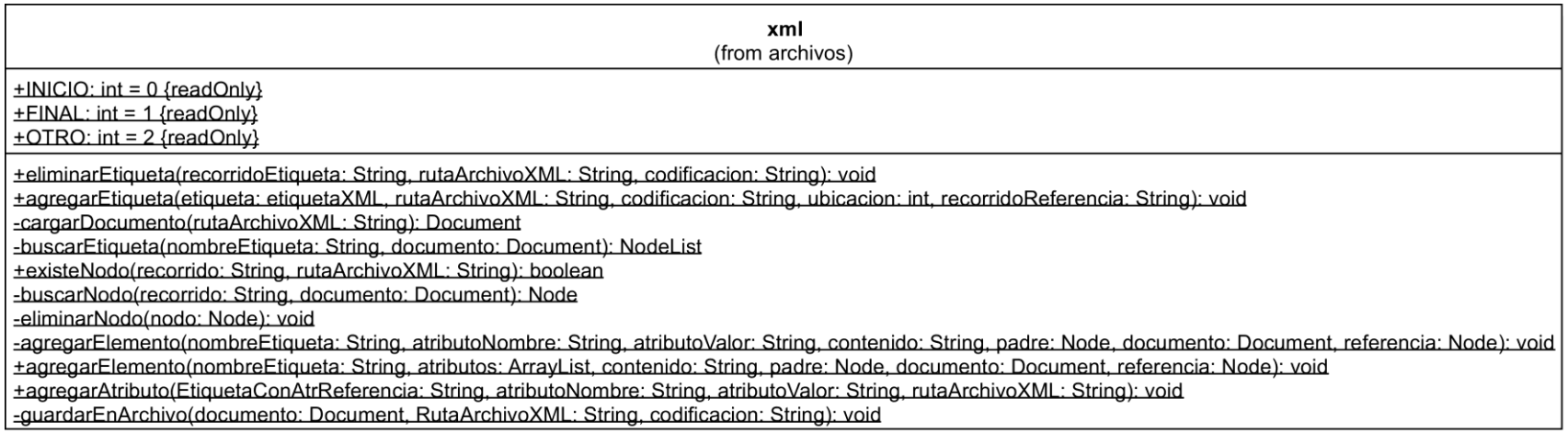
Detalle de Clases

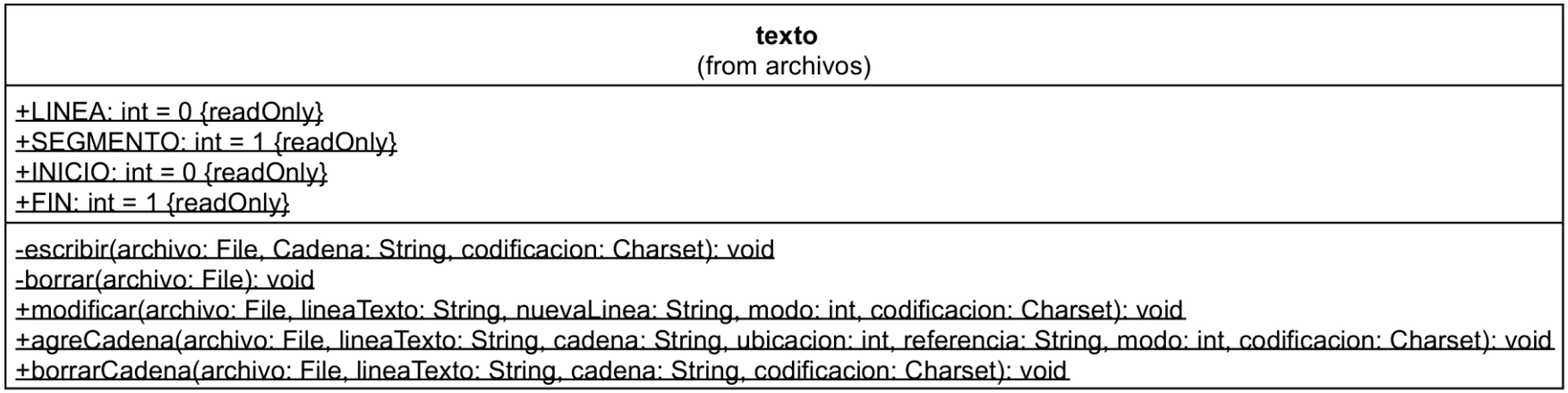
Atributo (from archivos)
-nombre: String -valor: String
«constructor»+Atributo(nombre: String, valor: String) +setNombre(nombre: String): void +setValor(valor: String): void +getNombre(): String +getValor(): String


Archivo
(from archivos)
+buscarArchivo(Carpeta: String, archivo: String): File
+listarArchivos(Carpeta: String, extension: String): ArrayList
+mover(Origen: String, Destino: String): int
+copiar(Origen: String, Destino: String, tipo: int): void
+eliminaDirectorio(directorio: String): boolean
-eliminaHijos(dir: File): boolean


Propiedad
(from archivos)
«constructor»+Propiedad()
+agrePropiedad(p: Atributo, rutaArchivo: String, Comentario: String): void
+agrePropiedades(p: ArrayList, rutaArchivo: String, Comentario: String): void
+buscarAtributo(nombre: String, rutaArchivo: String): Atributo
+ObtenerValor(nombre: String, rutaArchivo: String): String
+leerPropiedades(rutaArchivo: String): ArrayList


etiquetaXML
(from archivos)
-nombre: String
-contenido: String
-padre: String
«constructor»+etiquetaXML(nombre: String, atributo: ArrayList, contenido: String, padre: String)
«constructor»+etiquetaXML(nombre: String, atributo: Atributo, contenido: String, padre: String)
+setNombre(nombre: String): void
+setAtributo(atributo: ArrayList): void
+setContenido(contenido: String): void
+setPadre(padre: String): void
+getNombre(): String
+getAtributo(): ArrayList
+getContenido(): String
+getPadre(): String

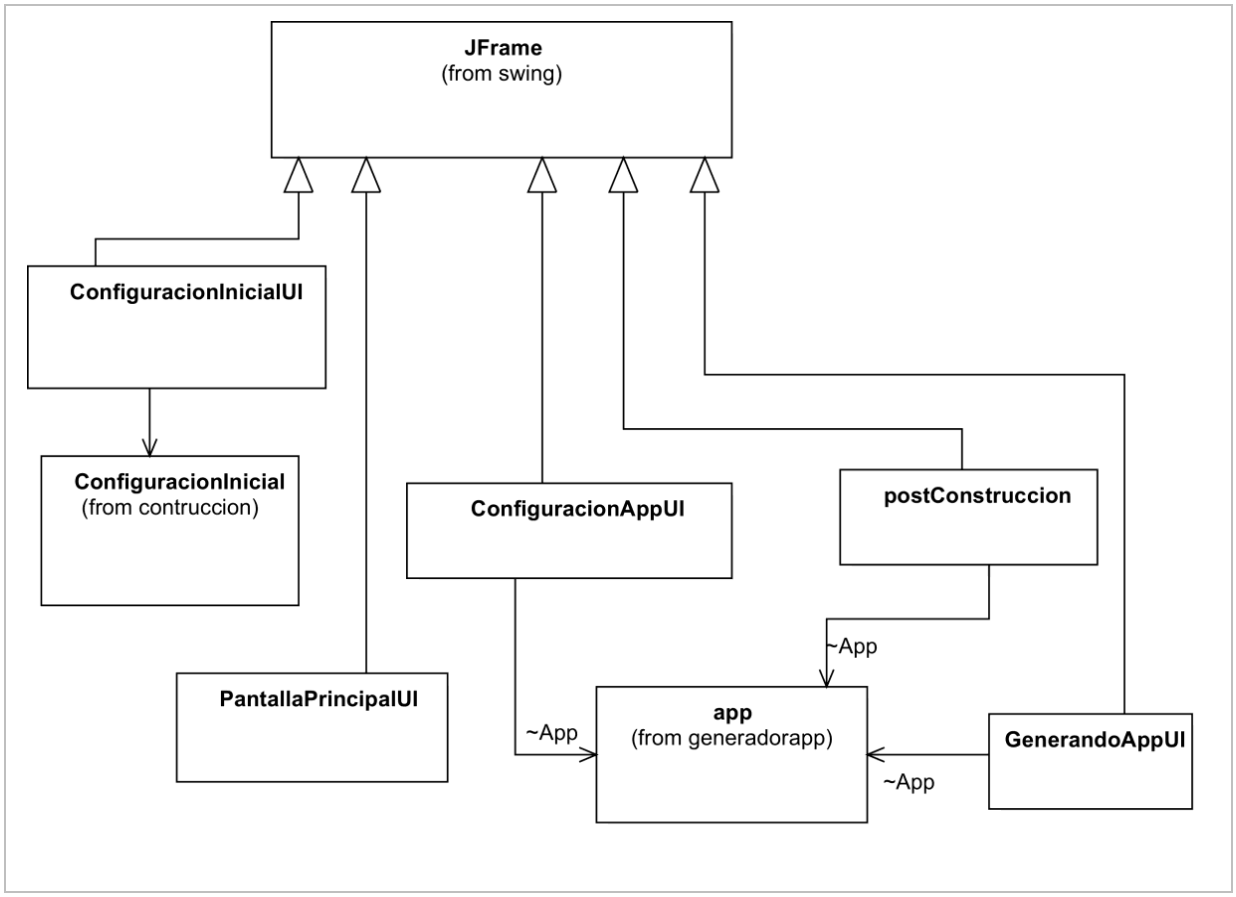

xml
(from archivos)
+INICIO: int = 0 {readOnly}
+FINAL: int = 1 {readOnly}
+OTRO: int = 2 {readOnly}
+eliminarEtiqueta(recorridoEtiqueta: String, rutaArchivoXML: String, codificacion: String): void
+agregarEtiqueta(etiqueta: etiquetaXML, rutaArchivoXML: String, codificacion: String, ubicacion: int, recorridoReferencia: String): void
-cargarDocumento(rutaArchivoXML: String): Document
-buscarEtiqueta(nombreEtiqueta: String, documento: Document): NodeList
+existeNodo(recorrido: String, rutaArchivoXML: String): boolean
-buscarNodo(recorrido: String, documento: Document): Node
-eliminarNodo(nodo: Node): void
-agregarElemento(nombreEtiqueta: String, atributoNombre: String, atributoValor: String, contenido: String, padre: Node, documento: Document, referencia: Node): void
+agregarElemento(nombreEtiqueta: String, atributos: ArrayList, contenido: String, padre: Node, documento: Document, referencia: Node): void
+agregarAtributo(EtiquetaConAtrReferencia: String, atributoNombre: String, atributoValor: String, rutaArchivoXML: String): void
-guardarEnArchivo(documento: Document, RutaArchivoXML: String, codificacion: String): void


texto
(from archivos)
+LINEA: int = 0 {readOnly}
+SEGMENTO: int = 1 {readOnly}
+INICIO: int = 0 {readOnly}
+FIN: int = 1 {readOnly}
-escribir(archivo: File, Cadena: String, codificacion: Charset): void
-borrar(archivo: File): void
+modificar(archivo: File, lineaTexto: String, nuevaLinea: String, modo: int, codificacion: Charset): void
+agreCadena(archivo: File, lineaTexto: String, cadena: String, ubicacion: int, referencia: String, modo: int, codificacion: Charset): void
+borrarCadena(archivo: File, lineaTexto: String, cadena: String, codificacion: Charset): void

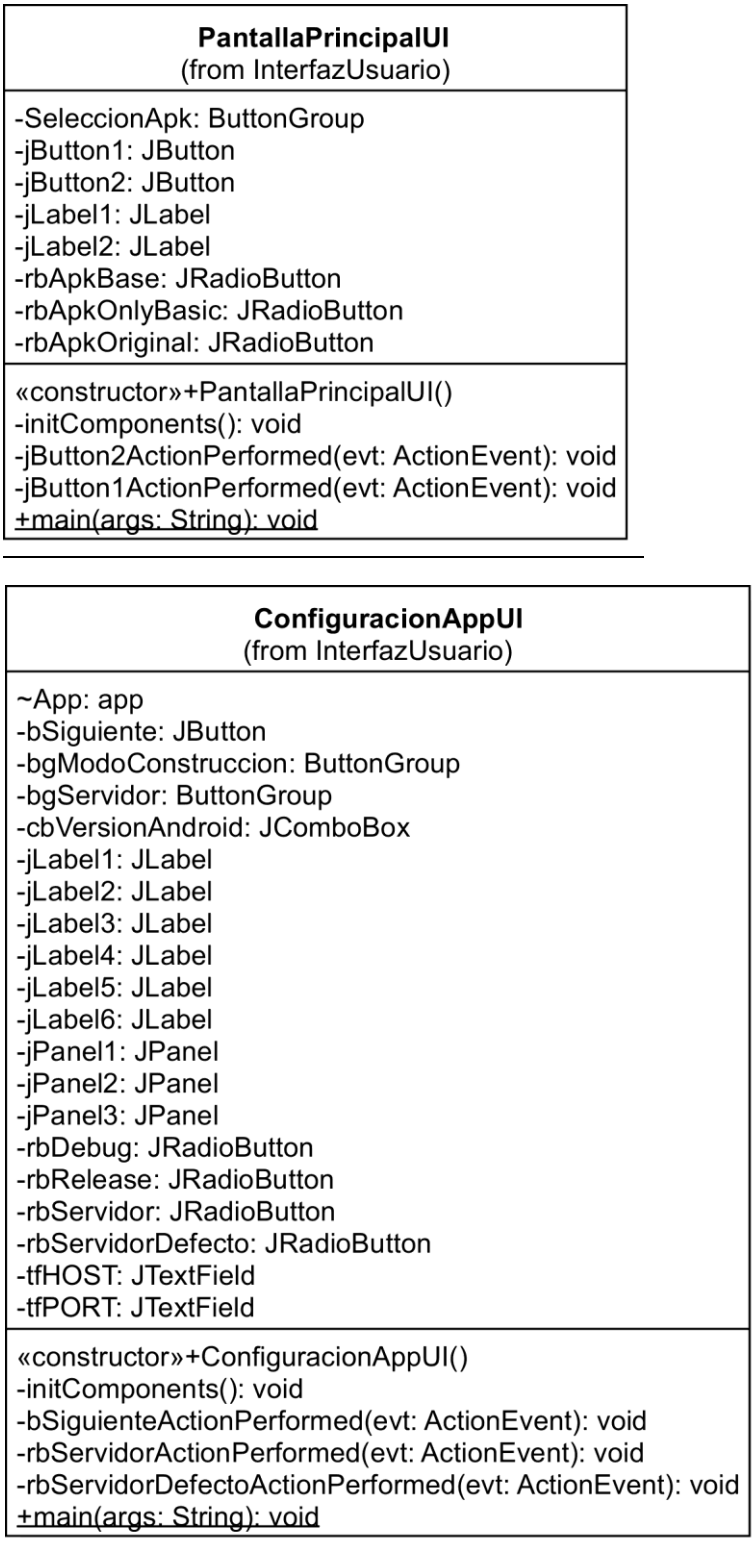
7.4.2.4. Paquete $generadorapp: IntefazUsuario$

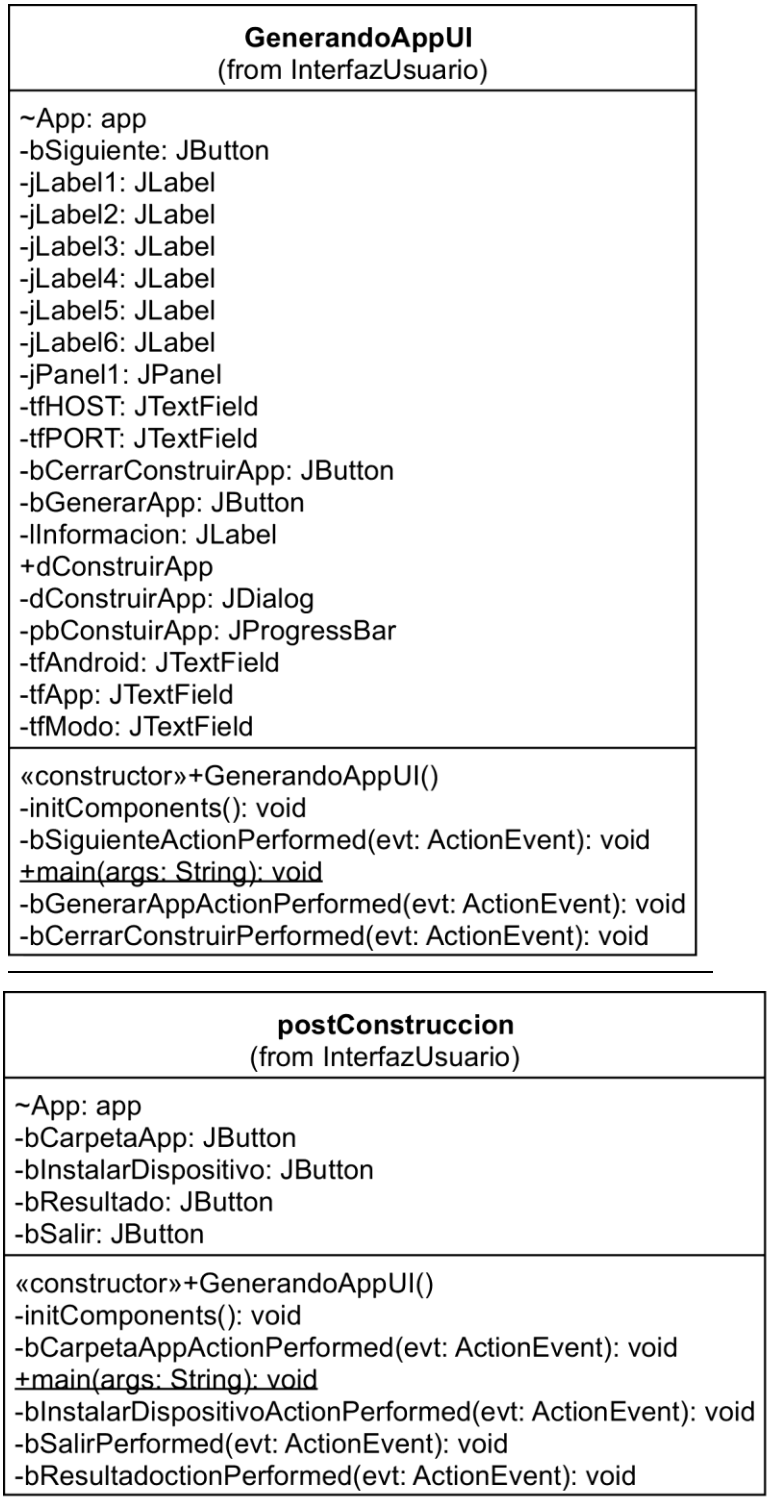
Diagrama de Clases

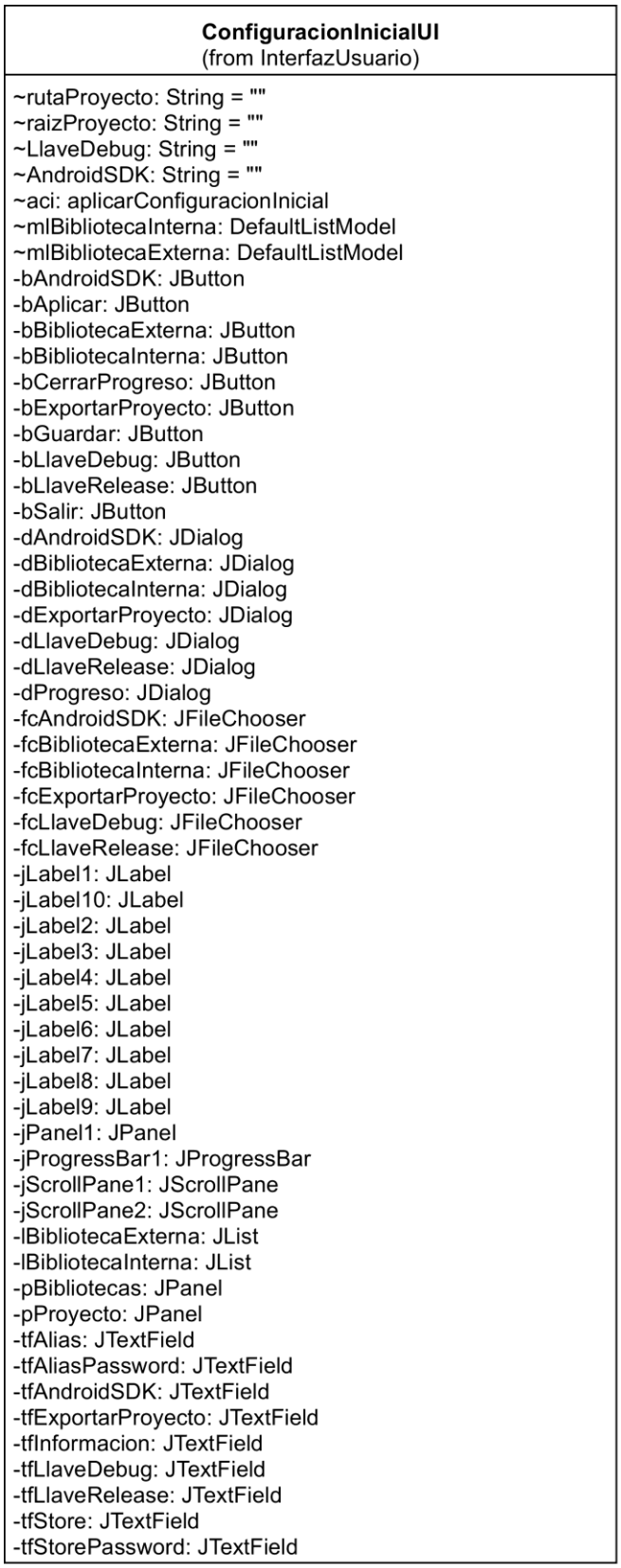
Se muestra el diagrama de clases del paquete $generadorapp: IntefazUsuario$

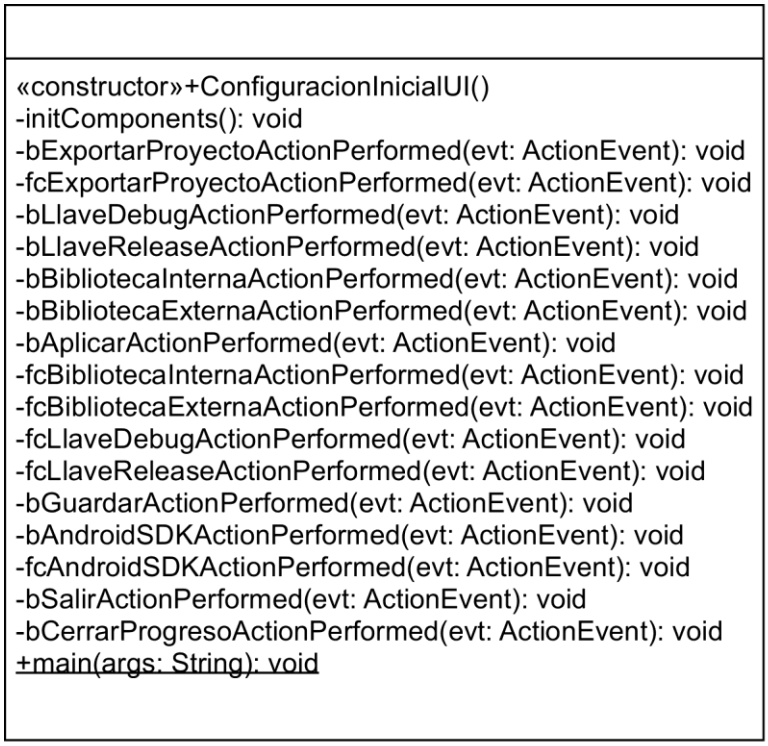


Detalle de Clases




GenerandoAppUI
(from InterfazUsuario)
~App: app
-bSiguiente: JButton
-jLabel1: JLabel
-jLabel2: JLabel
-jLabel3: JLabel
-jLabel4: JLabel
-jLabel5: JLabel
-jLabel6: JLabel
-jPanel1: JPanel
-tfHOST: JTextField
-tfPORT: JTextField
-bCerrarConstruirApp: JButton
-bGenerarApp: JButton
-lInformacion: JLabel
+dConstruirApp
-dConstruirApp: JDialog
-pbConstuirApp: JProgressBar
-tfAndroid: JTextField
-tfApp: JTextField
-tfModo: JTextField
«constructor»+GenerandoAppUI()
-initComponents(): void
-bSiguienteActionPerformed(evt: ActionEvent): void
+main(args: String): void
-bGenerarAppActionPerformed(evt: ActionEvent): void
-bCerrarConstruirPerformed(evt: ActionEvent): void
postConstruccion
(from InterfazUsuario)
~App: app
-bCarpetaApp: JButton
-bInstalarDispositivo: JButton
-bResultado: JButton
-bSalir: JButton
«constructor»+GenerandoAppUI()
-initComponents(): void
-bCarpetaAppActionPerformed(evt: ActionEvent): void
+main(args: String): void
-bInstalarDispositivoActionPerformed(evt: ActionEvent): void
-bSalirPerformed(evt: ActionEvent): void
-bResultadoctionPerformed(evt: ActionEvent): void

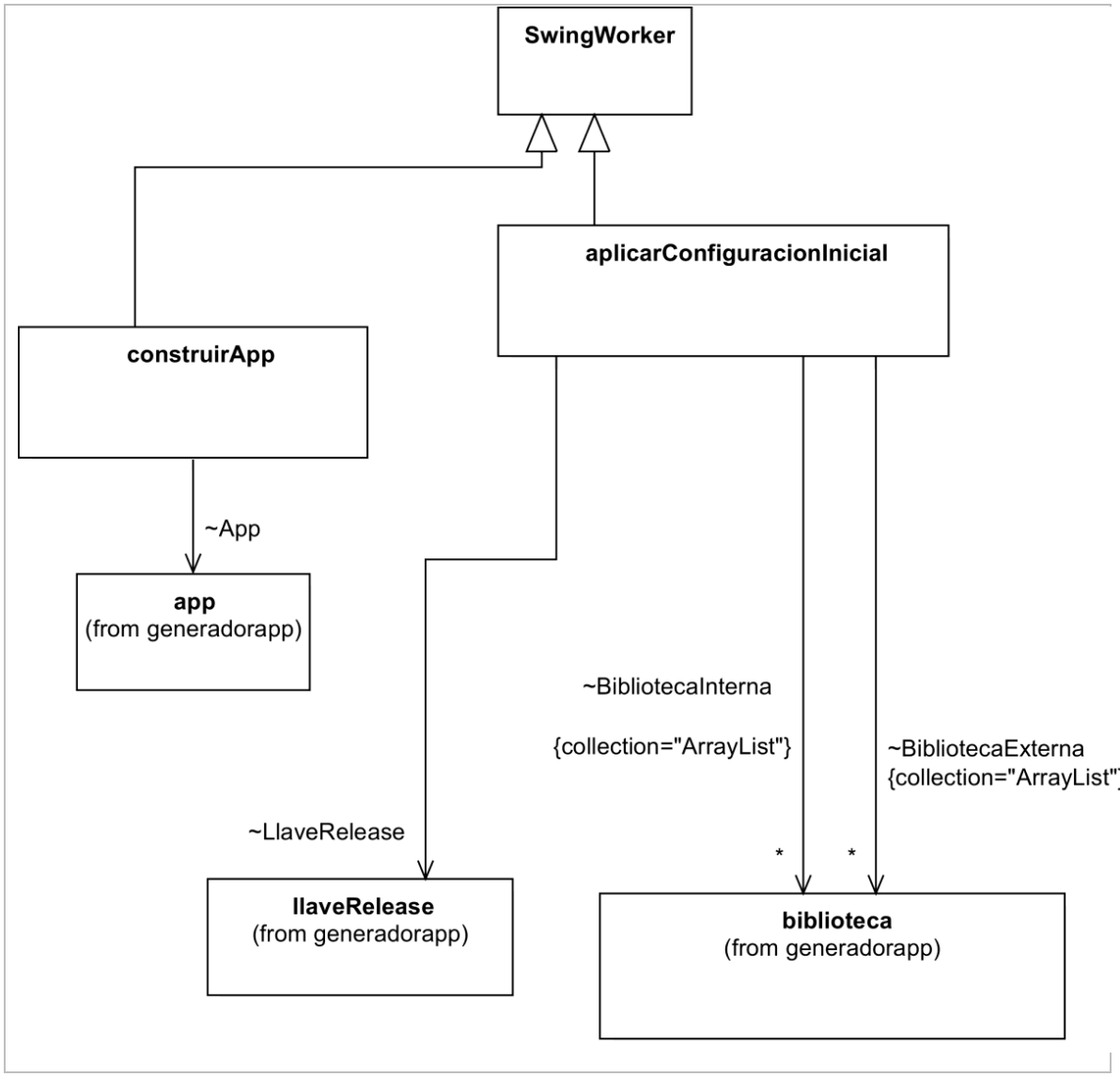

ConfiguracionInicialUI
(from InterfazUsuario)
~rutaProyecto: String = ""
~raizProyecto: String = ""
~LlaveDebug: String = ""
~AndroidSDK: String = ""
~aci: aplicarConfiguracionInicial
~mlBibliotecaInterna: DefaultListModel
~mlBibliotecaExterna: DefaultListModel
-bAndroidSDK: JButton
-bAplicar: JButton
-bBibliotecaExterna: JButton
-bBibliotecaInterna: JButton
-bCerrarProgreso: JButton
-bExportarProyecto: JButton
-bGuardar: JButton
-bLlaveDebug: JButton
-bLlaveRelease: JButton
-bSalir: JButton
-dAndroidSDK: JDialog
-dBibliotecaExterna: JDialog
-dBibliotecaInterna: JDialog
-dExportarProyecto: JDialog
-dLlaveDebug: JDialog
-dLlaveRelease: JDialog
-dProgreso: JDialog
-fcAndroidSDK: JFileChooser
-fcBibliotecaExterna: JFileChooser
-fcBibliotecaInterna: JFileChooser
-fcExportarProyecto: JFileChooser
-fcLlaveDebug: JFileChooser
-fcLlaveRelease: JFileChooser
-jLabel1: JLabel
-jLabel10: JLabel
-jLabel2: JLabel
-jLabel3: JLabel
-jLabel4: JLabel
-jLabel5: JLabel
-jLabel6: JLabel
-jLabel7: JLabel
-jLabel8: JLabel
-jLabel9: JLabel
-jPanel1: JPanel
-jProgressBar1: JProgressBar
-jScrollPane1: JScrollPane
-jScrollPane2: JScrollPane
-lBibliotecaExterna: JList
-lBibliotecaInterna: JList
-pBibliotecas: JPanel
-pProyecto: JPanel
-tfAlias: JTextField
-tfAliasPassword: JTextField
-tfAndroidSDK: JTextField
-tfExportarProyecto: JTextField
-tfInformacion: JTextField
-tfLlaveDebug: JTextField
-tfLlaveRelease: JTextField
-tfStore: JTextField
-tfStorePassword: JTextField


«constructor»+ConfiguracionInicialUI()
-initComponents(): void
-bExportarProyectoActionPerformed(evt: ActionEvent): void
-fcExportarProyectoActionPerformed(evt: ActionEvent): void
-bLlaveDebugActionPerformed(evt: ActionEvent): void
-bLlaveReleaseActionPerformed(evt: ActionEvent): void
-bBibliotecaInternaActionPerformed(evt: ActionEvent): void
-bBibliotecaExternaActionPerformed(evt: ActionEvent): void
-bAplicarActionPerformed(evt: ActionEvent): void
-fcBibliotecaInternaActionPerformed(evt: ActionEvent): void
-fcBibliotecaExternaActionPerformed(evt: ActionEvent): void
-fcLlaveDebugActionPerformed(evt: ActionEvent): void
-fcLlaveReleaseActionPerformed(evt: ActionEvent): void
-bGuardarActionPerformed(evt: ActionEvent): void
-bAndroidSDKActionPerformed(evt: ActionEvent): void
-fcAndroidSDKActionPerformed(evt: ActionEvent): void
-bSalirActionPerformed(evt: ActionEvent): void
-bCerrarProgresoActionPerformed(evt: ActionEvent): void
+main(args: String): void

7.4.2.5. Paquete $generadorapp: IntefazUsuario: worker$.

Diagrama de Clases

Se muestra el diagrama de clases para las clases en el paquete $generadorapp: IntefazUsuario: worker$.



Detalle de Clases

construirApp (from worker)
-jProgressBar1: JProgressBar {readOnly} -lInformacion: JLabel {readOnly} -boton: JButton {readOnly} -bSiguiente: JButton {readOnly} +ca: int
«constructor»+aplicarConfiguracionInicial(App: app, jProgressBar1: JProgressBar, lInformacion: JLabel, boton: JButton, bSguitente: JButton) #doInBackground(): Void #process(list: String[*]): void #done(): void

aplicarConfiguracionInicial (from worker)
~rutaProyecto: String ~raizProyecto: String ~LlaveDebug: String ~AndroidSDK: String -jProgressBar1: JProgressBar {readOnly} -tfInformacion: JTextField {readOnly} -boton: JButton {readOnly}
«constructor»+aplicarConfiguracionInicial(rutaProyecto: String, raizProyecto: String, LlaveDebug: String, AndroidSDK: String, LlaveRelease: llaveRelease, BibliotecaInterna: ArrayList, BibliotecaExterna: ArrayList, jProgressBar1: JProgressBar, tfInformacion: JTextField, boton: JButton) #doInBackground(): Void #process(list: String[*]): void #done(): void

7.5. ANEXO E

Guía de instalación Generador App para el desarrollador

Requisitos de instalación

1. JDK 7u80 o superior
2. NetBeans IDE 8.0.2 o superior
3. Apache Ant 1.9.6 o superior
4. Windows 7

Instalación

1. Instalar JDK-

2. Instalar NetBeans-

   *Nota: Se recomienda la instalación de JDK con NetBeans disponible en http://www.oracle.com/us/technologies/java/jdk-7-netbeans-download-432126.html

3. Descomprimir ADT con android 9 y android 19 disponible en los archivos del proyecto, puede utilizar cualquier ubicación, aunque se recomienda descomprimir directamente en el disco C.

4. Descomprimir apache ant (Se recomienda descomprimir directamente en C).

5. Añadir al Path del sistema operativo JDK, Ant y Android.

   Los siguientes pasos son comunes para las 3 variables de entorno.

   5.1. Dirigase a la barra de tareas y presione el botón de inicio .
   5.2. Presione con el botón secundario del mouse sobre "Equipo" y seleccione "Propiedades" en el menú desplegable. (también puede dirigirse directamente a "*Panel de control\Sistema y seguridad\Sistema*").

5.3. Presione sobre “Cambiar configuración” se abrirá la ventana “Propiedades del sistema”.

5.4. En la pestaña “Opciones avanzadas” presione el botón “Variables de Entorno…” Se abrirá la ventana “Variables de entorno”.

5.5. Añadir JDK al Path del sistema operativo.

5.5.1. En la sección “Variables del sistema” presione el botón “Nueva…”.

5.5.2. En “Nombre de la variable” escriba JAVA_HOME.

5.5.3. En “Valor de la variable” escriba el directorio donde se instaló jdk. Ejemplo: “*C:\Program Files\Java\jdk1.8.0_111*” (Sin comillas)

5.5.4. Presione Aceptar.

5.5.5. En la sección “Variables del sistema” busque la variable llamada “Path” y presione el botón “Editar…”.

5.5.6. En “Valor de la variable” escriba al final sin borrar lo que ya se encuentra escrito *;%JAVA_HOME%\bin* y presione “Aceptar”.

5.6. Añadir Apache Ant al Path del sistema operativo.

5.6.1. En la sección “Variables del sistema” presione el botón “Nueva…”

5.6.2. En “Nombre de la variable” escriba ANT_HOME

5.6.3. En “Valor de la variable” escriba el directorio raíz de Ant Ejemplo: “*C:\apache-ant-1.9.6*” (Sin comillas)

5.6.4. Presione Aceptar.

5.6.5. En la sección “Variables del sistema” busque la variable llamada “Path” y presione el botón “Editar…”.

5.6.6. En “Valor de la variable” escriba al final sin borrar lo que ya se encuentra escrito *;%ANT_HOME%\bin* y presione “Aceptar”.

5.7. Añadir Android al Path del sistema operativo.

5.7.1. En la sección “Variables del sistema” presione el botón “Nueva…”.

5.7.2. En “Nombre de la variable” escriba ANDROIDSDK_HOME.

5.7.3. En “Valor de la variable” escriba el directorio raíz del SDK de Anroid Ejemplo: “*C:\adt-bundle-windows-x86_64-20140702\sdk*” (Sin comillas).

5.7.4. Presione Aceptar.

5.7.5. En la sección “Variables del sistema” busque la variable llamada “Path” y presione el botón “Editar…”.

5.7.6. En “Valor de la variable” escriba al final sin borrar lo que ya se encuentra escrito lo siguiente:
;%ANDROIDSK_HOME%\platform-tools;%ANDROIDSK_HOME%\tools y presione “Aceptar”.

6. Importar proyecto al directorio de trabajo.
   6.1. Abra Netbeans
   6.2. Diríjase a File/Import Proyect/From Zip, se abrirá una ventana
   6.3. En ZIP File presione "Browse…" y busque el archivo Generador App.zip incluido en los archivos del proyecto.
   6.4. En Folder elija una carpeta para importar el proyecto o utilize la carpeta por defecto.
   6.5. Presione "Import".

7. Finalmente presione "F6" para correr el proyecto.

7.6. ANEXO F

Componentes de la aplicación Mapa Móvil

7.6.1. Componentes de la lógica de negocio

Componente	Descripción			T
ConcreteUserData	Interfaz	:	UserData	B
	Requiere	:	InstancesManager	
ConcreteUserLocationReaderA	Interfaz	:	UserLocationReader	B
	Requiere	:	InternetChecker	
DBLocalRepositoryA	Interfaz	:	LocalRepository	B
	Requiere	:		
XMLLocalRepositoryA	Interfaz	:	LocalRepository	B
	Requiere	:		
ConcreteLocalRepositoryReader	Interfaz	:	LocalRepositoryReader	B
	Requiere	:	LocalRepository DigitalResourceManager	
ConcreteLocalRepositorySynchronizer	Interfaz	:	LocalRepositorySynchronizer	B
	Requiere	:	UserData InfoDevice InternetChecker EventDispatchThread DebugHelper JSONParser InstancesManager LocalRepositoryReader EntitiesCreator EmergencyPriorities	

			StaticResourceTypes DynamicResourceTypes DigitalResourceTypes StringUtilities	
UserLocationSender	Interfaz	:	OptionalTask	O
	Requiere	:	*UserLocationReader* *UserData* *LocalRepositorySynchronizer* *DebugHelper* *JSONParser*	
ConcreteUserStatusSender	Interfaz	:	UserStatusSender	O
	Requiere	:	UserData InfoDevice InternetChecker EventDispatchThread DebugHelper JSONParser	
GMapA	Interfaz	:	Map	B
	Requiere	:	InfoBubbleManager PointOfInterestCreator ReflectionUtilities	
ConcreteInfoBubbleManager	Interfaz	:	InfoBubbleManager	B
	Requiere	:	StringUtilities LocalRepositoryReader UserData DigitalResourceManager	
EmergencyLayer	Interfaz	:	SingleLayer	B
	Requiere	:	Map LocalRepositoryReader PointOfInterestCreator	

			IconsManager TimeUtilities EmergencyPriorities DynamicResourceTypes	
ResourcesLayer	Interfaz	:	MultiLayer	B
	Requiere	:	Map InstancesManager LocalRepositoryReader UserData DigitalResourceManager PointOfInterestCreator CoordinateUtilities DistanceUtilities TimeUtilities StringUtilities StaticResourceTypes DynamicResourceTypes DigitalResourceTypes	
UserLocationLayer	Interfaz	:	SingleLayer	B
	Requiere	:	Map UserData UserLocationReader LocalRepositoryReader PointOfInterestCreator IconsManager TimeUtilities	
EmergencyDirectionLayer	Interfaz	:	SingleLayer	O
	Requiere	:	Map UserData	

			UserLocationReader PointOfInterestCreator IconsManager CoordinateUtilities	
ConcreteDigitalResourceManager	Interfaz	:	DigitalResourceManager	B
	Requiere	:	InfoDevice UserData	
ConcreteDigitalResourcesOpener	Interfaz	:	DigitalResourcesOpener	O
	Requiere	:	DigitalResourceManager PictureOpener VideoOpener DigitalResourceTypes	
ConcretePictureOpenerA	Interfaz	:	PictureOpener	O
	Requiere	:		
ConcreteVideoOpenerA	Interfaz	:	VideoOpener	O
	Requiere	:		
ConcretePictureMakerA	Interfaz	:	PictureMaker	O
	Requiere	:		
ConcreteVideoMakerA	Interfaz	:	VideoMaker	O
	Requiere	:		
ConcreteDigitalResourceDownloader	Interfaz	:	DigitalResourceDownloader	O
	Requiere	:	UserData InfoDevice InternetChecker EventDispatchThread DigitalResourceManager DebugHelper JSONParser	
ConcreteDigitalResourceSender	Interfaz	:	DigitalResourceSender	O

	Requiere	:	UserData InfoDevice InternetChecker EventDispatchThread DebugHelper JSONParser DigitalResourceTypes	
ConcreteEntitiesCreator	Interfaz	:	EntitiesCreator	B
	Requiere	:	EmergencyPriorities StaticResourceTypes DynamicResourceTypes DigitalResourceTypes IconsManager	
	Tipo	:	Básico	
ConcreteEmergencyPriorities	Interfaz	:	EmergencyPriorities	B
	Requiere	:	ReflectionUtilities	
ConcreteStaticResourceTypes	Interfaz	:	StaticResourceTypes	B
	Requiere	:	ReflectionUtilities	
ConcreteDynamicResourceTypes	Interfaz	:	DynamicResourceTypes	B
	Requiere	:	ReflectionUtilities	
ConcreteDigitalResourceTypes	Interfaz	:	DigitalResourceTypes	B
	Requiere	:	ReflectionUtilities	
	Tipo	:	Básico	
ConcretePointOfInterestCreator	Interfaz	:	PointOfInterestCreator	B
	Requiere	:		
ConcreteInstancesManager	Interfaz	:	InstancesManager	B
	Requiere	:	InfoDevice	
	Tipo	:	Básico	
ConcreteIconsManager	Interfaz	:	IconsManager	B

	Requiere	:		
ConcreteCoordinateUtilities	Interfaz	:	CoordinateUtilities	B
	Requiere	:		
	Tipo	:	Básico	
ConcreteDistanceUtilities	Interfaz	:	DistanceUtilities	B
	Requiere	:		
ConcreteTimeUtilities	Interfaz	:	TimeUtilities	B
	Requiere	:		
ConcreteStringUtilities	Interfaz	:	StringUtilities	B
	Requiere	:	ReflectionUtilities	
ConcreteReflectionUtilities	Interfaz	:	ReflectionUtilities	B
	Requiere	:		
ConcreteInternetCheckerA	Interfaz	:	InternetChecker	B
	Requiere	:		
ConcreteEventDispatchThreadA	Interfaz	:	EventDispatchThread	B
	Requiere	:		
ConcreteInfoDeviceA	Interfaz	:	InfoDevice	B
	Requiere	:		
ConcreteDebugHelperA	Interfaz	:	DebugHelper	B
	Requiere	:		
ConcreteJSONParser	Interfaz	:	JSONParser	B
	Requiere	:		

T(Tipo): B=componentes básicos, O=componentes opcionales

7.6.2. Componentes generales

Componente	Descripción		
ConcreteComponentsManagerA	Interfaz	:	ComponentsManager
	Paquete	:	*cl.ucsc.projectoftitle.newmobilemap.components.compmanagers*
ConcreteBasicComponentsA	Interfaz	:	BasicComponents
	Requiere	:	*cl.ucsc.projectoftitle.newmobilemap.components.compmanagers.basic*
ConcreteOptionalComponentsA	Interfaz	:	OptionalComponents
	Requiere	:	*cl.ucsc.projectoftitle.newmobilemap.components.compmanagers.optional*

7.6.3. Componentes de interfaz de usuario

Componente	Descripción			B/O*
EmergenciesList	Requiere	:	LocalRepositoryReader LocalRepositorySynchronizer OptionalTask UserLocationReader ReflectionUtilities CoordinateUtilities DistanceUtilities TimeUtilities	B
AccessCheckerToUserLocation	Requiere	:	UserLocationReader	B
EmergencyMenu	Requiere	:	LocalRepositoryReader LocalRepositorySynchronizer UserLocationReader UserData UserStatusSender ReflectionUtilities	B
UserStatusForm(*)	Requiere	:	LocalRepositoryReader LocalRepositorySynchronizer UserLocationReader UserData UserStatusSender	B
EmergencyMap	Requiere	:	LocalRepositoryReader LocalRepositorySynchronizer UserLocationReader	B

			UserData InfoBubbleManager Map SingleLayer MultiLayer DigitalResourceDownloader DigitalResourcesOpener ReflectionUtilities CoordinateUtilities DistanceUtilities	
DynamicResourcesList		Requiere	:	LocalRepositoryReader LocalRepositorySynchronizer UserLocationReader UserData DynamicResourceTypes ReflectionUtilities CoordinateUtilities DistanceUtilities TimeUtilities	B
DigitalResourcesList		Requiere	:	LocalRepositoryReader LocalRepositorySynchronizer UserLocationReader UserData ReflectionUtilities StringUtilities TimeUtilities DigitalResourceManager DigitalResourceDownloader DigitalResourcesOpener	B

DigitalResourcesMaker	Requiere	:	LocalRepositoryReader LocalRepositorySynchronizer UserLocationReader UserData DigitalResourceManager PictureMaker(***) VideoMaker(***) ReflectionUtilities	B
DigitalResourcesForm	Requiere	:	LocalRepositoryReader LocalRepositorySynchronizer UserLocationReader UserData DigitalResourceSender DigitalResourcesOpener DigitalResourceTypes StringUtilities	B

*B/O: B=componentes básicos, O=componentes opcionales