Universidad de las Ciencias Informáticas

Título: "Aplicación de Gestión de Interfaces de Usuario para el Sistema de Emisión de Documentos de Identificación del Centro de Identificación y Seguridad Digital"

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

AUTORES: **Maikel de la Torre Luis**
Diovis Proenza Labadie

TUTOR: ***Ing. Yasef Barban Freixas***

Ciudad de La Habana, Cuba.
Junio del 2009

"Seamos realistas y hagamos lo imposible"

Ernesto Che Guevara

DECLARACIÓN DE AUTORÍA

Declaro ser el único autor del trabajo titulado:

Aplicación de gestión de interfaces de usuario para el Sistema de emisión de documentos de identificación del Centro de Identificación y Seguridad y autorizo a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Maikel de la Torre Luis

Diovis Proenza Labadie

Yasef Barban Freixas

DEDICATORIA

A mis padres por todo lo que me han dado, es especial a mi madre que siempre ha sido mi principal motivo para vivir.

A mi hermano Dionis por ser un ejemplo para mí.

A mi abuela Ana a quien adoro y quiero muchísimo.

Diovis Proenza

A mis padres que son lo más importante en mi vida.

A mi abuela porque este también era su sueño.

A mi familia por siempre darme fuerzas en todo.

Maikel de la Torre

AGRADECIMIENTOS

A mi compañero de tesis Maikel por su perseverancia e insistencia.

A mi tutor Yasef por su gran trabajo de orientación en el desarrollo de la tesis.

A los amigos de todos los años de universidad, los que están y los que no.

A todos aquellos que de una forma u otra ayudaron a la realización de este trabajo.

Diovis Proenza

Maikel de la Torre

A mi novia por apoyarme siempre, darme fuerzas y mucho amor.

A todos los amigos que he encontrado durante estos cinco años en la universidad.

A mi compañero de tesis, que sin su ayuda esto no hubiera sido posible.

A mi tutor y todos los compañeros del proyecto que nos ayudaron.

RESUMEN

El uso de aplicaciones informáticas se ha generalizado y en la actualidad constituyen el eje de operaciones de cualquier empresa, brindado muchas facilidades y mejoras que posibilitan a dichas empresas brindar un mejor servicio. Una de las causas fundamentales del fracaso de estas aplicaciones es la rigidez que les imposibilita adaptarse a los cambios en los procesos de negocio que informatizan. Esto trae como consecuencia que cada vez que ocurra algún cambio en estos procesos, los desarrolladores tengan que hacer cambios en las aplicaciones lo que conlleva aumento de costos y pérdida de tiempo.

En el Centro de Identificación y Seguridad Digital de la Universidad de las Ciencias Informáticas existe la propuesta de desarrollar un sistema de emisión de documentos de identificación capaz de adaptarse a los cambios que puedan ocurrir en estos procesos. Este sistema estará compuesto por varias aplicaciones, una de las cuales se encargará de gestionar las interfaces de usuario a través de las cuales se obtendrá toda la información a tramitar.

En el presente trabajo se analizan los frameworks relacionados con la gestión de interfaces de usuario. Se lleva a cabo un estudio de las tendencias actuales en el mundo de la informática, con el propósito de determinar la metodología de desarrollo a utilizar para el desarrollo de la aplicación, así como la herramienta para el modelado de la misma. Además se realiza el análisis, diseño e implementación de la aplicación para la gestión de las interfaces de usuario que utilizará el Sistema de Emisión de Documentos de Identificación del Centro de Identificación y Seguridad Digital. Posteriormente se realizan las pruebas para la validación de la propuesta de solución.

ÍNDICE

ÍNDICE

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Debido al gran desarrollo científico que tuvo lugar a mediados del siglo pasado, los avances en los sectores de la informática sacudieron al mundo. A raíz de esto comienzan a surgir empresas y organizaciones dedicadas a desarrollar y brindar soluciones informáticas. Dentro de estas soluciones se encuentran las aplicaciones empresariales las cuales actualmente son el eje de las operaciones empresariales ya que regulan las actividades esenciales que ayudan a mejorar la satisfacción del cliente y a acelerar el crecimiento de la empresa. Existen dos tipos de aplicaciones, las aplicaciones web y las aplicaciones de escritorio. En el caso de las web podemos mencionar que son independientes del sistema operativo y no requieren de complicadas combinaciones hardware/software, solo requieren de un ordenador con un navegador y conexión a Internet mientras que las de tipo escritorio son más rápidas, potentes y presentan modelos de interacción más robustos.

En el diseño de cualquier aplicación informática con cierta complejidad, independientemente de si es web o de escritorio, uno de los aspectos fundamentales es la interfaz gráfica de usuario, la cual es el artefacto tecnológico que posibilita, a través del uso y la representación del lenguaje visual, la interacción del usuario con el ordenador. Dado a que en la actualidad, el uso de estas interfaces es generalizado, se debe prestar mucha atención a la implementación de las mismas debido a que se tiende fácilmente a mezclar las operaciones de manipulación de la información (lógica de negocio) con la presentación visual de esa información (lógica de interfaz de usuario). Esto trae como consecuencia que el código esté todo mezclado, sea difícil identificar que hace cada cosa y sea muy complicado corregir errores, además en caso de ocurrir cambios en la lógica de negocio, la implementación de la misma se vería afectada y por tanto necesitaría reimplementarse, repercutiendo directamente en la implementación de la interfaz de usuario. Esto además trae consigo un esfuerzo adicional por parte de los programadores que, en muchas ocasiones, tienen que reimplementar la lógica de interfaz de usuario y rediseñar las interfaces parcial o completamente. Este proceso provoca un incremento del tiempo y costo de desarrollo, además de crear dependencia entre los contratadores y los desarrolladores del sistema.

Dentro de la amplia gama de aplicaciones que se desarrollan actualmente, se encuentran las que se encargan de manipular la información referente a la identificación de los ciudadanos. Estos sistemas tienen como objetivo principal la emisión de los documentos de identificación establecidos bajo las leyes constitucionales de cada país, como cédulas y pasaportes. Estas aplicaciones no escapan de

los problemas que trae consigo la implementación de la lógica de negocio sobre la interfaz de usuario, pues los cambios que pudieran ocurrir en los procesos de emisión de documentos de identificación afectarían directamente las interfaces de usuarios que soportan dichos procesos.

El Centro de Identificación y Seguridad Digital de la Universidad de las Ciencias Informáticas (UCI) pretende desarrollar un sistema que permita manejar los procesos de emisión de documentos de identificación y que solucione los problemas planteados anteriormente mediante la creación de una aplicación de gestión de interfaces de usuario que separe la presentación de la información de la lógica de negocio.

Situación Problémica

Un gran número de problemas pueden presentarse cuando las aplicaciones mezclan el código de la lógica de negocios y el código de presentación. Tales aplicaciones son difíciles de mantener, en razón de que las interdependencias entre todos los componentes causan efectos colaterales cada vez que un cambio se realiza en algún lado.

Problema a resolver

¿Cómo gestionar interfaces de usuario para el Sistema de Emisión de Documentos de Identificación del Centro de Identificación y Seguridad Digital?

Objeto de Estudio

Gestión de interfaces de usuario.

Objetivos

Objetivo general:

Desarrollar una aplicación de gestión de interfaces de usuario para el Sistema de Emisión de Documentos de Identificación del Centro de Identificación y Seguridad Digital.

Objetivos específicos:

- Analizar aplicaciones que gestionen interfaces de usuario (IU).
- Realizar análisis y diseño de la aplicación a desarrollar.
- Desarrollar la aplicación de gestión de IU.
- Garantizar la calidad y el funcionamiento del componente de gestión de IU concebido.

Campo de Acción

Gestión de interfaces de usuario en el Sistema de Emisión de Documentos de Identificación del Centro de Identificación y Seguridad Digital de la UCI.

Idea a defender

El desarrollo de la aplicación, garantizará la gestión de interfaces de usuario, en el Sistema de Emisión de Documentos de Identificación del Centro de Identificación y Seguridad Digital.

Tareas de Investigación

- Revisión de las soluciones más utilizadas para la gestión de IU.
- Revisión de Lenguaje de Marcado Extensible (XML por sus siglas en inglés).
- Selección de los framework para la implementación de la aplicación.
- Definición de configuración para la comunicación y la entrada y salida de datos.
- Identificación del Modelo de Dominio.
- Especificación de los requisitos funcionales y no funcionales del sistema a desarrollar.
- Modelamiento del sistema.
- Realización del análisis y diseño de la aplicación a desarrollar.
- Implementación de la propuesta de solución.
- Realización de pruebas del funcionamiento de la aplicación concebida.

Métodos de Investigación

Métodos teóricos:

- *Analítico–sintético*: Se consultará la bibliografía necesaria para dar cumplimiento a las tareas de la investigación y se resumirán los principales aspectos de cada una de ellas. Estas operaciones no son independientes, pues el análisis de un objeto se realiza a partir de la relación que existe entre los elementos que lo conforman y a su vez, la síntesis se produce sobre la base de los resultados previos del análisis.
- *Histórico – lógico*: Como está vinculado al conocimiento de las distintas etapas de los objetos en su sucesión cronológica, posibilitará conocer los antecedentes de las aplicaciones de gestión de interfaces de usuario. Mediante este método, se analizará la trayectoria concreta de la teoría, su condicionamiento a los diferentes períodos de la historia y al basarse en el

desarrollo histórico, pondrá de manifiesto la lógica interna de su desarrollo, su teoría y encontrará el conocimiento más profundo de esta, su esencia.

- *Método de modelación*: Permitirá la creación de modelos, es decir representar lo que se quiere estudiar de forma más simple, explicando lo que pasa de una manera lógica.

Estructura del Trabajo

- **Capítulo 1:** Fundamentación teórica: En este capítulo se realiza un estudio del estado del arte revisando algunos de los sistemas más utilizados para la gestión de interfaces de usuarios exponiendo las ventajas y las desventajas de cada uno. Se realizará un análisis de las metodologías, herramientas y lenguajes que existen en la actualidad y que pudieran ser útiles en el desarrollo de la solución justificando la selección de los mismos.

- **Capítulo 2:** Características del sistema: Se analizarán los procesos de emisión de documentos de identificación, a los cuales tributará la aplicación a desarrollar. Se presentará una descripción general de la propuesta de sistema, se determinaran los conceptos fundamentales para realizar el modelo de dominio, se identificará los requisitos funcionales y no funcionales del sistema a desarrollar y se describirán en términos de casos de usos.

- **Capítulo 3:** Análisis y diseño: Se realizará el análisis y diseño donde se obtendrán los artefactos necesarios para la posterior implementación del componente entre los que podemos mencionar los diagramas de clases del análisis y del diseño y los diagramas de interacción. En este capítulo además se determinará la arquitectura y los patrones de diseño a utilizar en la implementación.

- **Capítulo 4:** Implementación y pruebas: Se llevará a cabo la implementación de la aplicación, obteniéndose los artefactos correspondientes como son los diagramas de componentes y el de despliegue, además posteriormente se le realizarán las pruebas pertinentes. También se realizará un análisis de los beneficios de la aplicación.

CAPÍTULO I. FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

Actualmente merece mucha importancia la interfaz gráfica de usuario a la hora de crear aplicaciones informáticas y para desarrollar las mismas se utilizan los frameworks de desarrollo que más ventajas proporcionan. Se tienen en cuenta también, las tendencias actuales para el desarrollo de aplicaciones con el objetivo de lograr un producto lo más adecuado posible a los requerimientos de los clientes, en el menor tiempo, además de estar basado en las tecnologías y métodos de desarrollo más actualizados.

1.2 Sistemas existentes en el Centro de Identificación y Seguridad Digital

El Centro de Identificación y Seguridad Digital de la UCI, ha desarrollado un sistema de emisión de documentos de identificación llamado Servicio Autónomo de Identificación, Migración y Extranjería (SAIME), el cual se encuentra en funcionamiento desde marzo del año 2007 en Venezuela con muy buenos resultados. Esta solución es una aplicación de escritorio que integra varios módulos y subproyectos, como son Identificación, que incluye el proceso de Cedulación y Emisión de Pasaporte; Migración para el control migratorio en Aeropuertos, puertos y puntos fronterizos y Extranjería.

Este sistema se realizó con el fin de mejorar la identidad de los ciudadanos de Venezuela donde los documentos de identificación eran muy fáciles de falsificar, los pasaportes se demoraban años en obtenerlos y a un precio elevado, además no existía control de los que entraban o salían del país. Para el enrolamiento de datos este sistema utiliza una aplicación de escritorio que permite conectarse con dispositivos de captura de datos biométricos, para esto se encuentra integrado con un sistema de AFIS Civil para el procesamiento de datos biométricos de los ciudadanos, de la empresa SAGEM de Francia y con productos para la emisión de pasaportes electrónicos de la empresa Bundesdruckerei de Alemania.

Este sistema resuelve la problemática planteada en esta investigación, a través del framework *Común*, el cual separa la interfaz de la lógica de negocio permitiendo la reutilización de las interfaces, para esto este framework se divide en varias capas, la capa de interfaz, la de gestión de interfaz, la capa de clases controladoras, y una de acceso a datos. En la capa de interfaz, la interfaz de usuario se implementa mediante formularios, y solo puede ir código referente a la misma. En la capa de gestión de interfaz es de donde se manejan los eventos que hacen referencia al negocio. En la capa de clases controladoras se definen las acciones que se pueden realizar, es decir las actividades que se realizan

en un proceso, una acción es la mediadora entre la capa de interfaz con la que el usuario interactúa, y la capa de negocio, y la capa de acceso a datos se emplea para persistir información que requiera el framework para su funcionamiento.(LANDRIAN, 2008)

1.3 Conceptos básicos asociados al dominio del problema

La interfaz realiza la función de mediación entre sistemas distintos, lo que hace posible la comunicación entre ellos. En la informática se define la **interfaz de usuario** (IU) como "*uno de los componentes más importantes de cualquier sistema computacional, pues funciona como el vínculo entre el humano y la máquina. La interfaz de usuario es un conjunto de protocolos y técnicas para el intercambio de información entre una aplicación computacional y el usuario"* (LARSON, 1992). Entre estas interfaces podemos encontrar las **gráficas**, muchas veces referida como GUI por sus siglas en inglés (Graphical User Interfaces), las cuales son "*un tipo de interfaz de usuario que se caracteriza y diferencia por el hecho de utilizar un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz"(SHNEIDERMAN, 1998)*.

La **gestión**, definida por la Real Academia Española como el "*efecto de gestionar o administrar",* se refiere a la realización de un grupo de actividades conducentes al logro de un negocio o de un deseo cualquiera. Uno de los elementos que requieren de una exhaustiva gestión dentro de las aplicaciones informáticas son las interfaces de usuario. Este es un tema que ha ido evolucionando en los últimos tiempos y en la actualidad existen numerosos frameworks dedicados a la gestión de interfaces a través de la creación y manipulación de las mismas. El proceso de **gestión de interfaz de usuario** se define como la acción de crear, cargar, actualizar, y eliminar las interfaces.

Entre los diferentes tipos de **aplicaciones** se encuentran las **de escritorio** las cuales facilitan la navegación en las interfaces de usuario, brinda a los usuarios una vía de rápido acceso a los recursos locales como el disco duro, el teclado, el ratón así como dispositivos de captura de imagen, impresoras y otros periféricos. Además estas aplicaciones permiten a los usuarios una manera fácil de acceder a aplicaciones locales en caso de que sea necesario compartir datos entre ellas. Una de las ventajas fundamentales de este tipo de aplicación es la de ofrecer mayor capacidad gráfica visual, menor tiempo de respuesta a las acciones y una mayor personalización, lo que posibilita que los usuarios se sientan a gusto a la hora de trabajar con ellas. La principal desventaja de estas aplicaciones es que requieren ser instaladas, es decir, es necesario llevar a cabo un proceso de instalación ordenador por ordenador, lo mismo ocurre a la hora de actualizar la aplicación. (DORAL, 2008)

1.4 Tendencias y tecnologías actuales más utilizadas

Una preocupación cada vez mayor de los ingenieros de software es la necesidad de mantenerse al corriente de la rápida evolución de las tecnologías. Resulta de gran importancia mantenerse actualizado en cuanto a los frameworks de desarrollo de aplicaciones, patrones arquitectónicos, estándares de comunicación, así como de las principales tendencias en el uso de estas tecnologías, para poder desarrollar una aplicación con calidad, en el tiempo requerido y que satisfaga las necesidades de los clientes.

1.4.1 Modelo Vista Controlador

Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que tiene como objetivo separar la lógica de negocio de la interfaz gráfica de manera que cambios en la misma no afecten a la lógica de negocio y viceversa. Este patrón se utiliza fundamentalmente en aplicaciones que manejan gran cantidad de datos donde es necesaria una mejor separación de conceptos para que el desarrollo esté estructurado de una mejor manera, facilitando la programación en diferentes capas de manera paralela e independiente.

Este está compuesto por tres componentes, el modelo que representa la información que tanto el usuario como la aplicación puede manipular, la vista que implica todos los elementos que componen la interfaz gráfica y el controlador que maneja la interacción y la comunicación entre el modelo y las acciones del usuario. La forma más sencilla de implementar este patrón es pensando en capas; como regla, los accesos a la base de datos se hacen en el modelo, la vista y el controlador no deben de saber si se usa o no una base de datos. El controlador es el que decide que vista se debe de imprimir y que información es la que se envía.

Entre sus ventajas podemos mencionar que facilita la agregación de múltiples representaciones de los mismos datos, crea independencia de funcionamiento, facilita la detección y eliminación de errores y permite el escalamiento de la aplicación en caso de ser requerido. Entre sus desventajas principales tenemos que la separación de conceptos en capas agrega complejidad el sistema, la cantidad de archivos a mantener se incrementa considerablemente y la curva de aprendizaje del patrón es más alta que otros patrones más sencillos. (REYNOSO, 2004)

Con el desarrollo de la aplicación de gestión IU se pretende separar la lógica de los procesos de negocio de la presentación de la información, por lo que será de gran utilidad tener en cuenta la idea propuesta por este patrón, aunque el mismo no se utilice a la hora de diseñar la solución.

1.4.2 Arquitectura orientada a servicios

La Arquitectura Orientada a Servicios (SOA por sus siglas en inglés), define la utilización de servicios para dar soporte a los requisitos del negocio. Aporta una metodología y un marco de trabajo basado en estándares. Permite la creación de sistemas altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma estándar de exposición e invocación de servicios (comúnmente pero no exclusivamente servicios web), lo cual facilita la interacción entre diferentes sistemas propios o de terceros.

Al contrario de las arquitecturas orientado a objetos, las SOAs están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma subyacente y del lenguaje de programación. La definición de la interfaz encapsula (oculta) las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo. Con esta arquitectura, se pretende que los componentes software desarrollados sean muy reusables, ya que la interfaz se define siguiendo un estándar.

Entre los principales beneficios de SOA podemos mencionar que mejora los tiempos de realización de cambios en procesos, facilita el abordar modelos de negocios basados en colaboración con otros entes (socios, proveedores) y la capacidad para reemplazar elementos de la capa aplicativa SOA sin disrupción en el proceso de negocio además de la integración de tecnologías disímiles.

En la actualidad este tipo de arquitectura está siendo muy utilizada y está llamada a ser la arquitectura del Siglo XXI. A raíz de las ventajas antes mencionadas, la aplicación de gestión de IU deberá ser capaz de integrarse a sistemas basados en este tipo de arquitectura.

1.4.3 Lenguaje de Marcado Extensible

El lenguaje de Marcado Extensible (XML por su nombre en inglés) es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del Standard Generalized Markup Language (SGML) o "Lenguaje de Marcado Generalizado" y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. XML no ha nacido sólo para su aplicación en Internet, sino que se

propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil. Entre sus potencialidades podemos mencionar que:

- Es extensible ya que después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML. Esto posibilita el empleo de cualquiera de los analizadores disponibles. De esta manera se evitan errores de software (bugs) y se acelera el desarrollo de aplicaciones.
- Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla. Mejora la compatibilidad entre aplicaciones.

XML se ha convertido en el estándar de comunicación entre aplicaciones más utilizado a nivel mundial, el cual aporta mucha potencia y flexibilidad a las mismas, además de un modo seguro para la comunicación y trasmisión de información, por lo que será de vital importancia el empleo de este tipo de tecnología.

1.4.4 Migración hacia Software Libre

Software libre es aquel que, una vez obtenido, puede ser usado, copiado, estudiado, modificado y redistribuido libremente. El software libre suele estar disponible gratuitamente en Internet, o a precio del coste de la distribución a través de otros medios; sin embargo no es obligatorio que sea así y, aunque conserve su carácter de libre, puede ser vendido comercialmente. Análogamente, el software gratuito (denominado usualmente Freeware) incluye en algunas ocasiones el código fuente; sin embargo, este tipo de software no es libre en el mismo sentido que el software libre, al menos que se garanticen los derechos de modificación y redistribución de dichas versiones modificadas del programa. (RUIZ, 2006)

La migración hacia el software libre evita la dependencia de los proveedores y se reciben mejores servicios y productos. Cuba y, en particular la UCI, han sustentado la posibilidad de migrar hacia software libre, por las ventajas que representa con respecto al software de tipo propietario. El software

libre representa la alternativa para los países pobres y, es por concepción, propiedad social. Económicamente, su utilización no implica gastos adicionales por concepto de cambio de plataforma de software, por cuanto es operable en el mismo soporte de hardware con que cuenta el país. La adquisición de cualquiera de sus distribuciones, puede hacerse de forma gratuita, descargándolas directamente de Internet o en algunos casos a muy bajos precios.

La migración hacia software libre, está planteada como una de las metas a alcanzar por los departamentos de informatización y ALBET (Empresa comercializadora de software de la UCI) para lograr la soberanía tecnológica. A raíz de esto en el Centro de Identificación y Seguridad Digital de la UCI surge una línea de desarrollo para la creación de soluciones y productos informáticos sobre los conceptos de software libre, la cual traerá muchos beneficios, sobre todo permitirá la disminución en costos de licencias.

1.5 Frameworks para el desarrollo de IU

En la actualidad existen numerosos frameworks de desarrollo de aplicaciones vinculados al proceso de gestión de interfaces de usuario, los cuales se encargan de implementar las interfaces de distintas formas, algunos tienen la capacidad de generarlas de forma automática. Por lo general los más usados a la hora de crear una aplicación son los que más facilidades brindan al desarrollador sin importar que se distribuyan bajo licencias libres o propietarias, o que estén orientados a aplicaciones de escritorio o web.

1.5.1 OpenXava

Genera automáticamente aplicaciones de gestión desde simples clases Java con anotaciones, incluyendo interfaces de usuario complejas usando simples definiciones XML, y un estilo orientado a objetos. OpenXava es una forma productiva de crear aplicaciones Web empresariales. De hecho, es mucho más rápido desarrollar las aplicaciones con OpenXava que con Ruby on Rails, Spring MVC o cualquier otro framework basado en MVC. Esto se debe a que en OpenXava solo tienes que escribir el modelo, los controladores son rehusados y las vistas se generan automáticamente.

La esencia de OpenXava es que el desarrollador define en vez de programar, y el framework provee automáticamente la interfaz de usuario, el acceso a los datos y el comportamiento por defecto. De esta manera, todo lo común se resuelve fácilmente, pero siempre el desarrollador tiene la posibilidad de programar manualmente cualquier parte de la aplicación, de esta forma es lo bastante flexible para resolver los casos particulares.

OpenXava está creado para desarrolladores Java y es muy potente, extensible y fácil de usar para expertos. No obstante permite a los menos expertos un comienzo placentero. Es lo suficientemente flexible como para crear aplicaciones sofisticadas, permite generar aplicaciones J2EE(Java 2 Enterprise Edition) completas, incluyendo la interfaz de usuario AJAX (Asynchronous JavaScript And XML), soporta varios servidores de aplicaciones como Tomcat, JBoss, WebSphere, entre otros (OPENXAVA). Se decidió no utilizarlo debido a que está orientado fundamentalmente a aplicaciones Web.

1.5.2 JMatter

Es un framework que pretende multiplicar la productividad de los programadores ya que permite centrarse en la programación de la lógica del negocio sin necesidad de ocupar tiempo trabajando en las interfaces de usuario o en la persistencia. Construye aplicaciones Swing de gestión para trabajo en grupo basándose en clases de modelo brindando varios servicios como la construcción de GUI basados en Swing que soportan operaciones CRUD, muy útiles a la hora de gestionar información desde bases de datos.

La persistencia está basada en Hibernate, uno de los frameworks de acceso a datos más utilizados en tecnología libre. Permite la autenticación y administración de usuarios y posee un soporte propio para la construcción e invocación de consultas a través de las GUIs. Luego de este análisis se llegó a la conclusión de que no se ajustaba a los requerimientos necesarios ya que se enfocaba principalmente a la gestión de información almacenada en bases de datos, además se distribuye bajo licencia propietaria o GNU Public License (GPL) que obliga a que cualquier producto que sea desarrollado utilizando este framework, tenga que distribuirse bajo la misma licencia, lo que significa que obliga a quien utilice este framework, a liberar el código del producto desarrollado. (JMATTER, 2009)

1.5.3 XForms

Es un nuevo formato XML para poder definir interfaces de usuario, principalmente formularios web. XForms ha sido diseñado para ser la nueva generación de formularios HTML/XHTML, pero es lo suficientemente genérico como para que pueda ser usado, de una manera independiente, para describir cualquier interfaz de usuario e incluso para realizar tareas simples y comunes de manipulación de datos. (OPTIC, 2006)

Solo Opera soporta XForms nativamente, aunque existen varios complementos (plugins) y extensiones que les dan soporte a otros navegadores, como es el caso de Firefox el cual soporta XForms a través

de una extensión. XForms también puede ser usado a través de varias tecnologías de servidor que convierten el código de XForms a formularios de HTML en tiempo de ejecución y de manera transparente. En la actualidad existe una herramienta llamada AJAXForms, que transforma, en tiempo de compilación, documentos XHTML/XForms en páginas HTML con Javascript, que si entienden los navegadores actuales. Estas páginas gestionan, sin interactuar con el servidor, tanto la presentación como la lógica de la interfaz de usuario y su comunicación con el servidor se restringe al intercambio de datos utilizando técnicas AJAX. Debido a que está orientado principalmente a generar aplicaciones Web, se decidió no utilizarlo.

1.5.4 Orbeon Forms

Orbeon Forms es una solución de código abierto para crear aplicaciones web centrado en tecnologías XML, lo que significa que en vez de utilizar un lenguaje procedural o imperativo se utiliza XML como lenguaje declarativo. Es compatible con los navegadores web estándares (incluyendo Internet Explorer, Firefox, Safari y Opera), gracias a la tecnología XForms y Ajax, sin necesidad de software de el lado del cliente o de complementos (plugins). Permite crear formas totalmente interactivas, con características que incluyen validación, secciones opcionales y repetidas, resúmenes de errores actualizados, Formato de Documento Portátil (PDF del inglés Portable Document Format) de salida, la plena internacionalización, y los controles, como auto realización, pestañas, cuadros de diálogo, árboles y menús.

Orbeon Forms se basa en documentos XML y XForms. Esto lleva a una arquitectura ideal para las tareas de captura, procesamiento y presentación de datos XML (en particular, los datos del formulario), y no requiere ningún script Java, JavaScript u otro código de *scripting* en absoluto. Se articula en torno a un AJAX fácil de usar basado en un motor XForms, lo que lleva la norma W3C XForms a los navegadores estándares, y el motor XPL, un maduro motor XML *pipeline* de alto rendimiento para la transformación de datos XML.

Orbeon Forms cuenta con una buena documentación y con muchos ejemplos publicados en la web. Se puede utilizar Orbeon Forms por sus funcionalidades XForms, pero también se puede aprovechar más Orbeon Forms utilizándolo para construir una aplicación basada en formularios. (ORBEON, 2009)

Es realmente open source y libre, se puede utilizar para construir aplicaciones open source (código abierto) o comerciales. Este framework está orientado a gestionar interfaces de usuario web por lo que se eligió no utilizarlo, dado que la investigación se enfoca en encontrar algún sistema, framework o

herramienta que permita gestionar las interfaces de usuario pero a través de una aplicación de escritorio.

1.5.5 QtJambi

Es un framework de desarrollo en Java con base en las librerías Qt C++ multiplataforma. Posee un API (Application Programming Interface, en español Interfaz de Programación de Aplicaciones) muy intuitivo que provee las mismas funcionalidades del API de C++. Tiene librerías muy completas para el desarrollo de aplicaciones cliente o servidor. Puede mezclar componentes hechos en java y C++ en un solo proyecto brindando varios beneficios entre los que podemos mencionar el aumento de la eficiencia de desarrollo, la libertad y la flexibilidad.

QtJambi además posee un poderoso diseñador de interfaces el cual genera archivos XML que contienen la información de las interfaces como son sus componentes y los eventos relacionados con estos, que luego se convierten a código Java el cual es compilado para generar finalmente las interfaces. Posee una amplia documentación y una gran comunidad de desarrollo. Incluye todos los componentes propuestos por Swing e incorpora otros, además de poseer una librería de clases mayor.

Cuenta también con una serie de herramientas de gran utilidad como el QtGenerator que permite mapear clases C++ a Java, el QtLinguist para el trabajo con la internacionalización de las aplicaciones y el QtAssistant para la generación de la documentación correspondiente.(QTJAMBI, 2008)

A pesar de que se encontró en él muchas ventajas, se decidió no utilizarlo dado la negativa de que se distribuye bajo la licencia GPL, la cual obliga a que cualquier producto que sea desarrollado utilizando este framework, tendrá que ser distribuido bajo la misma licencia, o de lo contrario comprar una licencia comercial.

1.5.6 AWT/Swing

Java proporciona una serie de paquetes estándares que nos facilitan la labor de crear las interfaces gráficas de usuario entre los que podemos encontrar AWT (Abstract Window Toolkit) y Swing. AWT proporciona los componentes básicos de entrada/salida de datos. Esta biblioteca estaba concebida como una API estandarizada que permitía utilizar los componentes nativos de cada sistema operativo. Esto traía como consecuencia que el programador AWT estaba confinado a un mínimo denominador común entre ellos. Es decir que sólo se disponían de las funcionalidades comunes en todos los sistemas operativos. También sucedía que el comportamiento de los controles variaba mucho de sistema a sistema y era cada vez más difícil construir aplicaciones portables. AWT tiene asociado su

propio recurso de ventana y esto origina, dependencia de la plataforma, además las aplicaciones con muchos componentes, consumen muchos recursos. Entre sus potencialidades podemos mencionar que posee además un robusto modelo manejador de eventos y herramientas para gráficos las cuales incluyen clases para colores, formas y fuentes. También brinda facilidades para el diseño de ventanas más flexibles las cuales no dependen de la resolución de pantalla.

Swing aparece en la versión 2.0 de Java y recoge la mayor parte de la funcionalidad de AWT, aportando clases más complejas y especializadas, así como otras de nueva creación. Swing extiende el AWT añadiendo un conjunto de componentes, JComponents, y sus clases de soporte, los cuales a diferencia de los AWT, están escritos en Java, lo que determina independencia respecto de la plataforma, además, al no tener su propia ventana, consumen mucho menos recursos. Swing ofrece también la posibilidad de cambiar fácil y rápidamente el aspecto y sensación (*Look&Feel*) de un único componente o grupo de componentes. Esta posibilidad, que se conoce como aspecto y sensación conectables (Pluggable Look&Feel), es un sello distintivo de Swing y permite emular la apariencia de los componentes nativos manteniendo las ventajas de la independencia de la plataforma.

El origen de los controles GUI que presenta Swing lo encontramos en las Clases de Fundamentos de Internet de Netscape (IFC) aunque los componentes Swing van más allá de las IFC, hasta el punto de que no hay un parecido apreciable entre los componentes Swing y los de las IFC. Las clases Swing están implementadas utilizando una modificación del patrón clásico MVC de diseño de GUIs. La vista y el controlador se han combinado para formar clases IU. En la práctica, las aplicaciones Java con interfaces gráficas suelen combinar AWT y Swing, AWT se encarga de toda la gestión de eventos y Swing ofrece una serie de componentes más sofisticados. (GUTIÉRREZ, 2005)

AWT/Swing está orientado al desarrollo de aplicaciones de escritorio orientadas a la gestión de interfaces gráficas, para las cuales brinda sus mayores potencialidades por lo que se decidió utilizar este framework para el desarrollo de la aplicación. Mencionar además que es nativo del API de java y por tanto no es necesario comprar licencias para su uso, brinda un gran número de componentes y funcionalidades para estos, que facilitan el trabajo con las interfaces gráficas de usuario, cuenta con una documentación abundante y es de fácil uso.

1.6 Java como lenguaje de programación

El lenguaje de programación Java tiene la capacidad de funcionar sobre cualquier plataforma de software y hardware. Este es un lenguaje basado en las tecnologías de software libre, robusto y

además es orientado a objetos lo cual resuelve los problemas en la complejidad de los sistemas. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos mucho más simple. Entre sus principales características se encuentran:

Simple: Ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. Reduce en un 50% los errores más comunes de programación con lenguajes como C y C++, al eliminar muchas de las características de éstos, entre las que se encuentran la aritmética de punteros y la herencia múltiple.

Orientado a objetos: Trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo.

Distribuido: Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales. Proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que corran en varias máquinas, interactuando.

Multiplataforma: Los programas en Java pueden ejecutarse en cualquier plataforma, sin necesidad de hacer cambios. La compatibilidad es total:

- A nivel de fuentes: El lenguaje es exactamente el mismo en todas las plataformas.
- A nivel de bibliotecas: En todas las plataformas están presentes las mismas bibliotecas estándares.
- A nivel del código compilado: el código intermedio que genera el compilador es el mismo para todas las plataformas. Lo que cambia es el intérprete del código intermedio

Robusto: Java realiza verificaciones en busca de problemas, tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria.

Multihebrado: Java es un entorno de ejecución multihebrado, lo que significa que un mismo programa Java puede subdividirse en varias unidades de proceso concurrentes, es decir, que se ejecutan al mismo tiempo, en el mismo espacio de direcciones y compartiendo las mismas variables, lo que permite crear de forma sencilla programas que realicen más de una cosa a la vez, aunque exige un gran cambio de mentalidad a la hora de programar. (MOREA, 2009)

1.7 Plataforma J2EE

J2EE es una plataforma que habilita soluciones para desarrollo, uso efectivo y manejo de multicapas en aplicaciones centralizadas en el servidor. Define los estándares para desarrollar aplicaciones empresariales en Java, simplificando el desarrollo de este tipo de aplicaciones, basándolas en componentes modulares y estandarizados, y proporcionando un conjunto de especificaciones que aseguran la portabilidad de las aplicaciones entre un amplio número de productos comerciales y de códigos abiertos existentes, capaces de soportar J2EE. Esta plataforma cuenta con las siguientes características:

- Portable: Puede utilizarse en cualquier plataforma para la que haya disponible una Máquina Virtual Java (JVM).
- Escalable: Soporta el aumento de clientes, sin tener que reescribir todo el código de nuevo, tan solo añadiendo nuevos componentes J2EE a una aplicación.
- Altamente Soportada: Prácticamente cualquier gran empresa de software tiene un contenedor de componentes compatible con J2EE.
- Segura: El entorno de seguridad de la plataforma J2EE permite que se definan unas restricciones de seguridad en el momento de despliegue de la aplicación, aislando las aplicaciones de la complejidad de las implementaciones de seguridad, la plataforma J2EE hace portables una gran complejidad de implementaciones de seguridad. (CIBERAULA, 2006)

1.8 NetBeans

NetBeans se refiere a una plataforma para el desarrollo de aplicaciones de escritorio usando Java y a un entorno de desarrollo integrado (IDE) desarrollado usando la Plataforma NetBeans. Es muy útil a la hora de crear aplicaciones de tipo escritorio. Este brinda todas las funcionalidades necesarias para la creación de aplicaciones completas.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (*manifest file*) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio 2000 y continúa siendo el patrocinador principal de los proyectos. (NETBEANS, 2009)

1.9 Metodologías de desarrollo

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos de software. Indican, paso a paso, todas las actividades a realizar para lograr el producto informático deseado, además de las personas que deben participar en el desarrollo de las actividades y qué papel deben tener.

Además se encargan de elaborar estrategias de desarrollo de software que promueven prácticas centradas en las personas o los equipos, orientadas hacia la funcionalidad y la entrega, de comunicación intensiva y que requieren implicación directa del cliente.

1.9.1 RUP y UML

El Proceso Unificado Racional (RUP) es un proceso de desarrollo de Software: lo que significa el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. Sin embargo, RUP es más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organización, diferentes niveles de aptitud y diferentes tamaños de proyectos. RUP está basado en componentes, lo cual quiere decir que el sistema software en construcción está formado por componentes software interconectados a través de interfaces bien definidas.

La Metodología RUP se distingue por sus tres características fundamentales: dirigida por casos de usos, centrada en la arquitectura e iterativo incremental. Ella se divide en ciclos de desarrollo, donde al

final de cada ciclo se obtiene un producto. El ciclo de vida de RUP está compuesto por cuatro fases (inicio, elaboración, construcción y transición) e hitos que controlan el avance del proyecto al finalizar cada una de estas. En cada fase de desarrollo, se lleva a cabo un conjunto de flujos de trabajo para la realización de todo el proyecto, desde sus inicios hasta el final del desarrollo del software. Se basa en un proceso iterativo lo cual permite acercarse poco a poco a la solución sin entrar demasiado rápido en detalles. RUP genera también entregables basados en los artefactos después de cada fase, pero en su caso no se limitan solo al código, si no que los entregables viene acompañados de todo lo que traería el producto final, es decir, notas de la versión, instrucciones de instalación, ayuda de uso, entre otros.

El Proceso Unificado utiliza el Lenguaje Unificado de Modelado (UML), para preparar todos los esquemas de un sistema de software. De hecho UML es una parte esencial del Proceso Unificado y sus desarrollos fueron paralelos. UML nos ofrece un modo estándar de visualizar, especificar, construir, documentar y comunicar los artefactos de un sistema muy basado en el software. UML es un medio, no un fin, el objetivo final es una aplicación de software robusta, flexible y escalable. Es necesario tanto un proceso como un lenguaje para poder obtenerla.

UML permite la creación de los diferentes modelos que ofrecen las vistas necesarias para la construcción de un software de calidad y permite la compresión del sistema que se quiere realizar tanto por parte de los usuarios finales, como de los desarrolladores que implementarán la solución. Entre sus principales ventajas podemos mencionar que UML se puede usar para modelar distintos tipos de sistemas, ya sean sistemas de software o sistemas de hardware, y organizaciones del mundo real. Además UML es una consolidación de muchas de las notaciones y conceptos más usados orientados a objetos. (IVARJACOBSON 2000)

1.10 Herramientas utilizadas para el modelado de sistema

Las aplicaciones informáticas que facilitan el trabajo dentro del ciclo de desarrollo del software son conocidas como herramientas CASE (Ingeniería de Software Asistida por Computadora) y se emplean para aumentar la productividad del desarrollo del software disminuyendo los tiempos de construcción y el costo de los mismos. La tecnología CASE supone la automatización del desarrollo del software, contribuyendo a mejorar la calidad en el desarrollo de sistemas de información.

1.10.1 Visual Paradigm

Es una herramienta diseñada para desarrollar software que utiliza la programación orientada a objetos, es desarrollada como software libre y es compatible con las metodologías existentes para el desarrollo

de software. Busca reducir la duración de un ciclo de desarrollo brindando ayuda a arquitectos, analistas, diseñadores y desarrolladores. Presenta varios beneficios entre los que se destacan:

- Poderosa herramienta de generación de PDF/HTML a partir de diagramas UML.
- Sincronización entre el código fuente y el modelo en tiempo real.
- Soporte para toda la notación UML.
- Presenta un diseño basado en casos de uso y enfocado al negocio que genera un software de mayor calidad.
- Ofrece capacidades de ingeniería directa e inversa.
- Colaborativa ya que soporta múltiples usuarios trabajando sobre el mismo proyecto.

1.11 Conclusiones

A partir del análisis de SAIME, de las tendencias actuales de soluciones orientadas a servicios y tecnologías que se integran con la misma, se determinó la necesidad de una aplicación de escritorio que brinde alta seguridad y funcione de forma desacoplada a la gestión de los procesos de negocio.

Además a partir de los enfoques internacionales sobre el desarrollo del software libre se concluye abogando por la utilización de los mismos, fundamentados en las ventajas que brinda la plataforma J2EE de Java con el framework AWT/Swing, se propone también utilizar como metodología RUP para el proceso de desarrollo de software, utilizando para el modelado UML como lenguaje y Visual Paradigm como herramienta, lo que permitirá realizar de manera adecuada el correspondiente análisis y diseño para la futura implementación de la aplicación.

CAPÍTULO II. CARACTERÍSTICAS DEL SISTEMA

2.1 Introducción

El desarrollo de la sociedad ha conllevado al nacimiento de los documentos de identificación en respuesta a la necesidad de otorgar identidad personal a los ciudadanos. Los documentos de identificación son documentos emitidos por las autoridades administrativas competentes para permitir la identificación personal de sus ciudadanos. Existen sistemas que se encargan de automatizar estos procesos de emisión de documentos de identificación, de manera que la emisión de los mismos se haga de la mejor forma posible para evitar errores muy comunes como el duplicado de identidad y la falsificación de documentos.

Para el desarrollo de cualquier sistema informático, RUP propone entre sus flujos de trabajo el de *Requerimientos*, en él se establece qué debe hacer exactamente el sistema que se construye, además de llegar a un acuerdo entre clientes y otros interesados sobre lo que el sistema podría hacer, proveer a los desarrolladores un mejor entendimiento de los requisitos del sistema, definir el ámbito del sistema, entre otros. Los requisitos se dividen en dos grupos: los requisitos funcionales representan la funcionalidad del sistema, los requisitos no funcionales representan aquellos atributos que debe exhibir el sistema, pero que no son una funcionalidad específica.

2.2 Sistema de emisión de documentos de identificación

El Centro de Identificación y Seguridad Digital tuvo muy buenos resultados con SAIME, sistema para la gestión de los procesos de emisión de documentos de identificación, pero no obstante este presentaba algunos inconvenientes, uno de ellos es su desarrollo con tecnología propietaria que requiere altos costos por mantenimiento y pago de licencias. Otra de las desventajas de este sistema es que se implementó de forma rígida pues es necesario reimplementarlo para adaptarlo a los cambios en los procesos de negocio ya que las interfaces no están lo más desacopladas posibles.

A partir de esto, el Centro de Identificación y Seguridad Digital, se propuso el desarrollo de un sistema para la emisión de documentos de identificación que siguiera la línea de software libre, que fuese lo más genérico posible, capaz de adaptarse rápidamente a los constantes cambios que ocurren en los procesos de negocio y que se acoplara a todas las tecnologías novedosas que han venido surgiendo en los últimos tiempos. Es a raíz de esto, que se realizaron una serie de investigaciones que dieron como resultado, la propuesta de un sistema basado en una arquitectura orientada a servicios, que

utilizara un Bus de Servicios Empresariales, también conocido por sus siglas en ingles ESB (Enterprise Service Bus), para modelar los procesos de emisión de documentos de identificación y que al mismo tiempo orquestara los servicios que tributan a estos procesos, y una *aplicación de gestión de interfaces de usuario*, que funcionara de manera independiente al resto del sistema, que fuera capaz de gestionar las interfaces de usuario que responden a estos procesos, para de esta manera garantizar una separación total de las interfaces de toda lógica de negocio del sistema.

Entre las principales ventajas que se esperan con el desarrollo de este sistema, podemos mencionar que podrá adaptarse fácilmente a las características y necesidades de cada cliente en particular, que estará basado en tecnología software libre y que permitirá adaptarse a los constantes cambios que puedan ocurrir en los procesos de emisión de documentos de identificación, de una manera rápida y eficaz reduciendo al máximo los costos.

2.3 Procesos de Emisión de Documentos de Identificación

El Sistema de Emisión de Documentos de Identificación (SEDI) es un sistema en creación que se adaptará fácilmente a cualquier negocio ya que los procesos de emisión de documentos de identificación son estándares en los diferentes negocios.

Existen cuatro procesos que aportan un valor determinado en cualquier sistema de este tipo, estos son ***Solicitud de documentos de identificación***, ***Enrolamiento de datos***, ***Personalización de documentos de identificación*** y ***Entrega de documentos de identificación***.

Un modelo por lo general es la mejor forma para lograr un entendimiento de estos procesos, para modelar los procesos anteriormente mencionados se utilizó BPMN (Business Process Management Notation) que permite modelar los procesos de una manera unificada y estandarizada. BPMN define la notación y semántica de un Diagrama de Procesos de Negocio (Business Process Diagram, BPD).

2.3.1 Descripción del proceso Solicitud de Documentos de Identificación

El ciudadano solicita vía web el trámite del documento de identificación, la entidad encargada de tramitar este documento al recibir la solicitud verifica que los datos enviados en la misma estén en formato correcto, si no son válidos envía una notificación de denegación de trámite al ciudadano, de ser válidos envía notificación de aceptación de trámite al ciudadano especificando la fecha de cita para su captura de información y el número de serie que identifica el trámite de la solicitud del documento. **Ver diagrama del proceso en el Anexo 1 Fig.1**.

2.3.2 Descripción del proceso Enrolamiento de Datos

El proceso se inicia al presentarse el ciudadano solicitante del documento de identificación en el área de captura de información dentro de la entidad trámite del documento de identificación. Primeramente se procede a capturar los datos personales requeridos para la realización del trámite, luego se realizará la revisión detallada de estos elementos que incluye la comprobación de la legalidad de la documentación presentada para identificar documentación fraudulenta.

Si en la revisión, los datos capturados no son adecuados se concluye el trámite del documento de identificación. Si los datos capturados son adecuados se realiza la captura de información biométrica al ciudadano, si la captura no fue realizada correctamente se captura la información biométrica nuevamente. Una vez que la captura de información biométrica se realiza correctamente se procede a validar entonces si la identificación aportada por el ciudadano es válida a través del subproceso *Validación de Identidad*. Cuando el resultado de la validación de información del ciudadano obtenido en el subproceso anterior no es válido se almacena dicha información de trámite con el objetivo de hacer un análisis posterior de la misma y decidir la terminación o no del trámite correspondiente. Por el contrario si la validación de la información del ciudadano es válida se prepara la información del trámite para su futura personalización y se actualiza el estado del trámite como indicador que permitirá mostrar en todo momento el estado en que se encuentra el trámite del documento de identificación. **Ver diagrama del proceso en el Anexo 1 Fig. 2**.

2.3.3 Descripción del subproceso Validación de Identidad

La entidad encargada del trámite luego de capturar toda la información del ciudadano la envía a un sistema externo el cual se encargará de verificar la validez de dicha información y emitir un resultado a la correspondiente entidad encargada del trámite del documento de identificación. **Ver diagrama del proceso en el Anexo 1 Fig. 3**.

2.3.4 Descripción del proceso Personalización del Documento de Identificación

El proceso inicia con la impresión de la información obtenida del ciudadano, luego se realiza un control de la calidad sobre el documento de identificación, si el documento es defectuoso se desecha y se procede a iniciar el proceso nuevamente, si es correcto se procede al ensobrado para su futura entrega y se actualiza el estado del trámite. **Ver diagrama del proceso en el Anexo 1 Fig. 4**.

2.3.5 Descripción del proceso Entrega Documento de Identificación

Se procede a buscar en el sistema el estado del trámite para verificar la finalización del documento de identificación, si el documento de identificación está listo para su entrega se informa al ciudadano solicitante vía mensajería electrónica para que se presente en la entidad trámite del documento de identificación a recogerlo. El mismo se presenta a la entidad encargada del trámite del documento de identificación y se le solicita la presentación de su identidad, tras verificar la validez de la misma se le hace entrega del documento de identificación. **Ver diagrama del proceso en el Anexo 1 Fig.5**. (CASTRO, 2009)

2.4 Propuesta de Sistema

Para dar solución a los objetivos expuestos se propone el desarrollo de una aplicación de escritorio para la gestión de interfaces de usuario, implementada en Java utilizando la librería Swing para el trabajo con interfaces visuales, capaz de integrarse al SEDI y que separe las interfaces de toda lógica de los procesos de negocio y se adapte a los cambios de actividades en estos procesos sin necesidad hacer cambios en el código.

La aplicación estará formada por cinco piezas fundamentales: *MainApp*, *AppUpdater*, *Communication*, *UILoader*, *UIManager*, además de un grupo de componentes visuales especializados en la captura y validación de datos.

La comunicación entre la aplicación que se quiere desarrollar y la lógica de negocio del SEDI, se realizará a través de servicios web. Toda la información necesaria para gestionar rápidamente las interfaces de usuario la obtendríamos de un grupo de *servicios web* facilitados por un proveedor de servicios del SEDI, de manera tal que cuando el flujo en los procesos de negocio indique que se debe mostrar una interfaz determinada, la aplicación sea capaz de acaparar esa solicitud, buscando la interfaz requerida en un repositorio de interfaces y mostrándola.

La aplicación constará con un núcleo principal *MainApp* que sería el motor de esta aplicación, que maneje varios componentes, uno para la comunicación entre la aplicación y el proveedor de servicios, otro para administrar las interfaces de usuario, y otro para la actualización del repositorio de interfaces.

Uno de los componentes será el *Communication* que se encargará de la comunicación con el proveedor de servicios, este será el único de todos los componentes que se debe adaptar a un negocio determinado, requiriendo una implementación que se adecue al mismo. Sería el punto de entrada y

salida de la aplicación, el cual tendría como función principal convertir la información obtenida del proveedor de servicios a XML, que es el estándar de comunicación que se pretende implementar, y de realizar el proceso inverso de convertir el XML facilitado por la aplicación al mismo formato que tengan los datos que fueron suministrados por el proveedor de servicios.

A partir de la información que sea obtenida a través del componente para la comunicación, el núcleo de la aplicación, a través de un administrador de interfaces de usuario *Manager,* se encargará de gestionar dicha información y de mostrarle al usuario las interfaces correspondientes, para que este componente logre tal propósito deberá apoyarse en otro componente, el *Loader*, que sería el que cumpliría con el propósito de cargar estas interfaces. Además se pretende que durante este proceso de cargar las interfaces, en caso de que no se encuentren en el repositorio local de interfaces, el componente *Updater* permita actualizar dicho repositorio con la interfaz requerida, descargándola de un repositorio global.

De manera general la aplicación debe brindar la posibilidad de cargar interfaces, entre las que se encuentra la plantilla principal donde se cargan los formularios y los menús de navegación disponibles para el usuario. Otra funcionalidad que debe permitir la aplicación es la de gestionar el proceso de captura y validación de los datos introducidos por el usuario en los formularios. Además la aplicación deberá cargar los estilos deseados por el cliente.

2.5 Flujo de funcionamiento de la propuesta de solución

A partir de que se le muestre al usuario la plantilla, con una serie de opciones en un menú, él comenzará a interactuar con la misma. Al seleccionar una opción cualquiera, se genera una petición que debe ser procesada por la aplicación para luego mostrar el formulario correspondiente a la opción seleccionada, por ejemplo si el usuario escoge la opción de realizar una solicitud para la realización de un pasaporte a un cliente determinado, la aplicación deberá gestionar la interfaz correspondiente, que sería un formulario para que entre los datos personales de este cliente. El flujo de las interfaces que se deben gestionar de acuerdo a la opción seleccionada por el usuario, se realizará en el proveedor de servicios, el cual notificará en tiempo real a la aplicación sobre cual interfaz debe ser gestionada en cada momento.

No solo se deberán gestionar las interfaces de usuario que responden a peticiones de los usuarios sobre el menú de la plantilla, sino que esto también se deberá cumplir con la interacción del usuario sobre los formularios que se carguen en la plantilla, por ejemplo en el mismo caso anterior de la

solicitud de pasaporte para un cliente, después de que el usuario haya entrado los datos correspondientes a esta solicitud, procede a aceptar la misma, esta acción genera una petición, que se resuelve en mostrarle un próximo formulario donde se muestra una planilla de control que debe ser impresa para proceder a imprimir el pasaporte solicitado, de esta manera la aplicación deberá enviar esta solicitud al SEDI, a través del proveedor de servicios para que este interprete la petición y envíe la respuesta correspondiente, que sería informar a la aplicación sobre el próximo formulario que se le debe mostrar al usuario con la información correspondiente. De esta forma la aplicación debe gestionar las interfaces de usuario requeridas para los procesos de emisión de documentos de identificación, desacoplando totalmente las interfaces del flujo de actividades de los procesos de negocio.

2.6 Seguridad en servicios web

Debido a que la transmisión de información a través de servicios web (WS) está basada en protocolos HTTP, esta es vulnerable a las amenazas y vulnerabilidades típicas de este tipo de protocolo. Para garantizar la seguridad de la información, los WS se basan en varios conceptos:

- Identificación y autenticación: Verificación de la identidad de un usuario, proceso, o dispositivo, como una condición previa para permitir el acceso a los recursos en un sistema de información.
- Autorización: El permiso para utilizar un equipo de recursos, concedida, directa o indirectamente, por un aplicación o un sistema propietario.
- Integridad: La propiedad de que los datos no han sido modificado de manera no autorizada, mientras estaban en almacenamiento, durante el procesamiento, o en la transmisión de los mismos.
- No repudio: Garantía de que el remitente de la información proporcione su identidad como prueba de la entrega y de que el receptor disponga de la identidad del remitente, para que más tarde, no se puede negar el acceso a la información.
- Confidencialidad: Preservación de las restricciones de autorización de acceso y divulgación de la información, que incluye los medios para la protección de la intimidad personal y la propiedad de la información.

Para garantizar la integridad y confidencialidad de los mensajes transmitidos hacia y desde un WS se utiliza el protocolo Secure Sockets Layer (SSL) el cual proporciona sus servicios de seguridad cifrando los datos con un algoritmo de cifrado simétrico, y cifrando la clave de sesión mediante un algoritmo de cifrado de clave pública. La clave de sesión es la que se utiliza para cifrar los datos que vienen y van al servidor seguro. Para cada transacción, se genera una clave de sesión distinta, lo cual permite que

aunque sea interferida por un atacante en una transacción dada, no sirva para descifrar futuras transacciones.

2.7 Modelo de Dominio

Una de las primeras actividades centrales de un ciclo de desarrollo consiste en crear un modelo conceptual para los casos de uso, en el cual se explican a sus creadores los conceptos significativos en un dominio del problema. (SCHEMULLER, 2000)

El Modelo de Dominio también conocido como Modelo Conceptual es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Este representa clases conceptuales del dominio del problema, conceptos del mundo real, no de los componentes de software. Una clase conceptual también conocida como entidad puede ser una idea o un objeto físico (símbolo, definición y extensión).

El modelo de dominio se representa en UML con un diagrama de clases en el que se muestra: conceptos u objetos del dominio del problema: clases conceptuales asociaciones entre las clases conceptuales. (RÍOS, 2007)

2.7.1 Justificación de la utilización del Modelo de Dominio

Teniendo en cuenta que la definición de procesos y roles del negocio se hace difícil encontrarlos, por lo que se ve a simple vista la necesidad de describir el funcionamiento de la aplicación mediante una serie de conceptos, entidades y sus relaciones, agrupándose en un modelo de dominio con el fin del fácil entendimiento de la aplicación.

2.7.2 Conceptos Asociados al Dominio

Un modelo de dominio captura los tipos más importantes de objetos en el contexto del sistema. Los objetos de dominio representan los principales conceptos. La modelación del dominio tiene como objetivo fundamental la comprensión y descripción de las clases más importantes en el sistema.

Concepto	Descripción
Usuario	Persona que interactúa con el sistema, en este caso con las interfaces gestionadas por la aplicación.
Ejecutable	Se refiere al fichero que es ejecutado por el

	usuario para inicializar la aplicación, su ejecución provoca que se le muestre al usuario la interfaz principal.
Aplicación de Gestión de IU	Es el sistema de gestión de interfaces de usuario a desarrollar. Es el que gestiona la interfaces.
Proveedor de Servicios	Sistema externo que maneja los procesos de emisión de documentos de identificación y se comunica con la aplicación de gestión de IU.
WebService	Se refiere a una serie de servicios web que nos brinda el proveedor de servicios, estos servicios contienen la información que necesita el componente para gestionar las interfaces de usuario.
Plantilla	Es la interfaz principal que se le muestra al usuario, el marco de trabajo, que contiene un conjunto de opciones que el usuario puede seleccionar.
Formulario	Es la interfaz que se carga dentro de la plantilla.
Menú	Conjunto de opciones que se brindan al usuario. Este se encuentra contenido en la plantilla.
Repositorio Interfaz	Contiene un conjunto de interfaces de usuario, ya sean plantillas o formularios que se encuentran disponibles en una localización determinada.

Tabla 2.1 Conceptos asociados al modelo del dominio

2.7.3 Modelo de Dominio

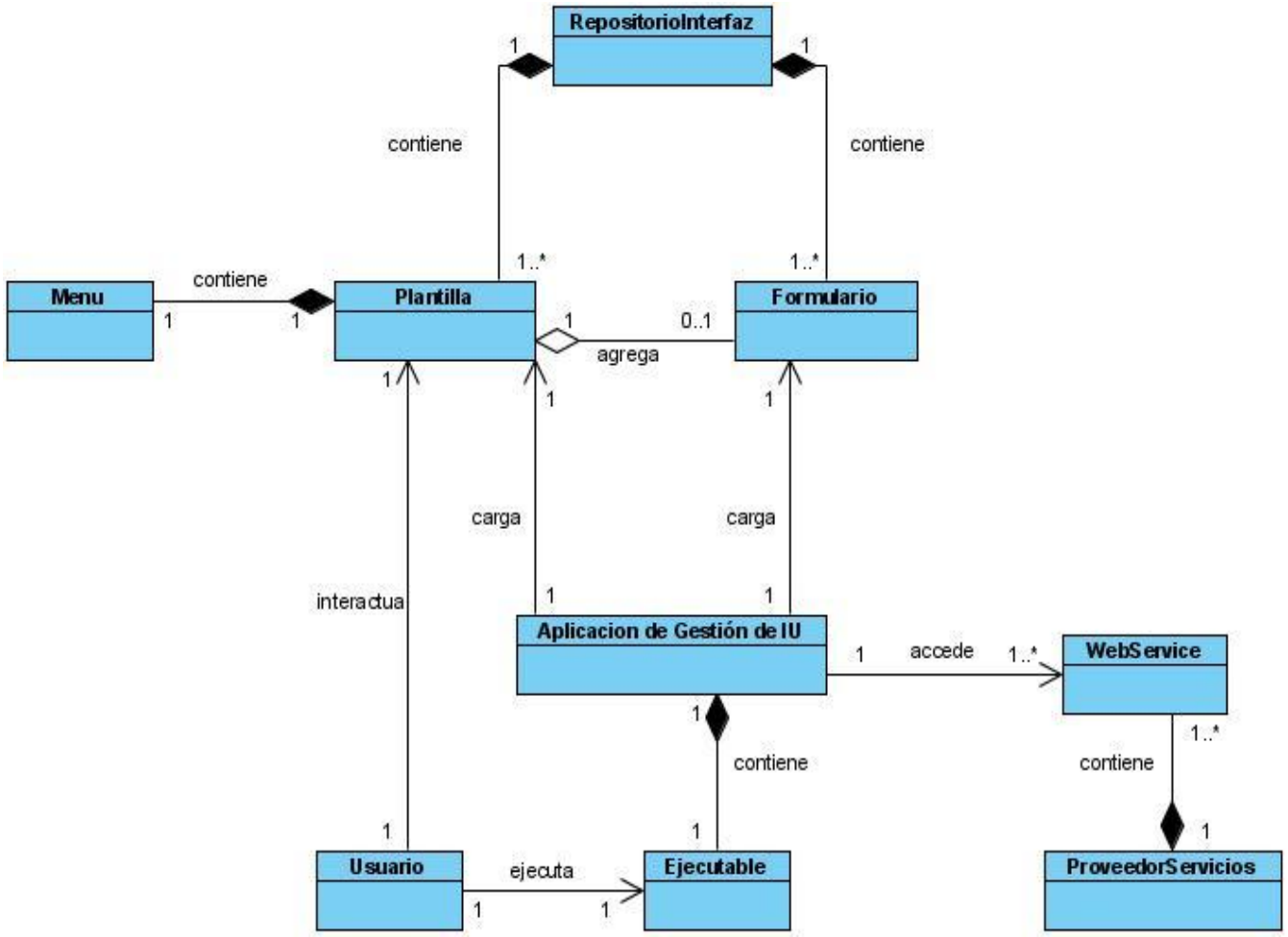


Figura 1 Modelo del Dominio.

2.7.4 Descripción del Modelo de Dominio

Cuando el usuario ejecuta la aplicación de gestión de IU, a partir de ese momento esta aplicación accede a los servicios web que ofrece un proveedor de servicios del SEDI y obtiene la información necesaria para su funcionamiento. Esta aplicación se encarga de cargar una plantilla principal, en la que le mostraría al usuario un menú con las opciones a las que este puede acceder. A partir de la interacción del usuario con estas opciones la aplicación procede a cargar el formulario correspondiente a la opción seleccionada por el usuario en el menú. Tanto la plantilla como el formulario se encuentran almacenados en un repositorio de interfaces, al cual la aplicación accede cada vez que requiera cargar una plantilla o un formulario cualquiera.

2.8 Especificación de los Requisitos del Software

Hacer una identificación exhaustiva y correcta de los requisitos fundamentales que debe cumplir el sistema es muy importante. Un proyecto no puede ser exitoso sin una especificación correcta y exhaustiva de los requerimientos.(LARMAN, 1999) Una vez que se han definido los conceptos principales relacionados con el objeto de estudio y el dominio, se puede comenzar a analizar qué debe hacer la aplicación para que cumpla con los objetivos planteados al inicio de este trabajo. Para ello, se enumeran a través de requerimientos funcionales y no funcionales, las acciones que el sistema deberá ser capaz de realizar.

2.8.1 Requerimientos Funcionales

La IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) define un requerimiento como la condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo. El propósito fundamental de la captura de los requisitos es guiar el desarrollo hacia el sistema correcto. Esto se consigue mediante una descripción de los requisitos del sistema suficientemente buena como para que pueda llegarse a un acuerdo entre el cliente (incluyendo a los usuarios) y los desarrolladores sobre qué debe y qué no debe hacer el sistema.(JAMES RUMBAUGH, 2000)

Requisitos funcionales de la aplicación de gestión de interfaces de usuario

RF 1 Cargar archivo de configuración.
- RF 1.1 Obtener el identificador de la plantilla a cargar.
- RF 1.2 Obtener la dirección del repositorio local.
- RF 1.3 Obtener la dirección del repositorio externo.
- RF 1.4 Obtener la dirección del archivo de configuración de estilos.
- RF 1.5 Obtener la dirección del XML con los datos del menú.

RF 2 Cargar plantilla.
- RF 2.1 Buscar plantilla en el repositorio local.
- RF 2.2 Buscar archivo de configuración de estilos (Look & Feel).
- RF 2.3 Aplicar a la plantilla estilo obtenido.
- RF 2.4 Mostrar plantilla principal.

RF 3 Cargar menú.
- RF 3.1 Obtener XML con los datos del menú.
- RF 3.2 Obtener datos del menú.
- RF 3.3 Adicionar datos al menú.

RF 4 Actualizar menú.

RF 4.1 Obtener del proveedor de servicios los datos a mostrar en el menú.

RF 4.2 Convertir esta información referente al menú a formato XML.

RF 4.3 Obtener del XML generado los datos del menú.

RF 4.4 Actualizar el menú de la plantilla con los datos obtenidos.

RF 5 Cargar formulario.

RF 5.1 Obtener del proveedor de servicios la información del formulario a cargar.

- Identificador del formulario.
- Listado de componentes del formulario.
- Listado de las propiedades de los componentes del formulario.
- Listado de valores de los componentes del formulario.
- Acciones de los componentes del formulario.

RF 5.2 Convertir la información referente al formulario a cargar a formato XML.

RF 5.3 Obtener del XML generado los datos del formulario a cargar.

RF 5.4 Buscar el formulario en un repositorio local.

RF 5.5 Adicionar formulario a la plantilla principal.

RF 5.6 Mostrar el formulario en la plantilla.

RF 6 Buscar interfaz en el repositorio externo.

RF 6.1 Descargar la interfaz al repositorio local.

RF 7 Enlazar los datos del XML con los componentes del formulario cargado.

RF 7.1 Propiedades de los componentes.

RF 7.2 Acciones de los componentes.

RF 8 Capturar datos del formulario.

RF 8.1 Datos entrados por el usuario en el formulario.

RF 8.2 Acciones generadas por el usuario en el formulario.

RF 9 Validar si los datos capturados en la interfaz de usuario son correctos.

RF 10 Enviar los datos capturados en el formulario al proveedor de servicios.

RF 10.1 Convertir los datos capturados del formulario a XML.

RF 10.2 Convertir los datos de este XML al mismo formato con que fueron suministrados los datos por el Proveedor de Servicios.

2.8.2 Requerimientos no Funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto usable, rápido o confiable. Aunque los mismos no aportan funcionalidades propiamente dichas dentro de la aplicación, son de vital importancia para una puesta en marcha exitosa del software, y para lograr que este responda a las expectativas del usuario.

Requerimientos de software

- La aplicación utilizará la maquina virtual de java JDK 1.6 o superior.

Requerimientos de Hardware

- 256 MB de RAM o superior.
- Procesador Pentium IV a 2.81 GHz o superior.

Restricciones de diseño e implementación

- La aplicación será implementada en la plataforma J2EE.
- EL lenguaje de programación será java.
- Para el análisis y el diseño de la aplicación debe ser utilizada la metodología RUP, usando el lenguaje de modelación UML y como herramienta para llevarlo a cabo el Visual Paradigm.

Usabilidad

- La aplicación podrá ser utilizada por usuarios con conocimientos de los procesos de emisión de documentos de identificación.

Seguridad

- La información manejada por el sistema estará protegida del acceso y divulgación no autorizado.
- La información estará disponible en todo momento para los usuarios que tenga acceso a ella.

Requerimientos de apariencia o interfaz externa

- La plantilla contendrá un menú a la izquierda con las opciones disponibles para el usuario autenticado, y a la derecha el área de contenido donde se cargarán los formularios.

2.9 Modelo de Casos de Uso del Sistema

El modelo de casos de uso es un modelo del sistema que contiene actores, casos de uso y sus relaciones. El actor es una entidad externa del sistema que de alguna manera participa en la historia del caso de uso. Por lo regular estimula el sistema con eventos de entrada o recibe algo de él.

Definición de los actores del sistema

Actores del Sistema	Justificación
Usuario	Persona que interactúa con la aplicación
Proveedor de Servicios	Sistema que con el que interactúa la aplicación de gestión de IU.

Tabla 2.2 Definición de actores

2.9.1 Casos de uso del sistema

Un caso de uso describe un proceso que deberá convertirse en una funcionalidad. Un proceso describe, de comienzo a fin, una secuencia de eventos, de las acciones y las transacciones que se requieren para producir u obtener algo de valor para una empresa o actor.

CU1	Cargar Plantilla
Actor	Usuario
Descripción	El caso de uso se inicia cuando el usuario ejecuta la aplicación, la cual carga las configuraciones necesarias para mostrar la plantilla.
Referencia	R1,R2,R3,R6

Tabla 2.3 Descripción resumida del caso de uso Cargar Plantilla.

CU2	**Cargar Formulario**
Actor	Usuario, Proveedor de Servicios
Descripción	El caso de uso se inicia cuando el usuario selecciona una opción en el menú o presiona un botón de un formulario ya cargado, la aplicación busca el formulario correspondiente a dicha opción y lo muestra.
Referencia	R1,R4,R5,R6,R7

Tabla 2.4 Descripción resumida del caso de uso Cargar Formulario.

CU3	**Capturar Datos**
Actor	Usuario, Proveedor de Servicios
Descripción	El caso de uso se inicia cuando el usuario genera un evento sobre el menú o el formulario, la aplicación captura este evento y ejecuta las acciones pertinentes.
Referencia	R8,R9,R10

Tabla 2.5 Descripción resumida del caso de uso Capturar Datos.

2.9.2 Diagrama de Casos de Uso del Sistema

Los casos de uso son el componente clave del modelado. Su propósito es ilustrar como un sistema permite a un actor cumplir una meta, ilustrando todos los posibles caminos apropiados que ellos pueden tomar para cumplirla, así como las situaciones que podrían hacerlo fallar. Un diagrama de casos de uso del sistema representa gráficamente las funcionalidades principales del sistema y su interacción con los actores.

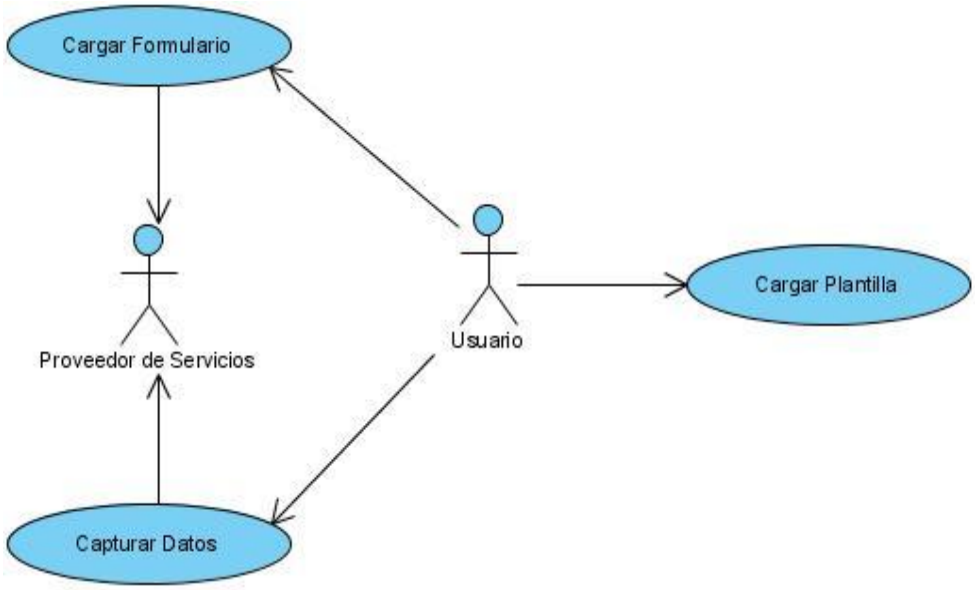


Figura 2 Diagrama de Casos de Uso del Sistema.

2.9.3 Descripciones de Caso de Uso

Caso de uso		
CU1	Cargar Plantilla.	
Propósito	Cargar la plantilla de la aplicación.	
Actores: Usuario(inicia)		
Resumen: El caso de uso se inicia cuando el usuario ejecuta la aplicación, la cual carga las configuraciones necesarias para mostrar la plantilla.		
Referencias	R1,R2,R3,R6	
Acción del actor		Respuesta del sistema
1. El usuario ejecuta la aplicación.		La aplicación busca en un fichero de configuración lo siguiente: • Identificador de la plantilla a cargar. • Dirección del repositorio local. • Dirección del repositorio externo. • Dirección del archivo de configuración de estilos (Look & Feel). • Dirección del XML con los datos del menú.
		3. La aplicación busca el archivo de configuración de estilos “Look & Feel”.
		4. La aplicación carga la configuración de estilos.
		5. La aplicación busca la plantilla con el identificador correspondiente en el repositorio local.
		6. La aplicación obtiene los datos del menú.
		7. La aplicación carga los datos en el menú.
		8. La aplicación muestra al usuario la plantilla.
Flujo alternativo		
Acción del actor		Respuesta del sistema
		5.1 Si no encuentra la plantilla con el identificador correspondiente en el repositorio local, la aplicación la busca en un repositorio externo.
		5.2La aplicación descarga la plantilla correspondiente al repositorio local. Ir al paso 6

Tabla 2.6 Descripción extendida del caso de uso Cargar Plantilla.

<table>
<tr><th colspan="3">Caso de uso</th></tr>
<tr><td>CU2</td><td colspan="2">Cargar Formulario.</td></tr>
<tr><td>Propósito</td><td colspan="2">Cargar un formulario dentro de la plantilla.</td></tr>
<tr><td colspan="3">Actores: Usuario(inicia), Proveedor de Servicios</td></tr>
<tr><td colspan="3">Resumen: El caso de uso se inicia cuando el usuario selecciona una opción en el menú o presiona un botón de un formulario ya cargado, la aplicación busca el formulario correspondiente a dicha opción y lo muestra.</td></tr>
<tr><td>Referencias</td><td colspan="2">R1,R4,R5,R6,R7</td></tr>
<tr><td colspan="2">Acción del actor</td><td>Respuesta del sistema</td></tr>
<tr><td colspan="2">1. El usuario selecciona una opción del menú o presiona un botón de un formulario ya cargado, para que se le muestre la interfaz correspondiente a esa opción.</td><td>La aplicación busca en un fichero de configuración lo siguiente:
Dirección del repositorio local.
Dirección del repositorio externo.</td></tr>
<tr><td colspan="2"></td><td>3. La aplicación accede a los servicios web que ofrece el Proveedor de Servicios.</td></tr>
<tr><td colspan="2">El proveedor de servicios le brinda la siguiente información:
Identificador del formulario a cargar.
Listado de componente del formulario a cargar.
Listado de propiedades de los componentes del formulario a cargar.
Listado de valores de los componentes del formulario.
Acciones de los componentes del formulario a cargar.
Datos del menú.</td><td></td></tr>
<tr><td colspan="2"></td><td>5. La aplicación obtiene la información brindada por el Proveedor de Servicios.</td></tr>
<tr><td colspan="2"></td><td>6. La aplicación convierte la información obtenida del proveedor de servicios a formato XML.</td></tr>
<tr><td colspan="2"></td><td>7. Si la aplicación obtiene datos del menú ir al paso 8. Si la aplicación no obtiene datos para el menú ir al paso 9.</td></tr>
<tr><td colspan="2"></td><td>8. La aplicación carga el menú con los datos obtenidos.</td></tr>
<tr><td colspan="2"></td><td>9. La aplicación busca el formulario con el identificador</td></tr>
</table>

	correspondiente en el repositorio local.
	10. La aplicación enlaza los componentes del formulario a cargar con sus propiedades y acciones que se obtienen del XML.
	11. La aplicación muestra al usuario el formulario correspondiente dentro de la plantilla.
Flujo alternativo	
Acción del actor	Respuesta del sistema
	9.1 Si no encuentra el formulario con el identificador correspondiente en el repositorio local, la busca en un repositorio externo.
	9.2 Descarga el formulario correspondiente al repositorio local. Ir al paso 10.

Tabla 2.7 Descripción extendida del caso de uso Cargar Formulario.

Caso de uso	
CU3	Capturar Datos
Propósito	Capturar los datos de la plantilla
Actores: Usuario(inicia), Proveedor de Servicios	
Resumen: El caso de uso se inicia cuando el usuario genera un evento sobre el menú o el formulario, la aplicación captura este evento y ejecuta las acciones pertinentes.	
Referencias	R8,R9,R10
Acción del actor	Respuesta del sistema
1. El usuario genera un evento sobre el menú o el formulario.	2. La aplicación captura el evento que se genera de la interacción del usuario con el menú o el formulario.
	3. La aplicación asigna la acción de dicho evento a la acción de la plantilla.
	4. Si el evento proviene del formulario ir al paso 5. Si el evento proviene del menú ir al paso 8.
	5. La aplicación captura los datos de los componentes del formulario.
	6. La aplicación valida los datos capturados.
	7. La aplicación crea un archivo XML, que contiene dichos datos.

	8. La aplicación ejecuta la acción de la plantilla enviando los datos al Proveedor de Servicios.
8. El proveedor de servicios recibe los datos enviados de la aplicación.	
Flujo alternativo	
Acción del actor	Respuesta del Sistema
	6.1 Si los datos no son válidos el sistema muestra un mensaje de error.

Tabla 2.8 Descripción extendida del caso de uso Capturar Datos.

2.10 Conclusiones

La aplicación de gestión de IU formará parte del SEDI, su propósito fundamental será el de gestionar las interfaces de usuario referentes a los procesos de emisión de documentos de identificación. Deberá estar desacoplada de la lógica de negocio del SEDI y se encargará de gestionar la lógica de presentación del mismo. Las principales funcionalidades que debe cumplir la aplicación son: brindar la posibilidad de cargar plantillas, a partir de sus partes principales, menús, formularios; gestionar el proceso de captura y validación de los datos de los distintos formularios y garantizar la conexión con un proveedor de servicios a través de servicios web.

En este capítulo se comenzó a desarrollar la propuesta de solución, determinando un listado con las propiedades que debe cumplir el sistema y las funcionalidades que debe realizar, representando las últimas mediante un diagrama de casos de uso del sistema y finalmente describiendo paso a paso todas las acciones del actor con los casos de uso que interactúa. Ahora se puede comenzar a construir el sistema, cumpliendo con todos los requerimientos y las funciones que se han considerado necesarias en este capítulo.

CAPÍTULO III. ANÁLISIS Y DISEÑO

3.1 Introducción

El modelo de análisis especifica el comportamiento funcional del sistema, independientemente del ambiente en el que va a ser finalmente implementado. Para ello tiene en cuenta completamente y con exactitud los requerimientos del sistema. Por otra parte el diseño es decisivo, en el comienzo de las iteraciones de la fase de construcción, contribuye a una arquitectura estable, y a crear un plano del modelo de implementación. En el diseño se modela el sistema y se encuentra su forma para que soporte todos los requisitos incluyendo los requisitos no funcionales y otras restricciones que le suponen.

Para el desarrollo de un software es necesario contar con descripciones detalladas y saber cómo se satisfacen los requerimientos y las restricciones. En este capítulo se describen los elementos más importantes correspondientes a la etapa de análisis y diseño del sistema, para eso se definen los diagramas de interacción (secuencia) correspondientes a cada caso de uso y los diagramas de clases de análisis y diseño del sistema, además se realizará la descripción de la arquitectura y patrones de diseño utilizados en el diseño de la aplicación.

3.2 Modelado de Análisis

El análisis se basa en un modelo de objetos conceptuales, que se denomina modelo de análisis. Este modelo estructura los requisitos de un modo que facilita su comprensión, su modificación, en general, su mantenimiento. Se describe utilizando el lenguaje de los desarrolladores, por tanto puede introducir un mayor formalismo y ser utilizado para razonar sobre los funcionamientos internos del sistema. (LARMAN, 1999)

Durante el análisis se analizan los requisitos que se obtuvieron en la captura de requerimientos, refinándolos y estructurándolos. El objetivo de hacerlo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que ayude a estructurar el sistema entero, incluyendo su arquitectura. Siempre hay que tener presente que es más fácil y menos costoso hacer cambios o arreglar defectos en el análisis que en las fases siguientes.

3.2.1 Realizaciones de Casos de Uso

Una realización de caso de uso del análisis es la que describe cómo se lleva a cabo y se ejecuta un caso de uso determinado en términos de clases del análisis y de sus objetos del análisis en

interacción. Por tanto, proporciona una traza directa hacia un caso de uso concreto del modelo de casos de uso.

3.2.2 Diagrama de clases del análisis

El diagrama de clases del análisis representa las clases del análisis que participan en las realizaciones de los casos de uso y las relaciones que se establecen entre ellas. Un diagrama de clases del análisis representa las cosas del mundo real y no de la implementación, se define como aquel artefacto en el que se figuran los conceptos en un dominio del problema. Una clase de análisis representa una abstracción de una o varias clases y/o subsistemas del diseño del sistema. Esta abstracción se centra en el tratamiento de los requisitos funcionales.

Diagramas de clases del análisis por casos de uso

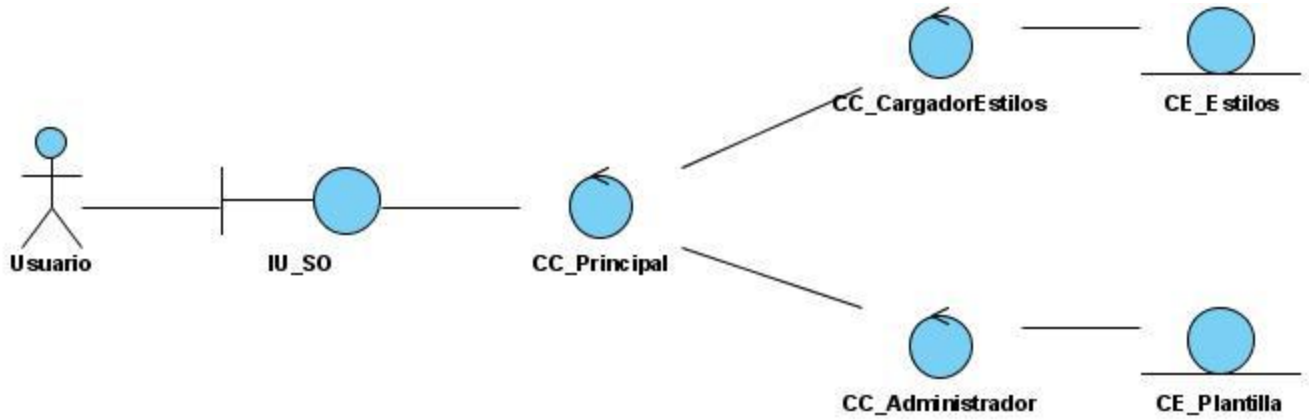


Figura 3 Diagrama de clases del análisis del CU Cargar Plantilla.

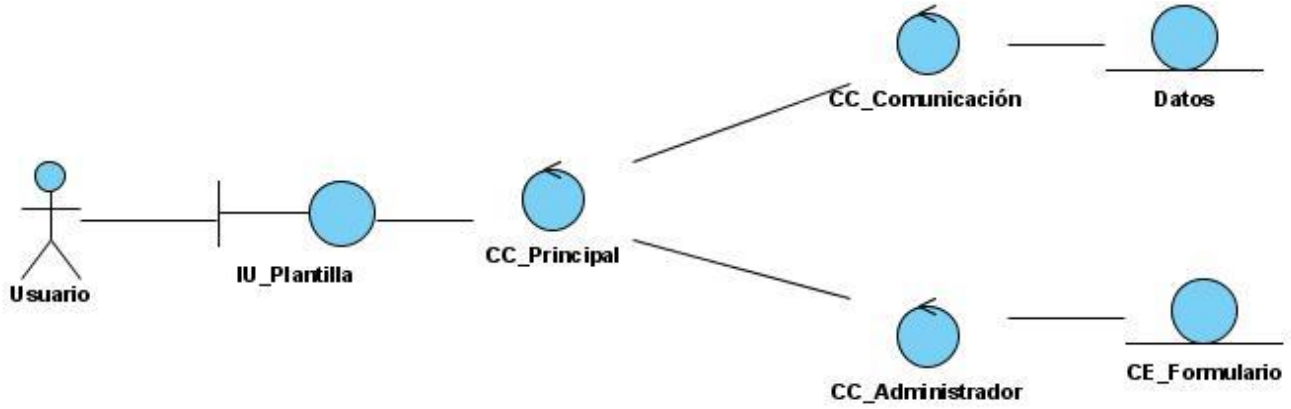


Figura 4 Diagrama de clases del análisis del CU Cargar Formulario.

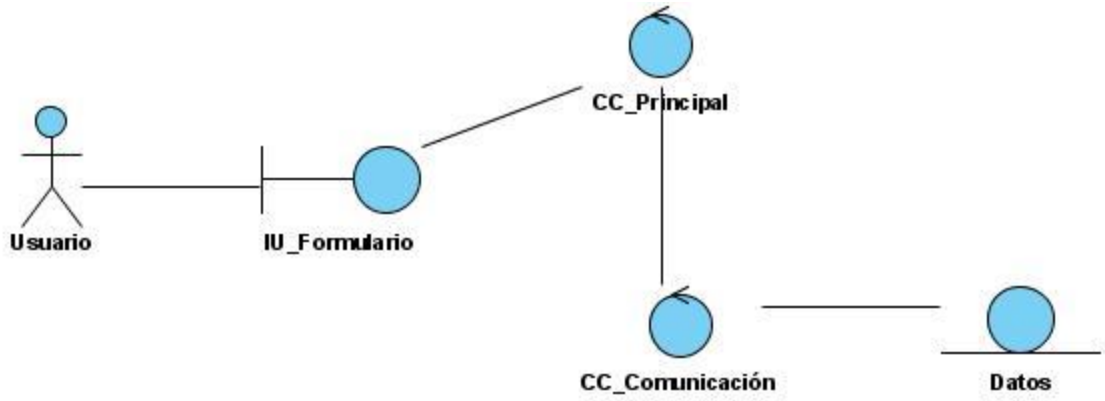


Figura 5 Diagrama de clases del análisis del CU Capturar Datos.

3.3 Modelo del Diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso, centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema. (LARMAN, 1999). Los propósitos del diseño son:

- Adquirir una comprensión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia, tecnologías de interfaz de usuario, tecnologías de gestión de transacciones, entre otras.
- Crear una entrada apropiada y un punto de partida para actividades de implementación subsiguientes capturando los requisitos o subsistemas individuales, interfaces y clases.
- Ser capaces de descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo, teniendo en cuenta la posible concurrencia.
- Ser capaces de visualizar y reflexionar sobre el diseño utilizando una notación común. (JAMES RUMBAUGH, 2000)

3.3.1 Realización de Casos de Uso del Diseño

Una realización de caso de uso del diseño es una colaboración en el modelo de diseño que describe cómo se realiza un caso de uso específico, y cómo se ejecuta, en términos de clases de diseño y sus

objetos. Proporciona una traza directa a una realización de casos de uso del análisis en el modelo de análisis.

Una realización de caso de uso del diseño tiene una descripción textual del flujo de eventos, diagramas de clases que muestran sus clases de diseño participantes, y diagramas de interacción que muestran las realizaciones de un flujo o escenario concreto de un caso de uso en términos de interacción entre objetos del diseño.

3.3.2 Diagrama de clases del diseño

Una clase de diseño es una abstracción sin costuras de una clase o construcción similar en la implementación del sistema. (LARMAN, 1999)

En este diagrama se representan las clases participantes en las realizaciones de caso de uso, subsistemas y sus relaciones. También puede darse el caso de algunas operaciones, atributos y asociaciones sobre una clase específica que son relevantes.

El lenguaje utilizado para especificar una clase de diseño es lo mismo que el lenguaje de programación. Consecuentemente las operaciones, parámetros, atributos, tipos y demás son especificados utilizando la sintaxis del lenguaje de programación elegido. La visibilidad de los atributos y las operaciones de una clase de diseño se especifica con frecuencia. Las relaciones de aquellas clases de diseño implicadas con otras clases, a menudo tienen un significado directo cuando la clase es implementada. Los métodos tienen correspondencia directa con el correspondiente método en la implementación de las clases.

Ver diagramas de clases del diseño por casos de uso en el Anexo 2.

3.3.3 Estructura en paquetes del diseño

Los diagramas de paquetes se usan para reflejar la organización de los paquetes y sus elementos, y para proveer una visualización de sus correspondientes nombres de espacio.(SPARXSYSTEMS, 2007) Este diagrama muestra como la aplicación se divide en agrupaciones lógicas, de igual manera muestra las dependencias que existen entre estas agrupaciones.

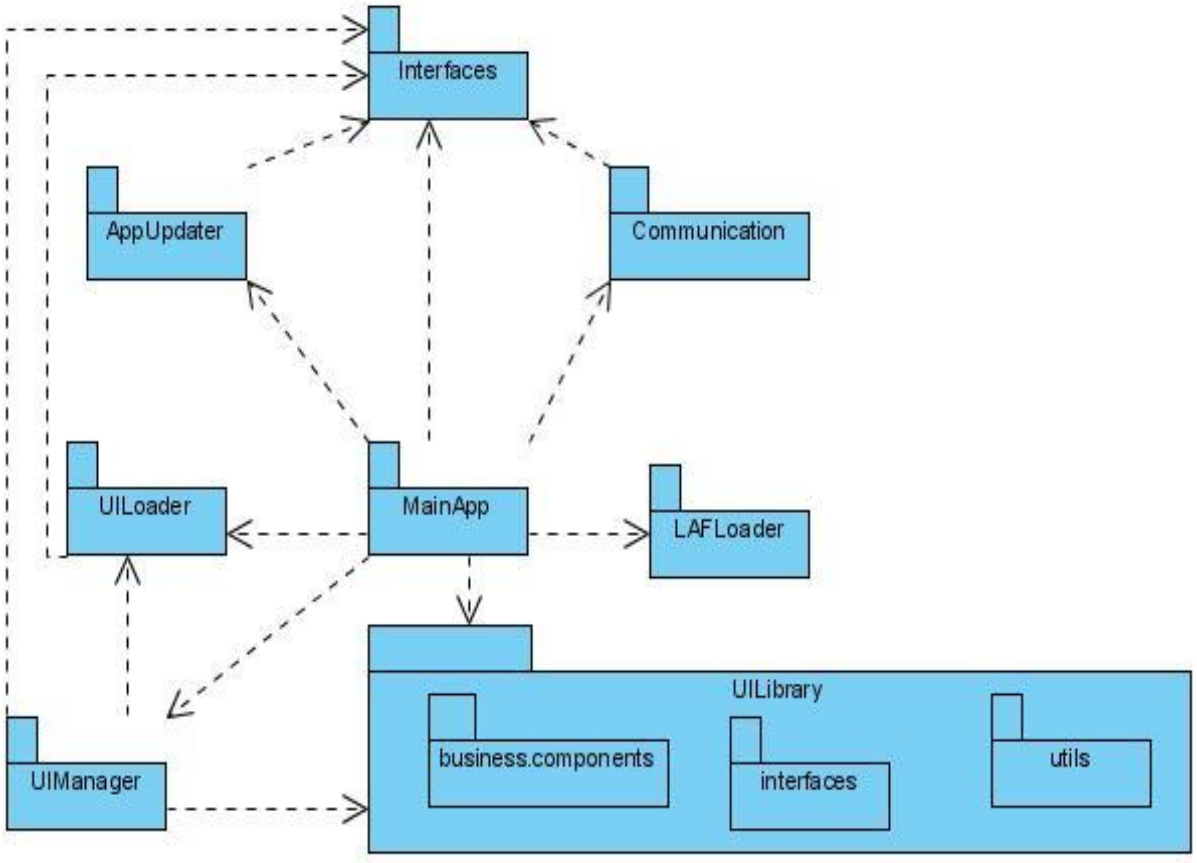


Figura 6 Diagrama por paquetes de la aplicación de gestión de IU.

El paquete "MainApp": Contiene el núcleo de la aplicación, con las clases que este requiere para su funcionamiento.

El paquete "UILibrary": Se encarga de agrupar otros paquetes con clases, interfaces o funcionalidades que son comunes, estas funcionalidades pueden llamarse desde otros paquetes con el objetivo de reutilizar código y aprovechar un mejor rendimiento de la aplicación.

El paquete "Interfaces": Agrupa las interfaces que exponen los componentes *AppUpdater, Communication, UIManager, LAFLoader*.

El paquete "Communication": Forma parte del componente con el mismo nombre, este paquete contiene las clases que se encargan de la comunicación de la aplicación con el proveedor de servicios.

El paquete "UIManager": Forma parte del componente con el mismo nombre, este paquete contiene las clases que se encargan de gestionar las interfaces de usuario, entiéndase por esto cargar las

interfaces con la información correspondiente y además capturar la información de las interfaces cargadas.

El paquete "UILoader": Forma parte del componente con el mismo nombre, este contiene las clases que se encargan de cargar las interfaces de usuario en el *classpath* de la aplicación, para que el componente *UIManager* pueda gestionarlas.

El paquete "AppUpdater": Forma parte del componente con el mismo nombre, este contiene las clases que se encargan de a partir de una solicitud del componente *UILoader*, buscar una interfaz de usuario en un repositorio externo y descargarla para el repositorio local de la aplicación.

El paquete "LAFLoader": Forma parte del componente con el mismo nombre, este contiene las clases necesarias para cargar un archivo de configuración de estilos (Look & Feel) en el *classpath* de la aplicación para que el núcleo de la aplicación pueda cargarlo.

3.3.4 Diagramas de Interacción

Los diagramas de interacción se utilizan para modelar los aspectos dinámicos de un sistema. Estos están compuestos por un conjunto de objetos y sus relaciones, incluyendo los mensajes que se pueden enviar entre ellos.

En UML los diagramas de interacción pueden representarse a través de los Diagramas de Colaboración y/o de los Diagramas de Secuencia. El tipo de diagrama seleccionado para construir los diagramas de interacción de esta aplicación fue el de Secuencia, debido a que muestra cómo los objetos se comunican unos con otros en una secuencia de tiempo, qué sucede en cada momento, y para ello contienen objetos con sus ciclos de vida y los mensajes que se envían entre ellos ordenados secuencialmente.

3.3.4.1 Diagramas de Secuencia

En el diseño, es recomendable representar las interacciones de los objetos que participan en la realización de un caso de uso, mediante diagramas de secuencias de esta forma podemos encontrar mejor las secuencias de interacciones detalladas y ordenadas en el tiempo. En los diagramas de secuencias, se muestran las interacciones entre objetos mediante transferencias de mensajes entre objetos o subsistemas. (LARMAN, 1999)

Ver los diagramas de secuencias del diseño por casos de uso en el Anexo 3.

3.4 Arquitectura de Software

Existen muchas definiciones de Arquitectura del Software y no parece que ninguna de ellas haya sido totalmente aceptada. En un sentido amplio se pudiera estar de acuerdo en que la Arquitectura del Software es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación y tiene la responsabilidad de:

- Definir los módulos principales
- Definir las responsabilidades que tendrá cada uno de estos módulos
- Definir la interacción que existirá entre dichos módulos:
- Control y flujo de datos
- Secuenciación de la información
- Protocolos de interacción y comunicación
- Ubicación en el hardware

La Arquitectura del Software aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño. La definición oficial de Arquitectura del Software, la cual fue dada por la IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) dice así: “La Arquitectura del Software es la organización fundamental de un sistema formado por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución”.

Existen varias corrientes principales de la arquitectura de software, como son la arquitectura estructurada, la arquitectura como etapa de la ingeniería de software orientada a objetos, la arquitectura procesual y metodologías, y la arquitectura basada en patrones, esta ultima además permite la redefinición de estilos como patrones arquitectónicos (POSA).

3.4.1 Estilos y Patrones de Arquitectura de Software

Los estilos expresan la arquitectura en el sentido más formal y teórico, describen entonces una clase de arquitectura, o piezas identificables de las arquitecturas empíricamente dadas. Esas piezas se encuentran repetidamente en la práctica, trasuntando la existencia de decisiones estructurales coherentes. Una vez que se han identificado los estilos, es lógico y natural pensar en reutilizarlos en situaciones semejantes que se presenten en el futuro.(REYNOSO, 2004).

Los patrones arquitectónicos capturan existencia, experiencia comprobada en el desarrollo del software y ayudan a promover buenas prácticas de diseño. Cada patrón es específico a un problema recurrente en el diseño e implementación de un sistema de software. Un patrón es un par problema - solución, resultado de la experiencia en el diseño de arquitecturas de sistemas y propone los patrones arquitectónicos como descripción de un problema particular y recurrente de diseño, que aparece en contextos de diseño específicos, y presenta un esquema genérico demostrado con éxito para su solución. El esquema de solución se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades y desarrollos, así como también la forma como éstos colaboran entre sí.

En sentido general se reconoce que los patrones se refieren más bien a prácticas de re-utilización y los estilos conciernen a teorías sobre la estructuras de los sistemas a veces más formales que concretas. Igual que los patrones de arquitectura, todos los estilos tienen un nombre, como los reflejados a continuación.

Estilos Arquitectónicos Naturales de Arquitectura de Software

- Estilos de Flujos de Datos
    - Tuberías y Filtros
- Estilos Centrados en Datos
    - Arquitectura Pizarra o Repositorio
- Estilos de Llamadas y Retornos
    - Modelo Vista Controlador (MVC)
    - Arquitectura en Capas
    - Arquitectura Orientada a Objetos
    - Arquitectura Basada en Componentes
- Estilos de Código Móvil
    - Arquitectura de Maquina Virtuales
- Estilos heterogéneos
    - Sistema de Control de Procesos
    - Arquitectura Basada en Atributos
- Estilos Peer to Peer (punto a punto)
    - Arquitectura basada en Eventos
    - Arquitectura Orientada a Servicios

- Arquitectura Basada en Recursos

(CONFERENCIA#6, 2008)

3.4.2 Descripción de la Arquitectura Utilizada

Una *arquitectura basada en componentes* describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. Características muy relevantes de la tecnología de programación basada en componentes son la modularidad, la rehusabilidad y componibilidad y en todos ellos coincide con la tecnología orientada a objetos de la que se puede considerar una evolución.

Otras características de esta arquitectura son:

- Es un estilo de diseño para aplicaciones compuestas de componentes individuales.
- Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas.
- Define una aproximación de diseño que usa componentes discretos, los que se comunican a través de interfaces que contienen métodos, eventos y propiedades. (PELAEZ, 2009)

Beneficios de esta arquitectura

- Facilidad de Instalación. Cuando una nueva versión esté disponible, se podrá reemplazar la versión existente sin impacto en otros componentes o el sistema como un todo.
- Costos reducidos. El uso de componentes de terceros permite distribuir el costo del desarrollo y del mantenimiento.
- Facilidad de desarrollo. Los componentes implementan una interfaz bien definida para proveer la funcionalidad definida permitiendo el desarrollo sin impactar otras partes del sistema.
- Reusable. El uso de componentes reutilizables significa que ellos pueden ser usados para distribuir el desarrollo y el mantenimiento entre múltiples aplicaciones y sistemas.

3.5 Visión general de la aplicación a través de sus componentes

La aplicación consta de cinco componentes principales que tienen diferentes propósitos, además de un núcleo principal que es el orquestador de los mismos. A continuación una explicación más detallada de las funcionalidades de cada uno de estos componentes:

UIManager

Este componente es el responsable de interactuar con la interfaz visual, el mismo muestra las interfaces visuales, relaciona el XML de los datos con los componentes de los formularios, extrae la información de los formularios una vez los usuarios terminen de interactuar con ellos y la convierte a XML. Requiere implementar la interfaz *IUIManager* y utiliza al componente *UILoader*.

UILoader

Su función es cargar el fichero con la interfaz desde el repositorio local, agregarlo al *classpath* de la aplicación y retornar una instancia del mismo. En el caso de no encontrarse en el repositorio local este componente acude a *AppUpdater* para actualizar el repositorio local, por lo que existe una relación entre ellos dos. Para crear una nueva implementación de este componente se debe implementar la interfaz *IUILoader*.

AppUpdater

Se relaciona directamente con el *UILoader,* a partir de que este último no encuentre la interfaz solicitada en el repositorio local, este accede al *AppUpdater* que es el encargado de buscar esta interfaz en un repositorio externo, al encontrarla la descarga al repositorio local, de manera que el proceso continúe normalmente, es decir que el *UILoader* pueda cargar la interfaz. Este componente tiene que implementar la interfaz *IAppUpdater*.

Communication

Este componente se encarga de establecer la conexión con el proveedor de servicios, esta conexión se realiza a través de los servicios web que brinda el mismo. Otra función de este componente es la de convertir la información obtenida del proveedor de servicios a XML, que es el formato que entiende el resto de la aplicación, de esta manera se convierte en el punto de entrada a la aplicación, y al mismo tiempo en el punto de salida, pues también se encarga de realizar el proceso inverso al enviar la información capturada de las interfaces de usuario a ese mismo proveedor de servicios.

LAFLoader

Su función es la de cargar el archivo de configuración de estilos de la aplicación (Look & Feel), para eso se encarga de cargar este archivo en el *classpath* de la aplicación y específicamente de agregar la clase que se quiera cargar de este “Look & Feel”, de manera tal que se pueda acceder a ella en tiempo de ejecución, y establecer un estilo determinado para las interfaces.

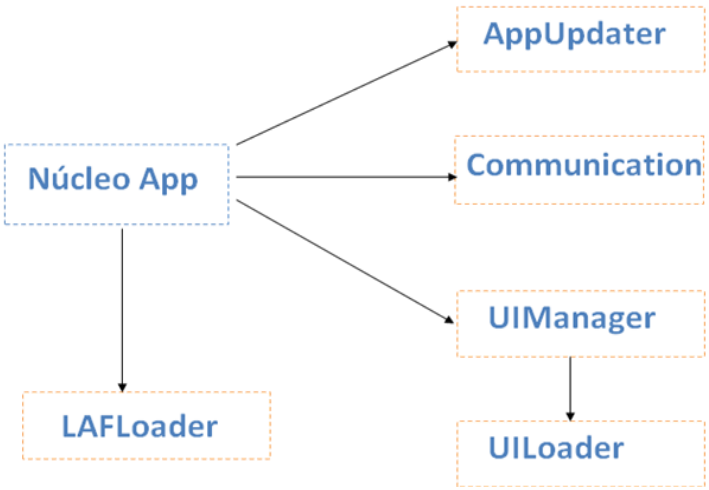


Figura 7 Estructura de la aplicación a través de sus componentes.

3.6 Definiciones de diseño que se aplican

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias. Los patrones de diseño pretenden:

- No ser reiterativos en la búsqueda de soluciones a problemas que son conocidos y ya han sido solucionados.
- Construir un vocabulario que sea común entre los diseñadores, para de esta manera estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.
- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.

Muchos diseñadores y arquitectos de software han definido el término de patrón de diseño de varias formas que corresponden al ámbito a la cual se aplican los patrones. Luego, se dividió los patrones en diferentes categorías de acuerdo a su uso.

Fue por el año 1994, que apareció el libro "Design Patterns: Elements of Reusable Object Oriented Sofware" escrito por los ahora famosos Gang of Four (GoF, que en español es la pandilla de los

cuatro) formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. El grupo de GoF clasificaron los patrones en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

- Creacionales: Patrones creacionales tratan con las formas de crear instancias de objetos. El objetivo de estos patrones es de abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.
- Estructurales: Los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.
- Comportamiento: Los patrones de comportamiento nos ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.

Por otro lado Craig Larman en su libro UML y Patrones introduce lo que se conoce como patrones GRASP (General Responsability Assignment Software Patterns). Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Los primeros cinco patrones GRASP son:

- Experto: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.
- Creador: Asignarle a la clase B la responsabilidad de crear una instancia de clase A
- Alta cohesión: Asignar una responsabilidad de modo que la cohesión siga siendo alta.
- Bajo Acoplamiento: Asignar una responsabilidad para mantener bajo acoplamiento.
- Controlador: Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase

3.6.1 Patrón Inyección de Dependencias

El patrón inyección de dependencias, es una técnica que como su nombre lo indica busca facilitar la resolución de dependencias entre objetos.(*PAEZ*, 2006). Este patrón pretende eliminar al máximo las dependencias entre componentes, básicamente consiste en que en vez de ser una clase la que crea objetos, estos se inyectan o resuelven en el último momento. Está basado en interfaces, se encarga de alterar el comportamiento de la clase sin cambiar internamente ésta. Para lograr esto el desarrollador

genera código contra una interfaz para la clase y usa un contenedor que inyecta la instancia de la clase, basándose en la interfaz o el tipo del objeto.

Este patrón evita las dependencias en las implementaciones de las clases que colaboran entre ellas, mediante su dependencia solo a las interfaces a las que estas clases se adhieren. Las técnicas para inyectar instancias son: inyección por interfaz, inyección por constructor, inyección por propiedad (setter) e inyección por método. El desacoplamiento entre componentes ofrece múltiples ventajas, entre ellas:

- Facilidad para realizar cambios en la aplicación sin afectar su estabilidad.
- Simplifica las pruebas unitarias.
- Aumenta la flexibilidad y mantenibilidad.
- La búsqueda de recursos es removida del código de la aplicación.
- No hay dependencia con respecto a una Interfaz de Programación de Aplicaciones provista por un contenedor específico.

3.6.2 Observador

El patrón observador (observer), define una dependencia de uno a muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y se actualicen automáticamente todos los objetos que dependen de él. (ERICH GAMMA, 1994)

Diferentes tipos de Objetos suscriptores están interesados en el cambio de estado o eventos de un Objeto publicador, y quieren reaccionar cada uno a su manera cuando el publicador genere un evento. También interesa que el emisor mantenga bajo acoplamiento con los suscriptores.

Este patrón suele observarse en los frameworks de interfaces gráficas orientados a objetos, en los que la forma de capturar los eventos es suscribir escuchadores (en inglés “listeners”) a los objetos que pueden disparar eventos.

3.6.3 Factoría Abstracta

El patrón factoría abstracta (abstract factory) parte de la idea de abstraer el código de creación de objetos del resto de la aplicación. Una factoría abstracta es una clase que los objetos que devuelve son a su vez factorías. Por este motivo, para que sea efectiva, estas factorías que devuelve, deben ser

de la misma familia (es decir, tener antecesores comunes), como ocurre con las factorías normales.(MOLPECERES, 2002)

Un ejemplo de utilización de este patrón es en Java donde podemos encontrar el mismo programa con distintos aspectos cambiando solo la línea de código que se encarga de indicar el "look&feel". Esto es posible gracias a que la clase *UIManager* de Java es una factoría abstracta que genera factorías de componentes visuales. De manera que cuando se crea un componente visual para una interfaz de usuario, Java acude al "look&feel" seleccionado, que es una factoría, y le pide el componente escogido con el aspecto que corresponda.

La ventaja que obtenemos del uso de este patrón está en la posibilidad de ampliar la funcionalidad de la aplicación con solo añadir una nueva familia de elementos y un mínimo retoque de la aplicación (para llevar a cabo la integración de esa familia).

3.7 Descripción de las clases del diseño

Nombre: Main	
Tipo de clase : controladora	
Atributo	**Tipo**
nodeMenu	Node
idMainUI	String
nodeUIData	Node
uiManager	IUIManager
uiLoader	IUILoader
lafLoader	LAFLoader
communication	DefaultCommunication
Para cada responsabilidad:	
Nombre:	main()
Descripción:	Método donde se ejecuta la aplicación. Las funciones se ejecutan a través de los atributos que representan objetos de las clases principales de la aplicación.
Nombre:	readXml(String fileName) : void
Descripción:	Recibe como parámetro la dirección de un archivo de configuración lo interpreta.
Nombre:	readXml(Document document) : void
Descripción:	Recibe como parámetro un documento XML de configuración y lo interpreta.

Tabla 3.1 Descripción de la clase "Main".

Nombre: AppConfigReader	
Tipo de clase : controladora	
Atributo	**Tipo**
keys	Hashtable
Para cada responsabilidad:	
Nombre:	AppConfigReader()
Descripción:	Constructor sin parámetros de la clase, que inicializa el *hashtable* y que llama al método *readXML*.
Nombre:	readXml(String fileName) : void
Descripción:	Interpreta el XML de configuración especificado como parámetro, adicionándole al hashtable los pares llave-valor, con la información del XML.
Nombre:	getValue(Object key) : Object
Descripción:	A partir de una llave especificada por parámetro, retorna el valor correspondiente a esa llave en el *hashtable*.
Nombre: IUILoader	
Tipo de clase : interface	
Atributo	**Tipo**
-	-
Para cada responsabilidad:	
Nombre:	getUI(String idUI): Object
Descripción:	Dado el identificador de la interfaz especificada por parámetro, ya sea una plantilla o formulario, retorna una instancia de la clase de la interfaz solicitada.
Nombre:	setDirectory(String directory): void
Descripción:	Modifica la ubicación del repositorio local de interfaces por una nueva especificada por parámetro.

Tabla 3.2 Descripción de la clase “AppConfigReader”.

Nombre: IUIManager	
Tipo de clase : interfaz	
Atributo	**Tipo**
-	-
Para cada responsabilidad:	
Nombre:	loadMainUI() : void

Descripción:	Se encarga de cargar la interfaz principal (Plantilla).
Nombre:	loadUI(String id): void
Descripción:	Se encarga de cargar el formulario especificado por parámetro y adicionarlo a la plantilla.
Nombre:	setUIData(Node node): void
Descripción:	Se encarga de enlazar las propiedades y valores de los componentes especificados en un nodo de un documento con estructura XML, con sus equivalentes en el formulario cargado.
Nombre:	setMenuData(Node node): void
Descripción:	Se encarga de inyectarle los datos correspondientes al menú contenido en la plantilla. La información del menú la obtiene del parámetro especificado, un nodo de un documento con estructura XML.
Nombre:	getUIData(): Node
Descripción:	Se encarga de crear y retornar un documento con estructura de XML, que contiene toda la información sobre el formulario cargado, la información capturada de los componentes del formulario, la acción generada por el usuario sobre el mismo.
Nombre:	getMainUI(): Object
Descripción:	Retorna la plantilla.
Nombre:	setUILoader(IUILoader uiLoader): void
Descripción:	Se encarga de inyectar a esta clase una dependencia de la interfaz *IUILoader*.
Nombre:	setIdUI(String idUI): void
Descripción:	Se encarga de modificar el identificador de la plantilla a cargar.

Tabla 3.3 Descripción de la interfaz "IUIManager".

Nombre: ICommunication	
Tipo de clase : interfaz	
Atributo	**Tipo**
-	-
Para cada responsabilidad:	
Nombre:	performAction() : void
Descripción:	Se encarga de enviar y recibir la información a través de un servicio web.
Nombre:	createSessionByAction():Session
Descripción:	Se encarga de crear un objeto de la clase *Session* el cual contiene la acción generada por el usuario.

Nombre:	createSessionByDocument(Document document): void
Descripción:	Recibe como parámetro un documento XML para crear un objeto de la clase *Session* el cual contiene la acción generada por el usuario así como el resto de los datos que van a ser enviados a través del servicio web.
Nombre:	createDocumentBySession(): Document
Descripción:	Obtiene la información de un objeto de la clase *Session* y partir de dicha información crea un documento XML con la información de los componentes que se cargarán en el formulario.

Tabla 3.4 Descripción de la interfaz "ICommunication".

Nombre: DefaultUILoader	
Tipo de clase : controladora	
Atributo	**Tipo**
jarPath	String
Para cada responsabilidad:	
Nombre:	DefaultUILoader(String jarPath)
Descripción:	Constructor con parámetro, que se encarga de inicializar el atributo *jarPath* con el parámetro que contiene la ubicación del repositorio local de interfaces.
Nombre:	existUI(String filename): boolean
Descripción:	Valida si el archivo especificado por parámetro existe o no, retorna false si no existe y true si existe.
Nombre:	setDirectory(String directory): void
Descripción:	Modifica la ubicación del repositorio local de interfaces por una nueva especificada por parámetro.
Nombre:	getUI(String idUI): Object
Descripción:	Dado el identificador de la interfaz, pasado por parámetro, verifica si esta existe en el repositorio local, en caso de que no exista actualiza el repositorio local, descargando la interfaz solicitada de un repositorio externo para este, en ambos casos carga la clase correspondiente a esta interfaz en el *classpath* de la aplicación y retorna una instancia de la misma, de tipo *Object*, independientemente de si la interfaz es una plantilla o un formulario.

Tabla 3.5 Descripción de la clase "DefaultUILoader".

Nombre: DefaultUIManager	
Tipo de clase : controladora	
Atributo	**Tipo**
idUI	String
uiLoader	IUILoader
ui	IBusinessUI
mainUI	IMainAppUI
Para cada responsabilidad:	
Nombre:	loadMainUI() : void
Descripción:	Se encarga de cargar la interfaz principal (Plantilla).
Nombre:	loadUI(String id): void
Descripción:	Se encarga de cargar el formulario especificado por parámetro apoyándose en *IUILoader* y de adicionarlo a la plantilla.
Nombre:	setUIData(Node node): void
Descripción:	Se encarga de enlazar las propiedades y valores de los componentes especificados en un nodo de un documento con estructura XML, con sus equivalentes en el formulario cargado.
Nombre:	setMenuData(Node node): void
Descripción:	Se encarga de inyectarle los datos correspondientes al menú contenido en la plantilla. La información del menú la obtiene del parámetro especificado, un nodo de un documento con estructura XML.
Nombre:	getUIData():Document
Descripción:	Se encarga de crear y retornar un documento con estructura de XML, que contiene toda la información sobre el formulario cargado, la información capturada de los componentes del formulario, la acción generada por el usuario sobre el mismo.
Nombre:	getMainUI():IMainAppUI
Descripción:	Retorna la plantilla.
Nombre:	setUILoader(IUILoader uiLoader): void
Descripción:	Se encarga de inyectar a esta clase una dependencia de la interfaz *IUILoader*.
Nombre:	setIdUI(String idUI): void
Descripción:	Se encarga de modificar el identificador de la plantilla a cargar.
Nombre:	setInformation String idUI, IUILoader uiLoader): void
Descripción:	Se encarga de modificar el identificador de la plantilla o formulario a cargar y del objeto que va a cargar dicha plantilla o formulario.

Tabla 3.6 Descripción de la clase "DefaultUIManager".

Nombre: DefaultCommunication	
Tipo de clase : controladora	
Atributo	**Tipo**
action	String
session	Session
result	Session
Para cada responsabilidad:	
Nombre:	DefaultCommunication ()
Descripción:	Constructor sin parámetro, que se encarga de inicializar los atributos de la clase.
Nombre:	ActionByComponent(String compname): String[]
Descripción:	Retorna un arreglo con las acciones que ejecutan cada uno de los componentes del formulario o plantilla.
Nombre:	ValueByComponent(String location): String[]
Descripción:	Retorna un arreglo con los valores de cada uno de los componentes del formulario o plantilla.
Nombre:	createSessionByAction(): Session
Descripción:	Se encarga de crear un objeto de la clase *Session* el cual contiene la acción generada por el usuario.
Nombre:	createSessionByDocument(Document d): void
Descripción:	Recibe como parámetro un documento XML para crear un objeto de la clase *Session* el cual contiene la acción generada por el usuario así como el resto de los datos que van a ser enviados a través del servicio web.
Nombre:	performAction(): void
Descripción:	Se encarga de enviar y recibir la información a través de un servicio web.
Nombre:	createDocumentBySession(): void
Descripción:	Obtiene la información de un objeto de la clase *Session* y partir de dicha información crea un documento XML con la información de los componentes que se cargarán en el formulario.
Nombre:	getSession(): Session
Descripción:	Retorna el atributo *session* de la clase.
Nombre:	setSession(Session session): void
Descripción:	Modifica el atributo *session* de la clase asignándole el parámetro *session*.
Nombre:	getResult(): Session

Descripción:	Retorna el atributo *result* de la clase.
Nombre:	setResult(Session result): void
Descripción:	Modifica el atributo *result* de la clase asignándole el parámetro *result*.
Nombre:	getAction(): String
Descripción:	Retorna el atributo *action* de la clase.
Nombre:	setAction(String action): void
Descripción:	Modifica el atributo *action* de la clase asignándole el parámetro *action*.

Tabla 3.7 Descripción de la clase "DefaultCommunication".

Nombre: LafLoader	
Tipo de clase : controladora	
Atributo	**Tipo**
jarFile	String
lafClassName	String
Para cada responsabilidad:	
Nombre:	LAFLoader(String jarFile, String lafClassName)
Descripción:	Constructor con parámetro, que se encarga de inicializar los atributos de la clase con el parámetro que recibe.
Nombre:	loadLAF(): String
Descripción:	Retorna el nombre de la clase que contiene el "Look & Feel" que se cargara en la aplicación.

Tabla 3.8 Descripción de la clase "LafLoader".

Nombre: IComponent	
Tipo de clase : Interfaz	
Atributo	**Tipo**
-	-
Para cada responsabilidad:	
Nombre:	getId():String
Descripción:	Devuelve el nombre del componente.
Nombre:	setId(String id): void
Descripción:	Cambia el nombre del componente por el especificado por parámetro.

Nombre:	getComponentValue(): Object
Descripción:	Retorna el valor del componente.
Nombre:	setComponentValue(Object value): void
Descripción:	Le inyecta al componente el valor especificado por parámetro.
Nombre:	getAccion():String
Descripción:	Devuelve la acción del componente.
Nombre:	setAccion(String[] accion): void
Descripción:	Cambia la acción del componente por la especificada por parámetro.

Tabla 3.9 Descripción de la interfaz "IComponent".

Nombre: IMainAppUI	
Tipo de clase : interfaz	
Atributo	**Tipo**
-	-
Para cada responsabilidad:	
Nombre:	addUIPanel(IBusinessUI container) : void
Descripción:	Adiciona un formulario especificado por parámetro a la plantilla, específicamente a un panel central en la misma.
Nombre:	setMenuData(Node node): void
Descripción:	Construye el menú con los datos pasados por parámetros.
Nombre:	getActionCommand(): String
Descripción:	Retorna una cadena que identifica la acción correspondiente al evento generado por el usuario en el formulario.
Nombre:	getMainMenu():IMainMenu
Descripción:	Retorna el menú de la plantilla
Nombre:	addActionListener(ActionListener listener): void
Descripción:	Adiciona un escuchador de eventos a un listado de escuchadores de eventos.
Nombre:	removeActionListener(ActionListener listener): void
Descripción:	Elimina de un listado de escuchadores de eventos un escuchador especificado por parámetro.

Tabla 3.10 Descripción de la interfaz "IMainAppUI".

Ver el resto de las descripciones de las clases del diseño en el Anexo 4.

3.8 Conclusiones

Se realizó el análisis de la propuesta de solución logrando una abstracción de la aplicación, a través de una representación de los casos de usos en un diagrama de clases del análisis. Con el modelo de diseño se logró abstraer el modelo de implementación, siendo la entrada esencial en las actividades relacionadas con la implementación, para eso se generaron diferentes artefactos como diagramas de clases del diseño, diagramas de interacción, los cuales son una aproximación real a la futura implementación.

El diseño de la aplicación se realizó a partir de una arquitectura basada en componentes, la cual tiene como características fundamentales la modularidad, la rehusabilidad y componibilidad, además de que coincide con la tecnología orientada a objetos de la que se puede considerar una evolución.

Durante el diseño se utilizaron diferentes patrones como el de *inyección de dependencias* que permitió eliminar al máximo la dependencia entre componentes, otro patrón fue el *observador* empleado en el tratamiento de los eventos generados en los formularios, y otro es el *factoría abstracta* para el manejo de los estilos de los componentes. Las descripciones de las clases, permitieron un mayor entendimiento de la aplicación, de manera que quedaron bien detalladas todas las funcionalidades de cada una de las clases y sus atributos.

CAPITULO IV. IMPLEMENTACIÓN Y PRUEBA

4.1 Introducción

La fase de implementación en el desarrollo de un producto de software, es el mecanismo donde se ponen en práctica todas las descripciones y arquitecturas propuestas en las fases de análisis y diseño, es el complemento del trabajo de las fases que lo preceden dentro del proceso unificado de software. La implementación ofrece una materialización precisa de los requisitos. En ella se obtienen como resultado componentes de código que se compilan e integran en versiones ejecutables.

Una de las últimas fases del ciclo de vida antes de entregar un programa para su explotación, es la fase de pruebas. La fase de pruebas del sistema tiene como objetivo verificar el sistema para comprobar si este cumple sus requisitos. Dentro de esta fase pueden desarrollarse varios tipos distintos de pruebas en función de los objetivos de las mismas.

4.2 Modelo de Despliegue

El modelo de despliegue es un modelo de objeto que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre nodos de cómputo. Se utiliza como entrada fundamental en las actividades de diseño e implementación debido a que la distribución del sistema tiene una influencia principal en su diseño. (LARMAN, 1999)

El modelo de despliegue describe la distribución física de de los componentes lógicos desarrollados en la aplicación. Es decir, se sitúa el software en el hardware que lo contiene. En el caso de la aplicación, un nodo representaría la PC cliente en la cual estaría desplegada la aplicación de gestión de IU, otro nodo sería el proveedor de servicios que en este caso se trabajó con el Bus de Servicios Empresariales “Open ESB”, la comunicación entre estos es a través de servicios web, por un protocolo de tipo SOA/HTTP.

Diagrama de despliegue

Figura 8 Diagrama de despliegue de la aplicación.

4.3 Modelo de componentes

El modelo de componentes describe, como se implementan las clases en término de componentes, como ficheros de código fuente, ejecutables entre otros. Describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponible en el entorno de implementación y en el lenguaje de programación utilizado. Además muestra las dependencias entre componentes.(LARMAN, 1999). Los diagramas de componentes son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre los elementos de implementación.

Ver diagrama de componentes de la aplicación en el Anexo 5.

4.4 Descripción Preliminar del Modelo de Pruebas

El modelo de pruebas permite principalmente, probar los componentes ejecutables del modelo de interacción. También se puede probar aspectos específicos como, la interfaz. Contiene una gran cantidad de casos de prueba, procedimientos de prueba y componentes de prueba. Entre los casos de prueba se puede distinguir dos tipos comúnmente utilizados: las llamadas pruebas de “caja negra” y las de “caja blanca”.

Las pruebas de **caja blanca** estas son las encargadas de comprobar el comportamiento de las interacciones de los componentes internos del sistema.

Las pruebas de **caja negra** verifican el resultado de la interacción entre los usuarios y el sistema, comprobando que se cumplan las precondiciones y poscondiciones especificadas para cada caso de uso siguiendo la secuencia de acciones previstas para el mismo.

4.4.1 Caja Negra

Se realizaron pruebas de caja negra a cada caso de uso. La técnica seleccionada es la de **partición equivalente**. Para realizar esta se divide el dominio de entrada de un programa en clases de datos que tengan algo en común, de ahí derivan clases de prueba y por lo tanto también derivan clases de errores, de esta forma no hay necesidad de ejecutar muchas pruebas para encontrar errores genéricos. Se evalúan las clases de equivalencias posibles para una condición de entrada.

- Clases de equivalencias: Son los conjuntos de estados (válidos o no) para las condiciones de entrada.
- Condiciones de entrada: Valor numérico, rango de valores, **condición booleana (si o no)**.

- Si una condición de entrada especifica un rango, se propone definir una clase de equivalencia valida y 2 inválidas.
- Si una condición de entrada especifica un número, un valor, se define una clase de equivalencia igual al caso anterior, (1 valida y 2 inválidas).
- Si una condición de entrada especifica un miembro de un conjunto, se define una condición valida y una invalida.
- **Si es booleana, se define una clase de equivalencia igual al caso anterior (1 valida y 1 inválida).**

El caso de prueba se define para el **caso de uso Cargar Plantilla**, el mismo inicia cuando el usuario ejecuta la aplicación, la cual debe obtener el identificador de la plantilla, el nombre y la dirección del archivo de configuración de estilos, y los datos del menú, con toda esta información la aplicación debe cargar la plantilla y mostrársela al usuario.

Matriz de Casos de Prueba del caso de uso Cagar Plantilla

Id del escenario	**Escenario**	**Variable 1** Archivo de configuración	**Variable 2** Menú	**Variable 3** Plantilla repositorio local	**Variable 4** Plantilla repositorio externo	**Respuesta del Sistema**	**Resultado de la Prueba**
EC 1	Datos correctos para búsqueda en el repositorio local	V	V	V	I(No se utiliza)	Muestra la plantilla al usuario con el menú y estilo especificado.	*Satisfactorio*
EC2	Datos correctos para búsqueda en el repositorio externo	V	V	I(no se encuentra la planilla)	V	Muestra la plantilla al usuario con el menú y estilo especificado.	*Satisfactorio*

EC3	Datos incorrectos	I(no encontrado)	V	V	I(No se utiliza)	Muestra la plantilla con el estilo por defecto.	*Satisfactorio*
		V	I(no encontrado)	V	I(No se utiliza)	Muestra la plantilla sin menú.	*Satisfactorio*

Tabla 4.1 Matriz del caso de prueba para el caso de uso Cargar Plantilla.

El caso de prueba se define para el **caso de uso Capturar Datos**, específicamente para la búsqueda por cédula de un ciudadano dentro del proceso de enrolamiento de datos, el cual inicia cuando el usuario introduce la cédula del ciudadano, seguidamente la aplicación valida el valor introducido y si el mismo es correcto se deben obtener todos los datos previamente registrados durante el proceso de captura de datos para ese ciudadano y mostrar una planilla de control con los mismos.

Matriz de Casos de Prueba

Id del escenario	**Escenario**	**Var 1** Cédula	**Respuesta del Sistema**	**Resultado de la Prueba**
EC 1	Datos incorrectos	I(caracteres distintos de números)	Muestra mensaje de error de validación del campo incorrecto	*Satisfactorio*
		I(cantidad de caracteres numéricos distinto de 8)	Muestra mensaje de error de validación del campo incorrecto	*Satisfactorio*
		I(campo vacío)	Muestra mensaje de error de validación del campo incorrecto	*Satisfactorio*

		I(no encontrada)	Muestra mensaje de error indicando que la cédula no se encontró en el servidor.	
	Datos correctos	V	Muestra la planilla de control con los datos del ciudadano.	*Satisfactorio*

Tabla 4.2 Matriz del caso de prueba para el caso de uso Capturar Datos.

El caso de prueba siguiente se define para el **caso de uso Cargar Formulario**, el cual inicia cuando el usuario selecciona una opción del menú o presiona un botón de un formulario ya cargado, seguidamente se debe mostrar en la plantilla el formulario correspondiente a la opción seleccionada, para alcanzar este resultado la aplicación accede al proveedor de servicio y obtiene la información referente a ese formulario: identificador del formulario, listado de propiedades, valores, y acciones de los componentes, además de los datos del menú en caso de que sea necesario actualizar el mismo. **Ver la matriz de casos de prueba para este caso de uso en el Anexo 6.**

4.4.2 Caja Blanca

La prueba del camino básico es una técnica de prueba de caja blanca que permite obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Se basa en construir un caso de prueba por camino básico que se encuentre en el grafo de programa asociado al método de la clase que se desea someter a pruebas. A continuación se enumera las sentencias de código del procedimiento a probar:

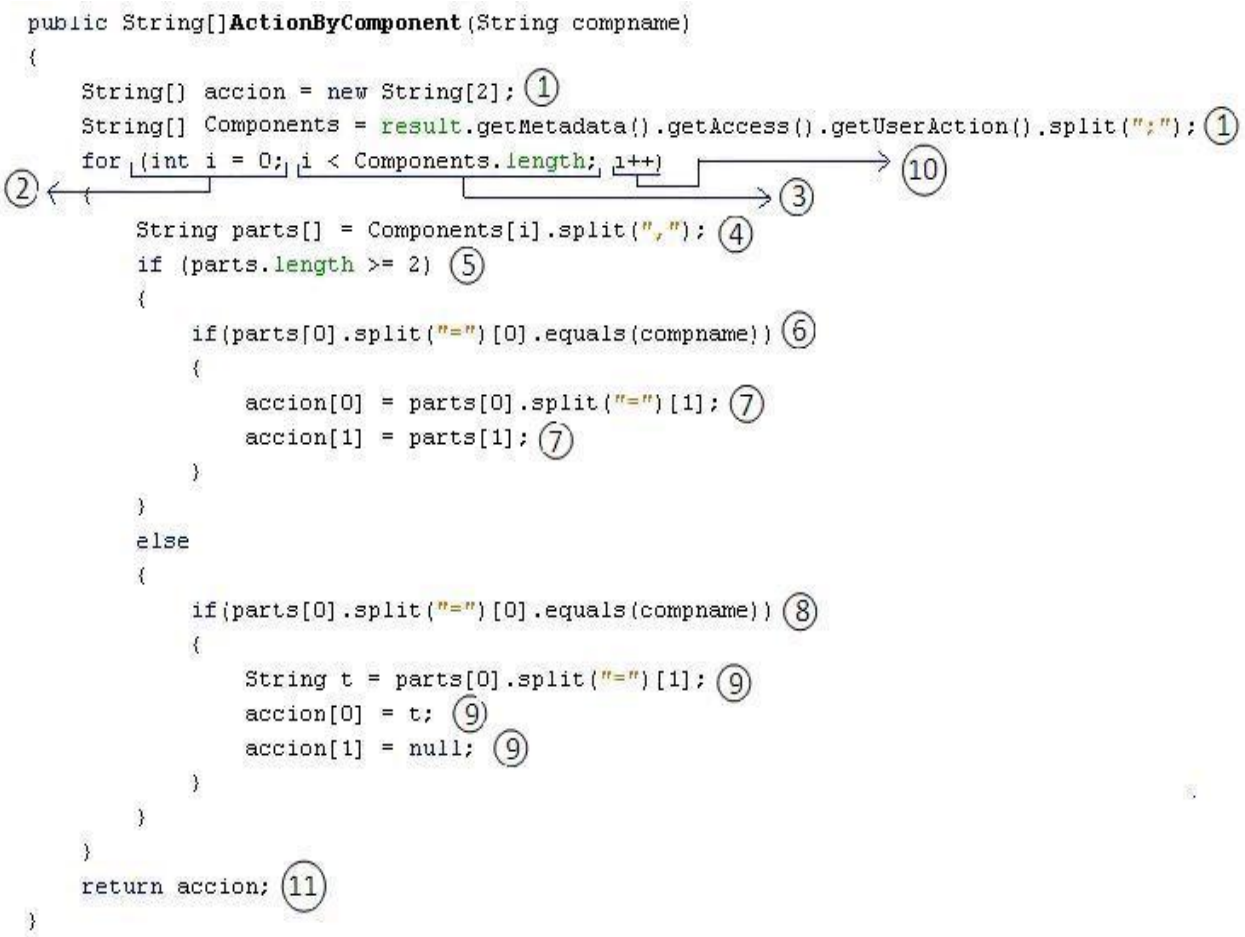


Figura 9 Sentencias de código del procedimiento a probar.

A continuación se construye el grafo de flujo asociado al código anterior.

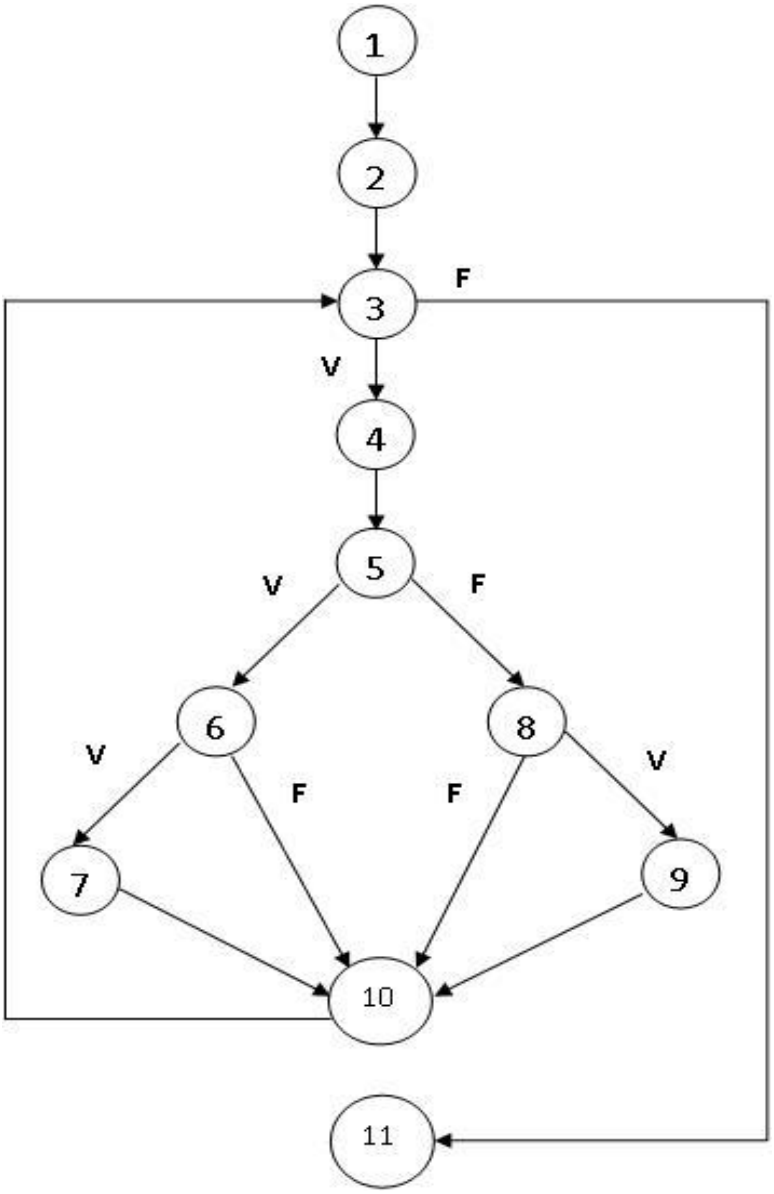


Figura 10 Grafo de flujo del procedimiento a probar.

Luego de haber construido el grafo, se realiza el cálculo de la complejidad ciclomática (métrica del software que proporciona una medición de la complejidad lógica de un programa). Fórmulas para calcular complejidad ciclomática.

4.5 V (G) = (A – N) + 2.
4.6 V (G) = P + 1.
4.7 V (G) = R.

Aplicando estas fórmulas al grafo de flujo de la figura x se obtienen los siguientes resultados:
Calculando mediante la fórmula 1: V (G) = (14 -11) + 2 V (G) = 5.
Calculando mediante la fórmula 2: V (G) = 4 + 1 V (G) = 5.
Calculando mediante la fórmula 3: V (G) = 5.

El cálculo efectuado mediante las tres fórmulas ha dado el mismo valor, por lo que se puede decir que la complejidad ciclomática del código es de 5, lo que significa que existen cinco posibles caminos por donde el flujo puede circular, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado. Seguidamente es necesario representar los caminos básicos por los que puede recorrer el flujo.

- Camino básico # 1. **(1-2-3-4-5-6-7-10-3-11)**
- Camino básico # 2. **(1-2-3-4-5-6-10-3-11)**
- Camino básico # 3. **(1-2-3-4-5-8-10-3-11)**
- Camino básico # 4. **(1-2-3-4-5-8-9-10-3-11)**
- Camino básico # 5. **(1-2-3-11)**

Después de haber extraído los caminos básicos del flujo, se procede a ejecutar los casos de pruebas para este procedimiento, se debe realizar al menos un caso de prueba por cada camino básico. Para realizar los casos de pruebas es necesario cumplir con las siguientes exigencias:

Descripción: Se hace la entrada de datos necesaria, validando que ningún parámetro obligatorio pase nulo al procedimiento o no se entre algún dato erróneo.

Condición de ejecución: Se especifica cada parámetro para que cumpla una condición deseada para ver el funcionamiento del procedimiento.

Entrada: Se muestran los parámetros que entran al procedimiento

Resultados Esperados: Se expone resultado que se espera que devuelva el procedimiento.

Caso de prueba para el camino básico # 1

Descripción: Los datos de entrada cumplirán con los siguientes requisitos: El nombre del componente pasado por parámetro debe ser de tipo "*String*", Components debe ser un arreglo de "*String*" que contiene en cada posición la información referente a un componente, parts debe ser un arreglo de

"*String*" que contiene en la primera posición la información del nombre y acción de un componente específico, y en la segunda posición puede o no tener el nombre del componente con quien se relaciona.

Condición de ejecución: El nombre del componente debe ser "aceptar", los valores del arreglo Components deben ser ["aceptar = autentication/credential/captura/entrardato/, tiptramite","cancelar=close"].

Entrada:
compname = "aceptar"
Components = ["aceptar = autentication/credential/captura/entrardato/, tiptramite","cancelar=close"]

Resultados esperados: Se espera que el arreglo con nombre "acción", contenga información sobre el componente especificado por parámetro.

Resultado obtenido: Se obtuvo el arreglo acción con los siguientes datos: acción = ["autentication/credential/captura/entrardato/", "tiptramite"]

Caso de prueba para el camino básico # 2.
Descripción: La misma descripción que el caso de prueba para el camino básico # 1

Condición de ejecución: El nombre del componente debe ser "buscar", los valores del arreglo Components deben ser ["aceptar = autentication/credential/captura/entrardato/, tiptramite","cancelar=close"].

Entrada:
compname = "buscar"
Components = ["aceptar = autentication/credential/captura/entrardato/, tiptramite","cancelar=close"]

Resultados esperados: Se espera que el arreglo con nombre "acción", contenga información sobre el componente especificado por parámetro.

Resultado obtenido: Se obtuvo el arreglo acción vacío porque el nombre del componente no se encontró en Components.

Caso de prueba para el camino básico # 3.

Descripción: La misma descripción que el caso de prueba para el camino básico # 1

Condición de ejecución: El nombre del componente debe ser “buscar”, los valores del arreglo Components deben ser [“aceptar = autentication/credential/captura/entrardato/”,”cancelar=close”].

Entrada:
compname = “buscar”
Components = [“aceptar = autentication/credential/captura/entrardato/”,”cancelar=close”]

Resultados esperados: Se espera que el arreglo con nombre “acción”, contenga información sobre el componente especificado por parámetro.

Resultado obtenido: Se obtuvo el arreglo acción vacío porque el nombre del componente no se encontró en Components.

Caso de prueba para el camino básico # 4.

Descripción: La misma descripción que el caso de prueba para el camino básico # 1

Condición de ejecución: El nombre del componente debe ser “cancelar”, los valores del arreglo Components deben ser [“aceptar = autentication/credential/captura/entrardato/”,”cancelar=close”].

Entrada:
compname = “cancelar”
Components = [“aceptar = autentication/credential/captura/entrardato/”,”cancelar=close”]

Resultados esperados: Se espera que el arreglo con nombre “acción”, contenga información sobre el componente especificado por parámetro.

Resultado obtenido: Se obtuvo el arreglo acción con el valor “close” en la primera posición y null en la segunda posición.

Caso de prueba para el camino básico # 5.

Descripción: Los datos de entrada cumplirán con los siguientes requisitos: El nombre del componente pasado por parámetro debe ser un String, Components debe ser un arreglo de String que contiene en cada posición la información referente a un componente.

Condición de ejecución: El nombre del componente debe ser “aceptar”, el arreglo Components debe estar vacío.

Entrada:
compname = “aceptar”
Components = [ ]

Resultados esperados: Se espera que el arreglo con nombre “acción”, contenga información sobre el componente especificado por parámetro.

Resultado obtenido: Se obtuvo el arreglo acción vacío porque el arreglo Components estaba vacío.

4.5 Beneficios de la aplicación

- Constituye una nueva propuesta de solución de aplicación de escritorio para la gestión de interfaces de usuario, capaz de adaptarse a los cambios en los procesos de negocio que son manejados por sistemas basados en la arquitectura orientada a servicios (SOA).
- Funciona de manera desacoplada del SEDI favoreciendo así al funcionamiento distribuido del mismo.
- Posibilita agregar interfaces de usuario sin necesidad de modificar el código de la aplicación.

4.6 Conclusiones

Se realizó el modelo de implementación de la aplicación, donde se obtuvieron los artefactos fundamentales propuestos por RUP, uno de ellos es el modelo de despliegue que permitió definir la arquitectura física del sistema y el otro es el diagrama de componentes que permitió mostrar la relación entre los componentes de la aplicación, sus dependencias, su comunicación y su ubicación.

Las pruebas de calidad son un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación, a través de su realización se logró validar y verificar la aplicación. Se realizaron pruebas de caja negra, específicamente las del tipo partición equivalente, a través de las cuales probaron cada una de las funcionalidades de la aplicación. Para complementar la validación de la aplicación se realizaron pruebas de caja blanca del tipo camino básico, las cuales permitieron ejercitar conjuntos específicos de condiciones y\o ciclos dentro de la aplicación así como examinar el estado del programa en varios puntos para determinar si el estado real era o no el esperado.

CONCLUSIONES

El presente trabajo posibilitó desarrollar una aplicación de escritorio que gestiona las interfaces de usuario separando la lógica de negocio de la presentación en el SEDI, eliminando de esta manera los problemas planteados en la situación problémica. Para la realización de la propuesta de solución, se determinó desarrollar una aplicación de escritorio sobre la plataforma J2EE, por las ventajas que esta brinda al estar orientada a software libre. Se utilizó AWT/Swing como framework de desarrollo y RUP como metodología de desarrollo. Se realizó el análisis y diseño de la aplicación logrando un acercamiento a la misma para su posterior implementación.

Con la aplicación se garantizó la gestión de interfaces de usuario en el Sistema de Emisión de Documentos de Identificación del Centro de Identificación y Seguridad Digital. Luego de realizadas las pruebas de calidad, se validó la solución y se determinó que esta cumple con las necesidades del centro.

Por todo lo anterior expuesto se concluye que los objetivos propuestos para el presente trabajo han sido cumplidos satisfactoriamente ya que la aplicación da solución a la situación problémica que le dio origen.

RECOMENDACIONES

Los objetivos generales de este trabajo han sido logrados, pero a lo largo de su desarrollo, han ido surgiendo ideas que podrían implementarse en un futuro, de forma que se logre una la aplicación más útil y efectiva, para lo cual se recomienda:

- Ampliar la biblioteca de componentes visuales de la aplicación.
- Implementar la internacionalización de la aplicación.

REFERENCIAS BIBLIOGRÁFICAS

CASTRO, R. *Propuesta de procesos para el desarrollo de un Sistema de Emisión de Documentos de Identificación.* Investigativa, Universidad de las Ciencias Informáticas, 2009.

CIBERAULA. *Java J2EE Orientado a Internet* Disponible en: http://www.ciberaula.com/curso/java2/que_es/.

CONFERENCIA#6. *Tendencias arquitectónicas* Habana: Universidad de la Ciencias Informáticas, Disponible en: http://eva.uci.cu/file.php/259/CURSO_2008-2009/Materiales_Basicos/Semana_11/Conf/Conf_6._Tendencias_arquitectonicas.pdf.

DORAL, C. *Tipos de aplicaciones* Disponible en: www.haztuprograma.com.

ERICH GAMMA, R. H., RALPH JOHNSON, JOHN VLISSIDES. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison Wesley, 1994.

GUTIÉRREZ, J. *Interfaces Gráficas de Usuario* Disponible en: http://informatica.uv.es/it3guia/LP/teoria/apuntes/cuatr2/Tema5.pdf.

IVARJACOBSON , G. B. Y. J. R. *El Proceso Unificado de Desarrollo de Software.* Madrid: Addison Wesley, 2000.

JAMES RUMBAUGH, I. J., GRADY BOOCH. *El Proceso Unificado de Desarrollo de Software.* Madrid: Addison Wesley, 2000.

JMATTER. *JMatter* Disponible en: http://www.jmatter.org/.

LANDRIAN, J. *Framework Común v2.0* La Habana: Universidad de las Ciencias Informáticas, 2008,

LARMAN, C. *UML y Patrones. Introducción al análisis y diseño orientado a objetos* México: Prentice Hall 1999.

LARSON, J. *Interactive software.* Englewood Cliffs, New Jersey: Yourdon Press, 1992.

MOLPECERES, A. *Diseño de Software con Patrones (parte 3)* Disponible en: http://www.javahispano.org/contenidos/es/patrones_de_software_parte_3/.

MOREA, L. *Introducción a Java* Disponible en: http://www.monografias.com/trabajos/java/java.shtml.

NETBEANS. *NetBeans Platform Learning Trail* Disponible en: http://www.netbeans.org/kb/trails/platform.html.

OPENXAVA. *OpenXava* Disponible en: http://www.gestion400.com/web/guest/home.

OPTIC. *Normas y Estándares de Portales Gubernamentales* República Dominicana: Gobierno Electrónico Disponible en:

http://www.sap.gob.do/LinkClick.aspx?fileticket=NuzJ%2Bqs48Hg%3D&tabid=63&mid=420&language=es-ES.

ORBEON. *Orbeon Forms User Guide* Disponible en: http://www.orbeon.com/ops/doc/home-faq#xforms.

PAEZ, N. Inyección de dependencias y contenedores livianos en .NET Disponible en: http://web.fi.uba.ar/~npaez/content/ioc_frameworks_v1.pdf.

PELAEZ, J. C. *Arquitectura basada en Componentes* Disponible en: http://geeks.ms/blogs/jkpelaez/archive/2009/04/18/arquitectura-basada-en-componentes.aspx.

QTJAMBI. *Qt Jambi Reference Documentation* Disponible en: http://doc.trolltech.com/qtjambi-4.4/html/com/trolltech/qt/qtjambi-index.html.

REYNOSO, C. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft* 1. ed. Buenos Aires:

RÍOS, S. S. *Análisis y UML. Modelo conceptual* Chile: Universidad Viña del Mar, Disponible en: http://www.uvmsf.cl/~ssanchez/images/Metodologias/Unidad4_MAD.pdf.

RUIZ, D. A. D. *Software libre* Disponible en: http://www.monografias.com/trabajos33/software-libre/software-libre.shtml.

SCHEMULLER, J. *Aprendiendo UML en 24 horas.* México: Pentrice Hall, 2000.

SHNEIDERMAN, B. *Designing The user interface, Strategies for effective Human-computer interaction.* Tercera ed. Massachusetts: Addison-Wesley, 1998,

SPARXSYSTEMS. *Guía de Usuario de Enterprise Architect 7.0: Diagrama del Paquete* Disponible en: http://www.sparxsystems.com.ar/download/ayuda/index.html?packagediagram.htm.

BIBLIOGRAFÍA

ALARCÓN, R. Diseño Orientado a Objetos con UML. Madrid, España: 2000.

ALEJANDRO ALVAREZ B, E. V. R. Generadores de Interfaces de Usuario: QT Designer, NetBeans y Windows Forms Designer de 2009]. Disponible en: http://www.di-mare.com/adolfo/cursos/2007-1/pp-GenGUI.pdf.

CASANOVAS, J. Usabilidad y arquitectura del software Disponible en: http://www.desarrolloweb.com/articulos/1622.php.

COLONIA, B. Capítulo 2. Análisis. Razones para determinación de metodología RUP. 2009, Disponible en: http://blog.pucp.edu.pe/category/5169/blogid/1627?page=2.

DANIELE, M. El arte de modelar Disponible en: http://dc.exa.unrc.edu.ar/nuevodc/materias/sistemas/2007/TEORICOS/TEORIA_7_UML_DI_2007.pdf.

DYKES, L. XML for Dumies Cuarta. ed. Wiley Publishing, Disponible en: http://bibliodoc.uci.cu/pdf/978-0-7645-8845-7.pdf.

ENGLANDER, R. Java and SOAP Disponible en: http://bibliodoc.uci.cu/pdf/0596001754.pdf.

FROUFE, A. Tutorial de Java. Tabla de Contenido Disponible en: http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/introduccion/indice.html

GONZÁLEZ, C. Patrón de Inyección de dependencias Disponible en: http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=DependencyInjector#_Toc186529749.

IVARJACOBSON , G. B. Y. J. R. El Proceso Unificado de Desarrollo de Software. Madrid: Addison Wesley, 2000.

LAGO, R. Patrones de diseño software Disponible en: http://www.proactiva-calidad.com/java/patrones/index.html.

LARMAN, C. UML y Patrones. Introducción al análisis y diseño orientado a objetos México: Prentice Hall 1999.

MARTIN FOWLER, D. R. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.

MICROSOFT. Application Architecture Guide 2.0. Designing Applications on the .NET Platform. 2008.

PRESSMAN, R. Ingeniería del Software: Un enfoque práctico. Quinta ed. La Habana: Félix Varela, 2002.

REUSSNER, R. Enhanced Component Interfaces to Support Dynamic Adaption and Extension Disponible en: http://doi.ieeecomputersociety.org/10.1109/HICSS.2001.927238.

BIBLIOGRAFÍA

REYNOSO, B. Architect Academy: Seminario de Arquitectura de Software Disponible en: http://download.microsoft.com/download/5/6/8/568d7ce9-d0ca-46d6-a6cb-8b53cf10b134/050608-Architect%20Academy%20Webcast%201.ppt.

SÁNCHEZ, J. Un método para validar requisitos y generar interfaces de usuario multiplataforma Valencia, España: Universidad Politécnica de Valencia, Disponible en: http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER02/sanchez.pdf.

SUEZ, E. Building Software Applications with JMatter Disponible en: http://www.jmatter.org/documentation/html/index.html.

TOROSSI, G. El Proceso Unificado de Desarrollo de Software Disponible en: http://www.chaco.gov.ar/UTN/disenodesistemas/apuntes/oo/ApunteRUP.pdf.

YOUNG, R. R. The Requirements Engineering Handbook Disponible en: http://bibliodoc.uci.cu/pdf/1-58053-266-7.pdf.

ZHANG, Q. Reengineering User Interfaces of E-Commerce Applications Using Business Processes Disponible en: http://doi.ieeecomputersociety.org/10.1109/ICSM.2006.51.

GLOSARIO

- **AFIS:** Sistema Automático de Identificación de Huellas Dactilares.
- **API**: Application Programming Interface (Interfaz de Programación de Aplicaciones). **C**onjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.
- **ClassPath:** La variable CLASSPATH puede incluir la ruta de directorios o ficheros. Determina dónde buscar tanto las clases o librerías de Java (el API de Java) como otras clases de usuario. Indica al JDK dónde debe buscar los archivos a compilar o ejecutar, sin tener que escribir en cada ejecución la ruta completa.
- **CRUD:** Es el acrónimo de Crear, Obtener, Actualizar y Borrar (Create, Read, Update y Delete en inglés). Es usado para referirse a las funciones básicas en bases de datos o la capa de persistencia en un sistema de software.
- **Framework**: Estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje de *scripting* entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.
- **FTP**: File Transfer Protocol. Protocolo de transferencia de ficheros entre sistemas conectados a una red TCP basado en la arquitectura cliente-servidor, de manera que desde un equipo cliente nos podemos conectar a un servidor para descargar ficheros desde él o para enviarle nuestros propios archivos independientemente del sistema operativo utilizado en cada equipo.
- **Hardware:** Conjuntos de componentes que integran la parte material de una computadora. Componentes físicos de una tecnología.
- **Http:** Protocolo de Transmisión Hipertexto. Protocolo de comunicaciones utilizado por los programas clientes y servidores de WWW para intercambiar archivos (texto, gráfica, imágenes, sonido, video y otros archivos multimedia).

GLOSARIO

- **IFC:** Internet Foundation Classes de Netscape son una biblioteca de clases de Java que los desarrolladores pueden utilizar en crear programas Java.
- **Java Script:** Es un lenguaje interpretado orientado a las páginas web, para realizar tareas y operaciones en el marco de la aplicación cliente.
- **Look & Feel (Parecer y Percepción):** Es la apariencia que se proporciona a los diferentes componentes de Swing en Java.
- **PDF:** ***P****ortable* ***D****ocument* ***F****ormat*, (formato de documento portátil) es un formato de almacenamiento de documentos.
- **Pipeline:** Es el proceso según el cual, mientras una instrucción es ejecutada, otra está siendo interpretada por el ordenador y una más está siendo leída.
- **Plugin**: Aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica, generalmente muy específica.
- **Protocolo**: Conjunto de reglas que controlan la secuencia de mensajes que ocurren durante una comunicación entre entidades que forman una red.
- **Ruby on Rails**: También conocido como **RoR** o **Rails** es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC).
- **Sagem:** Empresa Francesa de servicios de análisis de sistemas y procesamiento informático.
- **Script**: Es un guión o conjunto de instrucciones que permiten la automatización de tareas creando pequeñas utilidades.
- **SGML:** ("Standard Generalized Markup Language" o "Lenguaje de Marcación Generalizado"), consiste en un sistema para la organización y etiquetado de documentos.
- **Software**: Programas de sistema, utilerías o aplicaciones expresados en un lenguaje de máquina.

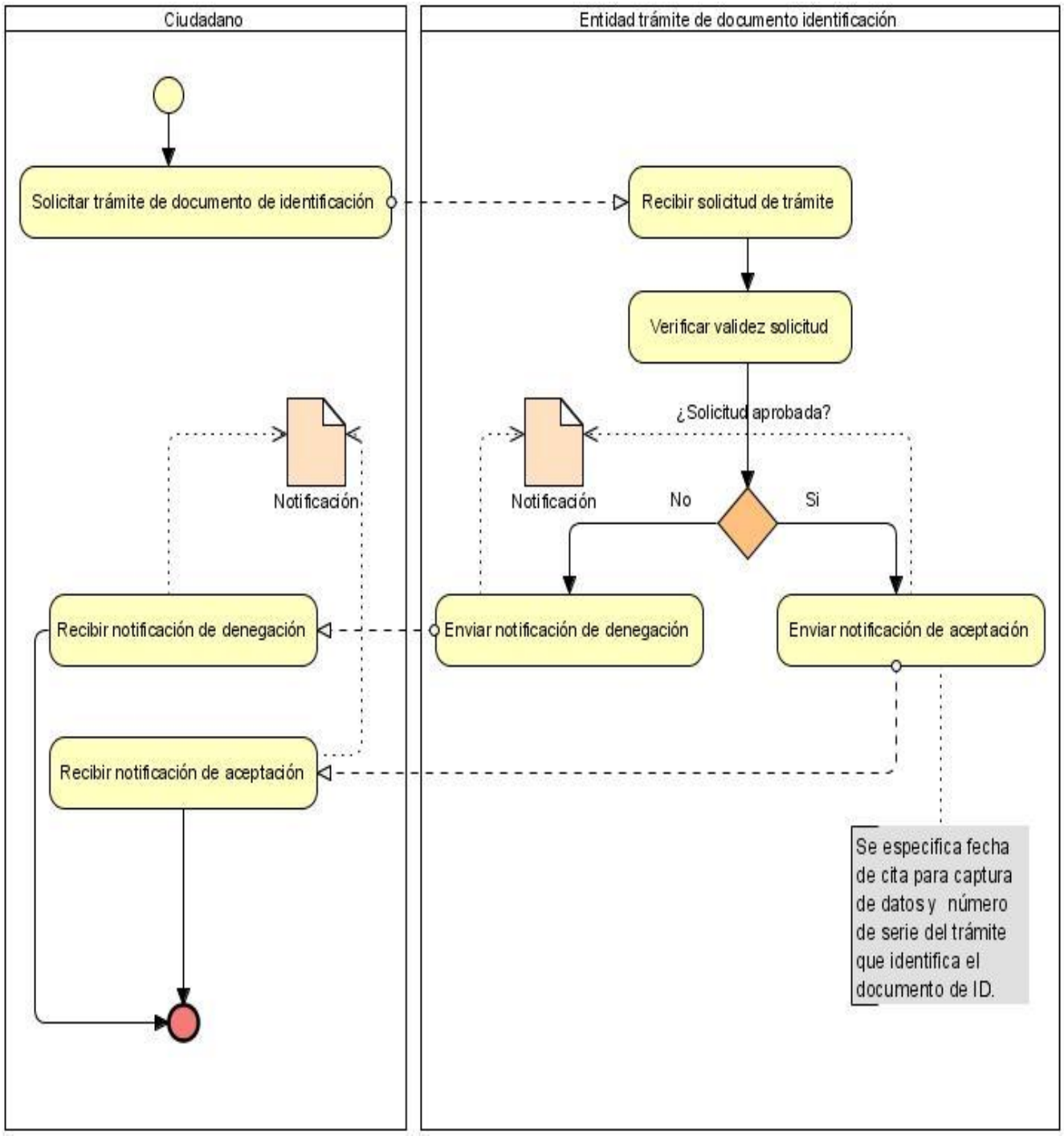
GLOSARIO

- **Spring Framework**: También conocido simplemente como Spring es un Framework Open Source (código abierto) de desarrollo de aplicaciones para la plataforma Java.
- **TCP/IP:** Protocolo de Control de Transmisión (TCP) y Protocolo de Internet (IP), se utiliza a nivel mundial para conectarse a Internet y a los servidores web.
- **WEB (WWW):** Red de documentos HTML intercomunicados y distribuidos entre servidores del mundo entero.
- **W3C:** Es un consorcio internacional donde las organizaciones miembro, personal a tiempo completo y el público en general, trabajan conjuntamente para desarrollar estándares Web.
- **XHTML**: e**X**tensible **H**yper**t**ext **M**arkup **L**anguage (lenguaje extensible de marcado de hipertexto), es el lenguaje pensado para sustituir a HTML como estándar para las páginas web.
- **XPL:** Es un dialecto del lenguaje de programación PL/1.

ANEXOS

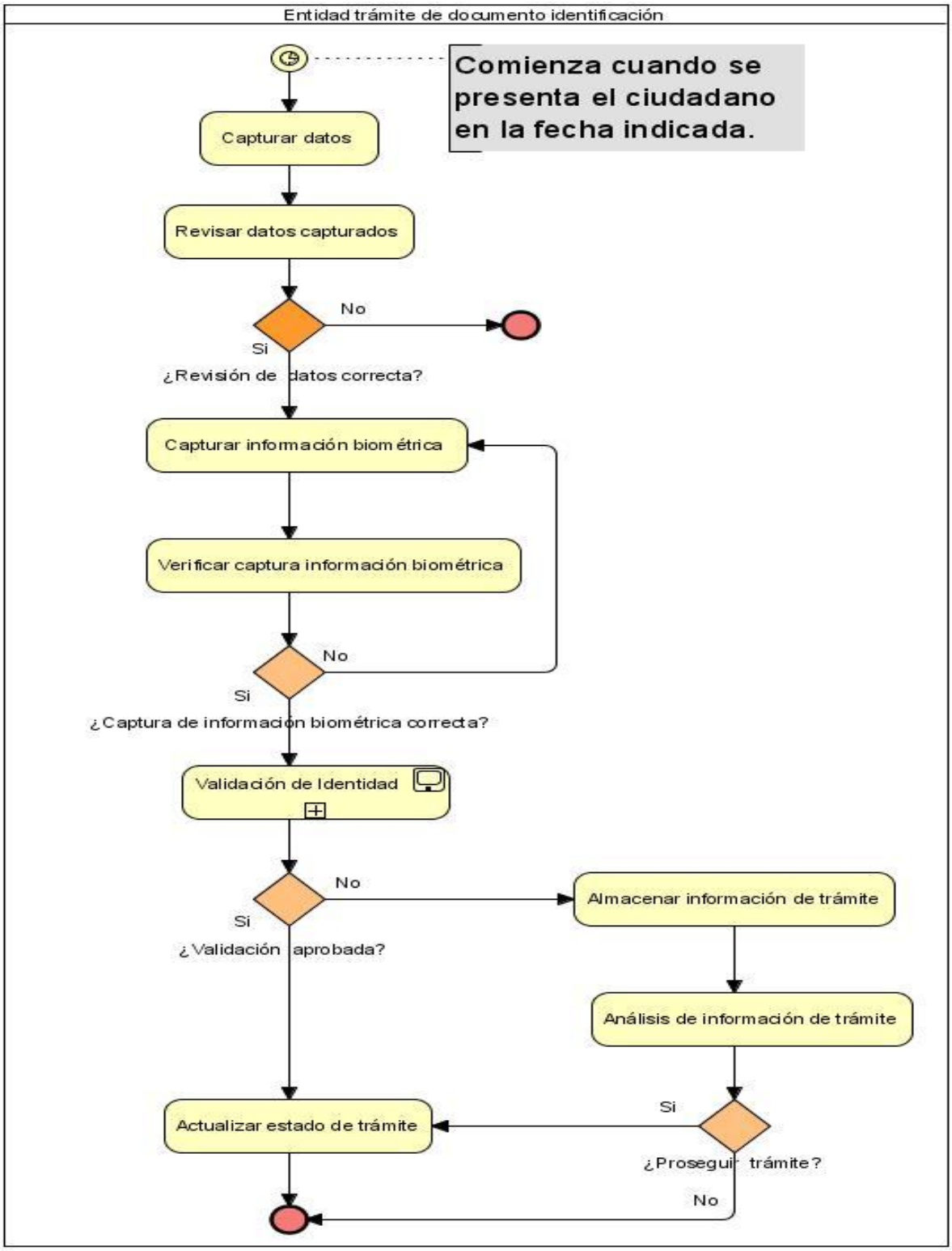
Anexo 1: Diagramas de los procesos de emisión de documentos de identificación

BPD Proceso Solicitud de Documentos de Identificación



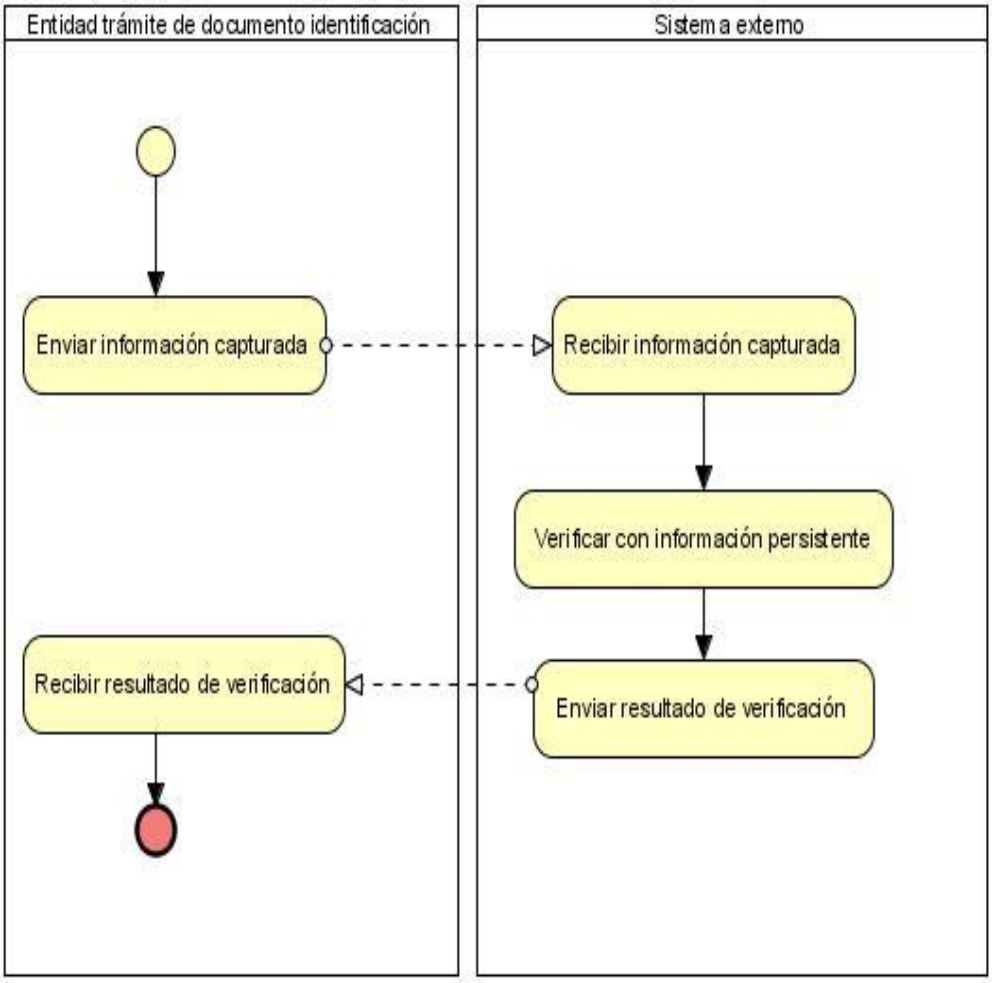
Anexo 1 Fig. 1 BPD Proceso Solicitud de Documentos de Identificación.

BPD Proceso Enrolamiento de Datos



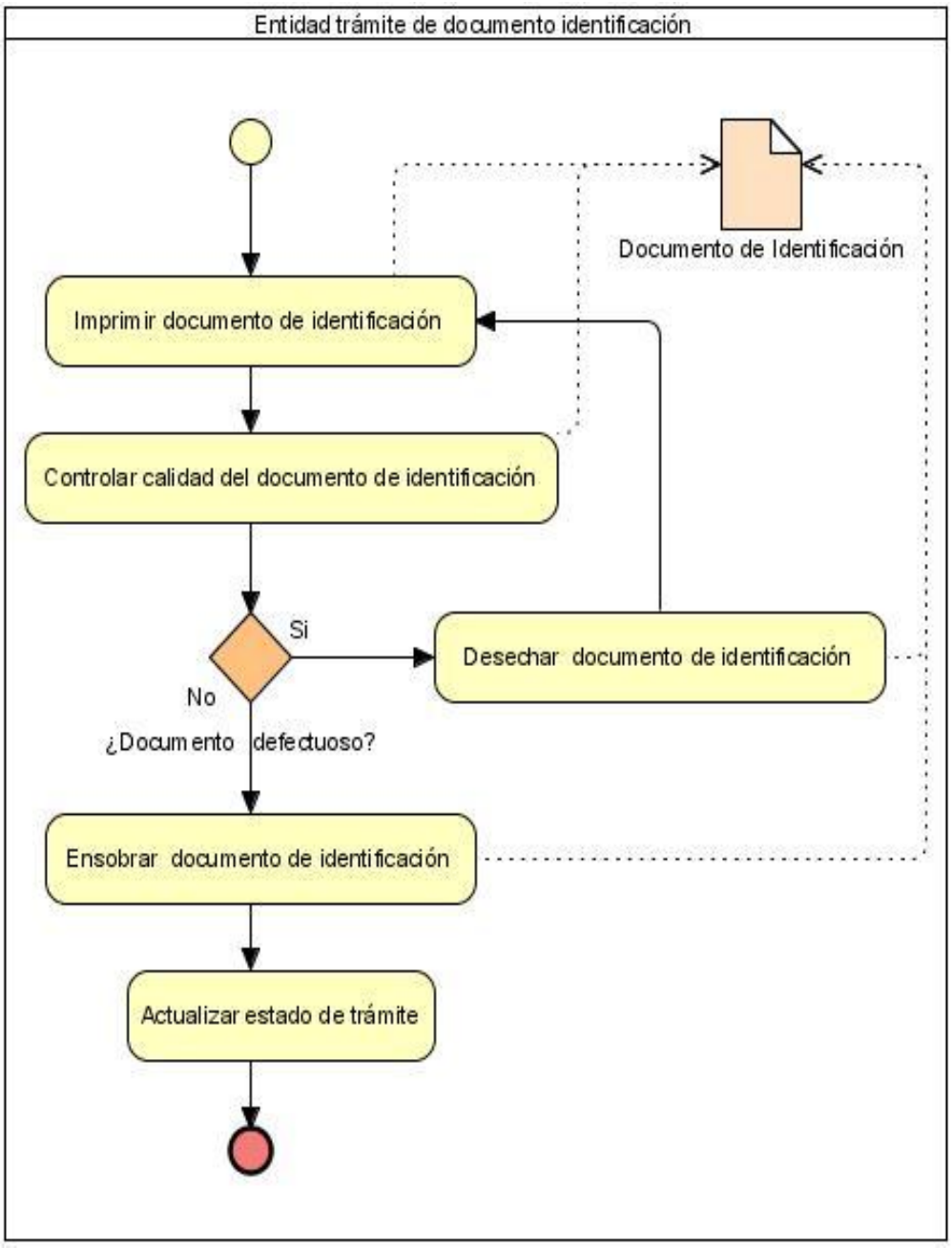
Anexo 1 Fig. 2 BPD Proceso Enrolamiento de Datos.

BPD Subproceso Validación de Identidad



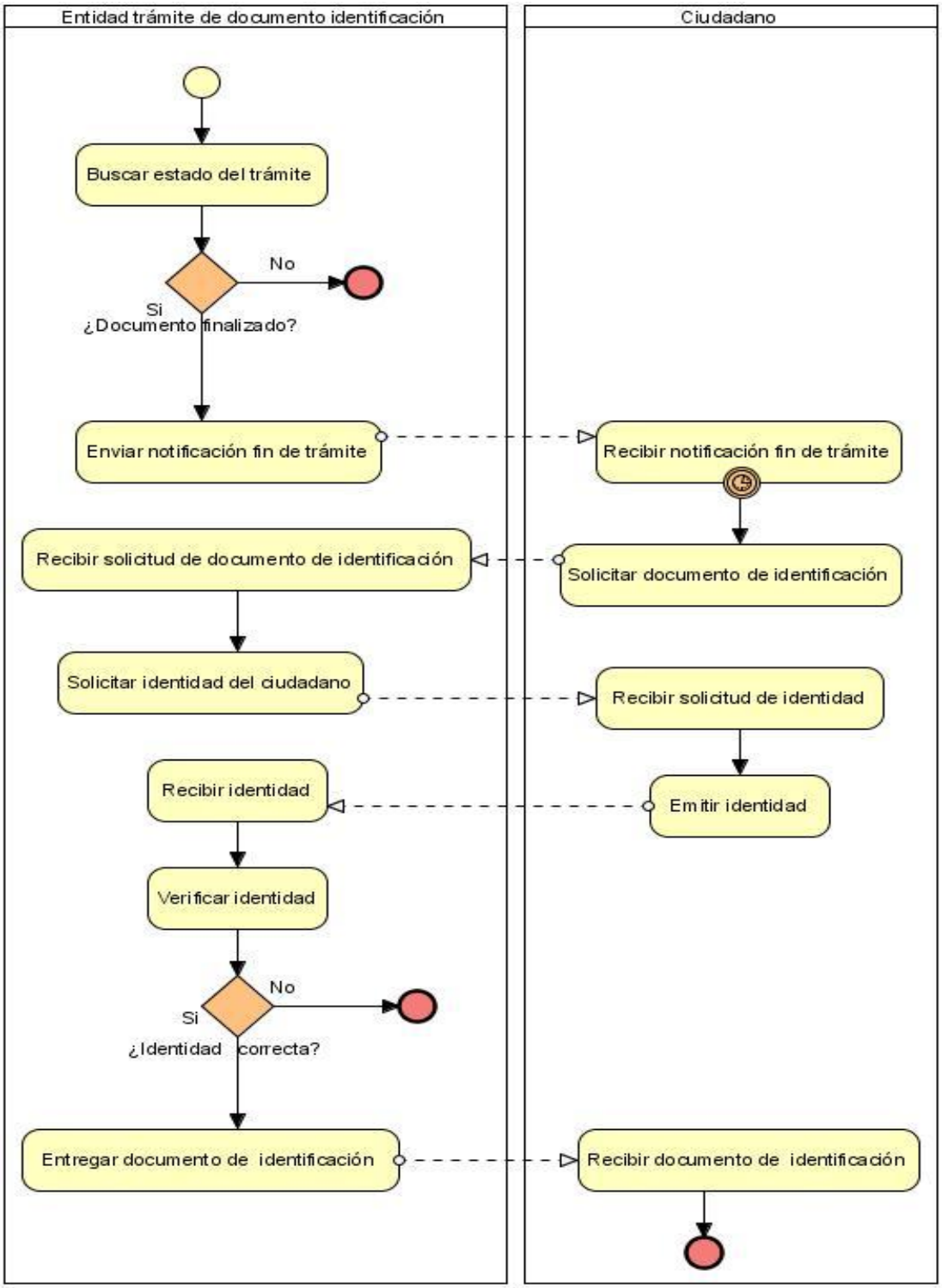
Anexo 1 Fig. 3 BPD Subproceso Validación de Identidad.

BPD Proceso Personalización del Documento de Identificación



Anexo 1 Fig. 4 BPD Proceso Personalización del Documento de Identificación.

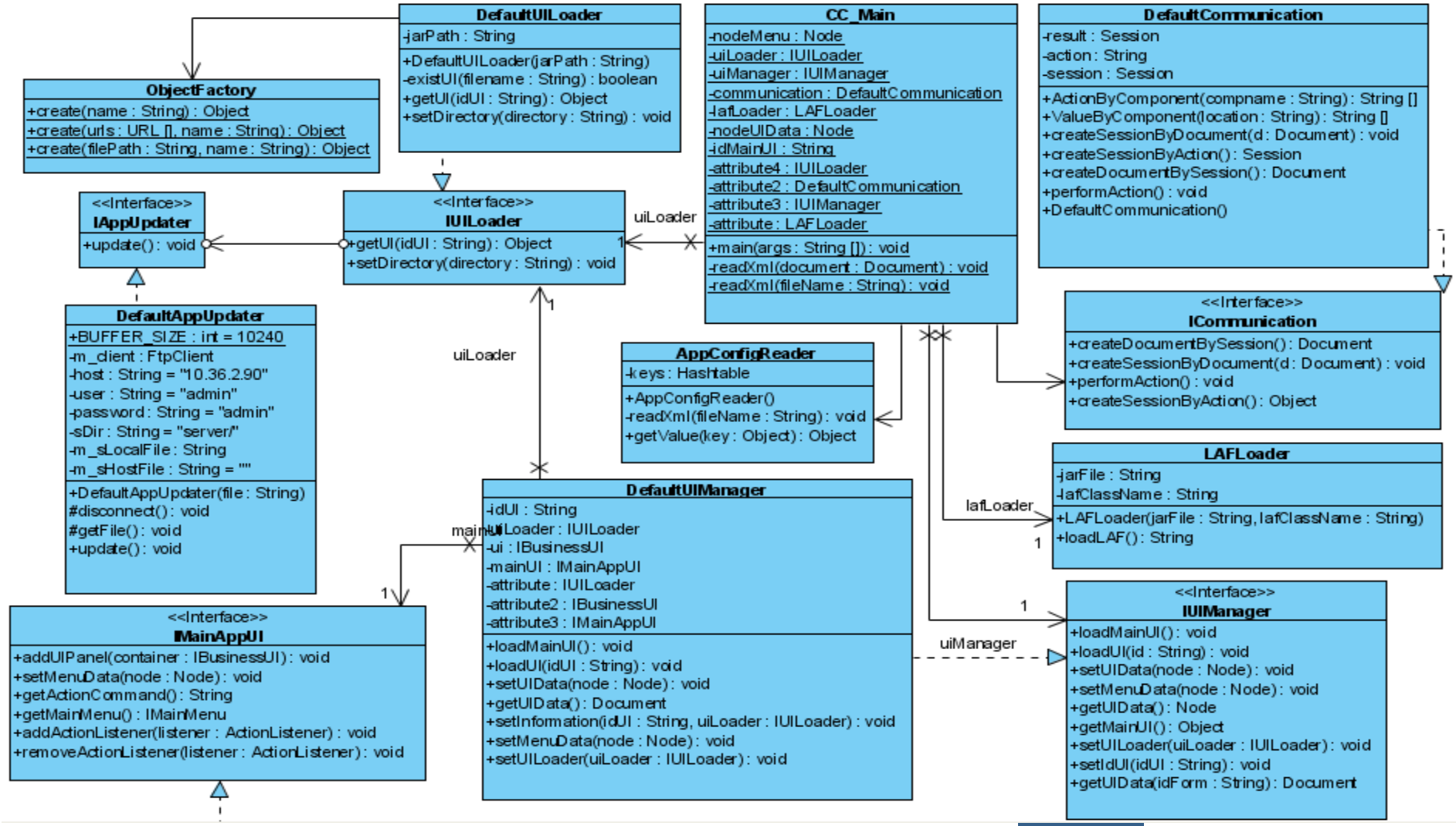
BPD Proceso Entrega Documento de Identificación

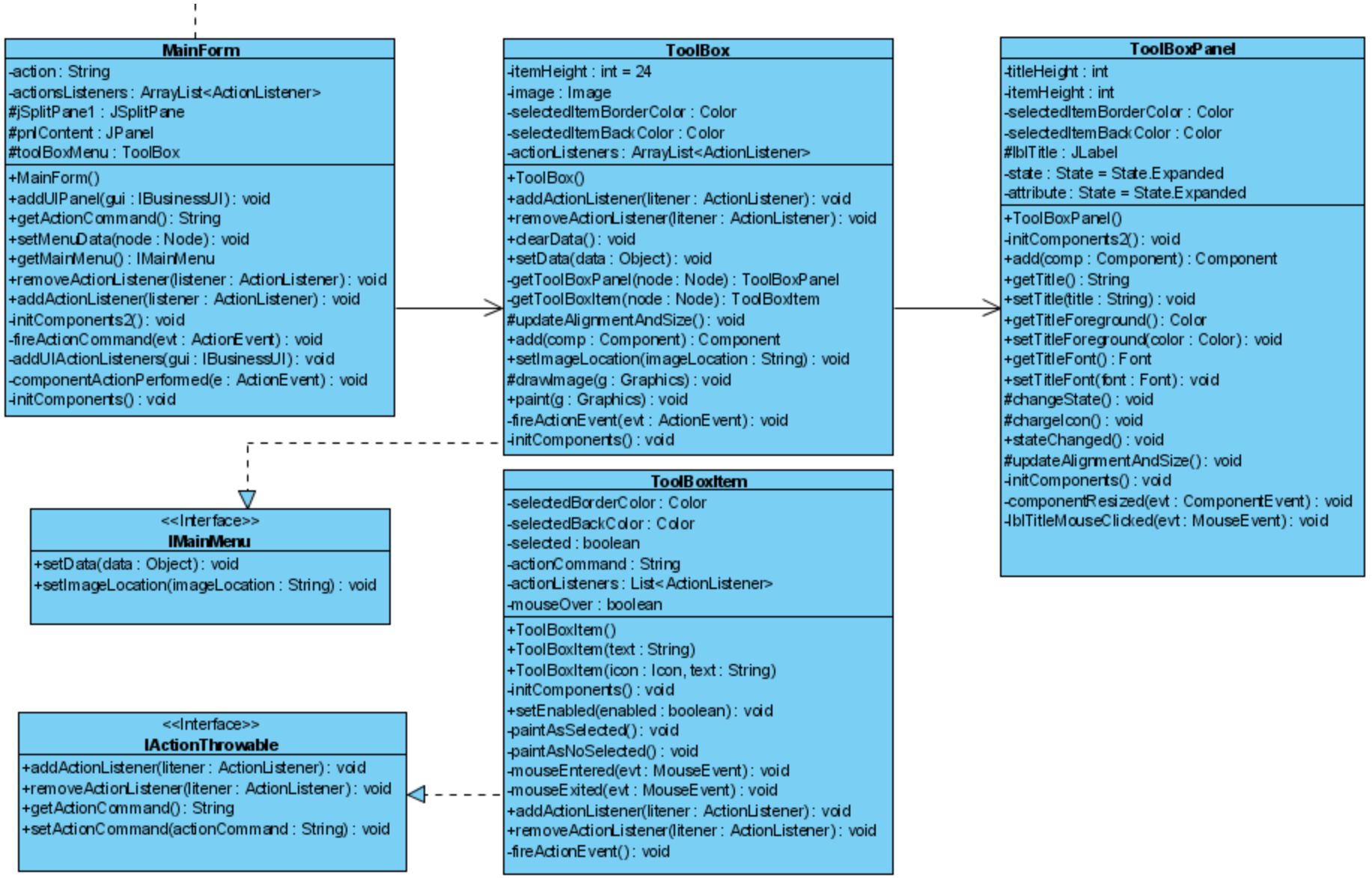


Anexo 1 Fig. 5 BPD Proceso Entrega Documento de Identificación.

Anexo 2: Diagramas de clases del diseño

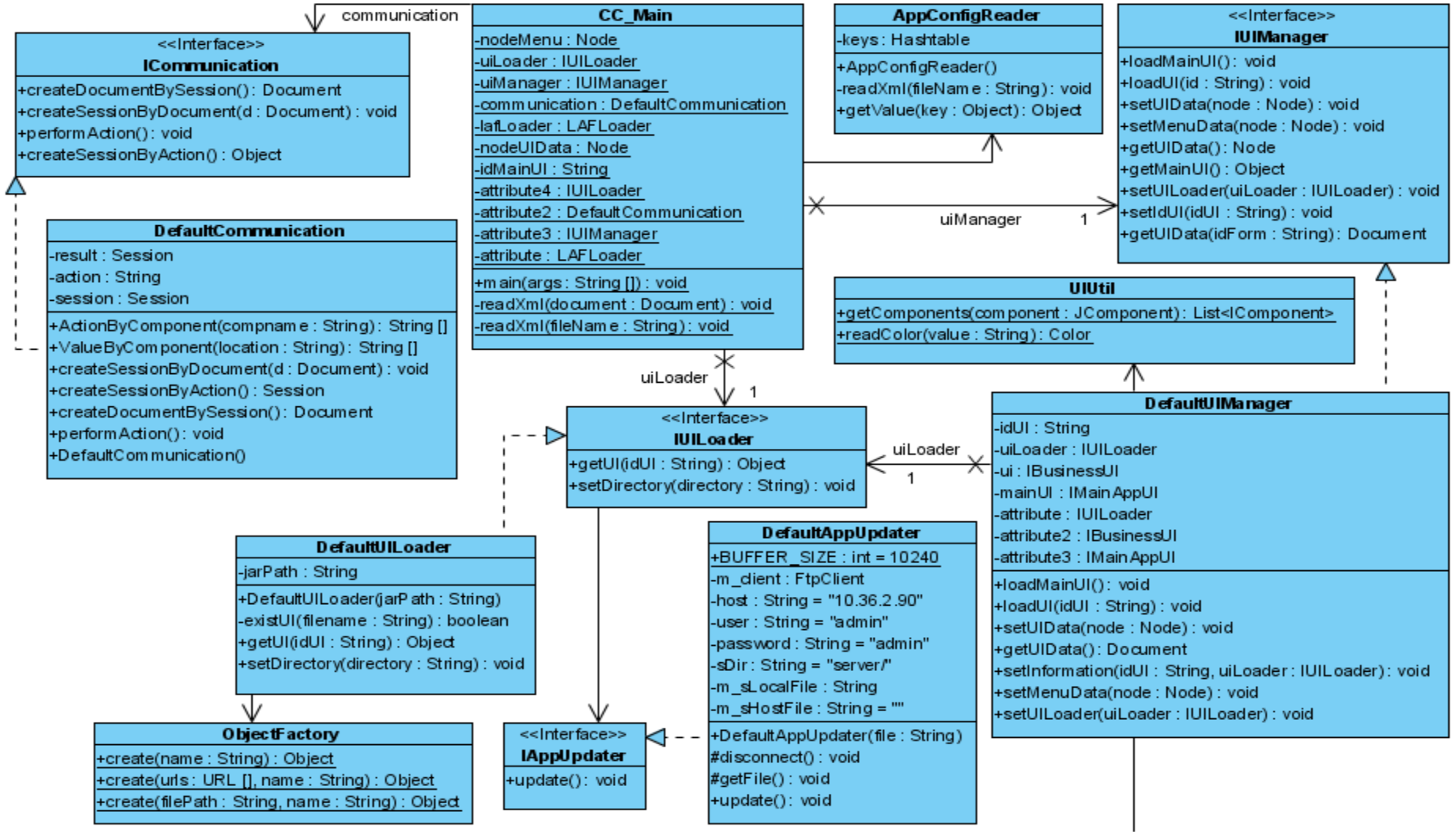
Diagrama de clases del diseño del CU Cargar Plantilla

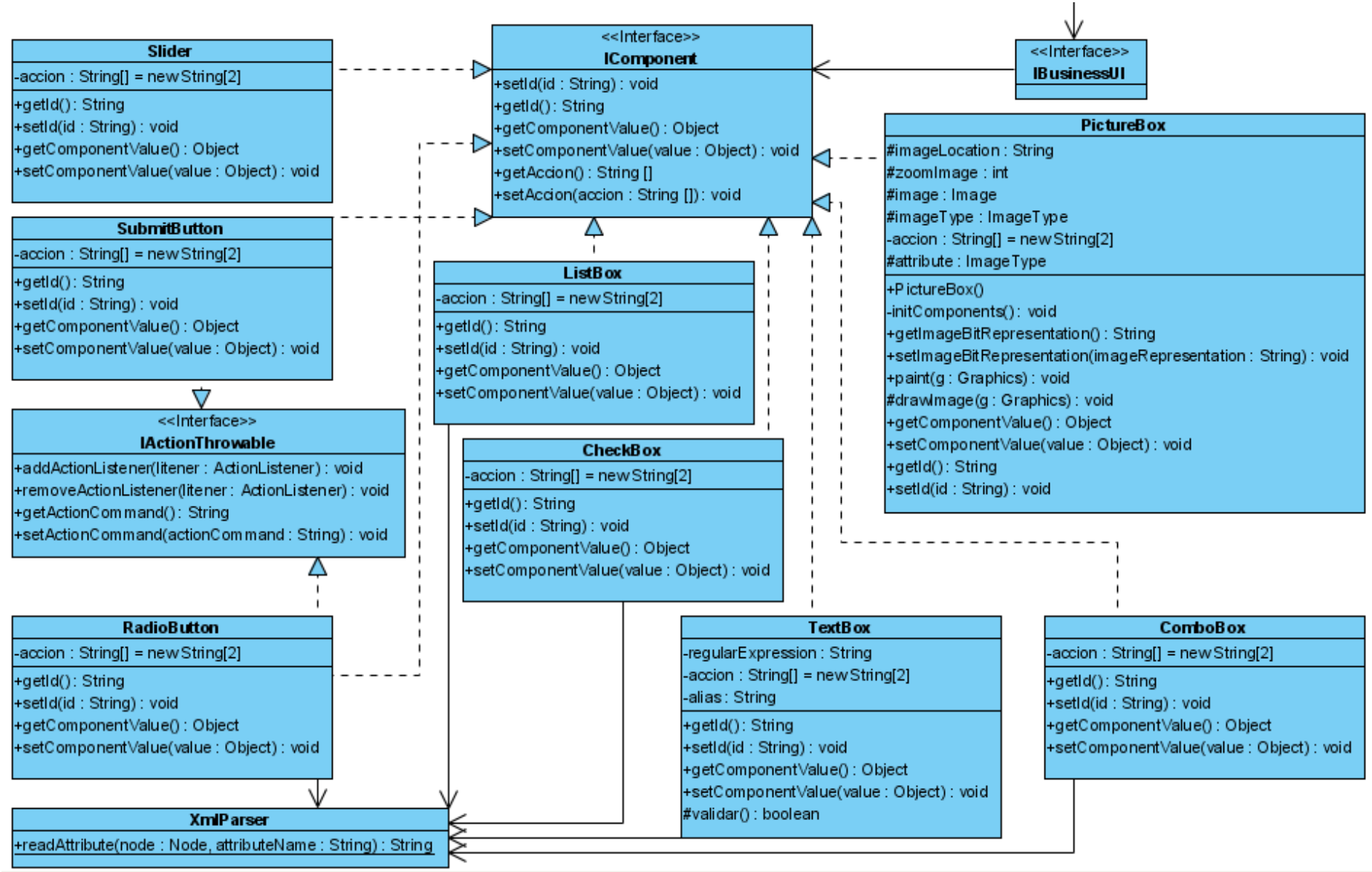




Anexo 2 Fig. 1 Diagrama de clases del diseño del CU Cargar Plantilla.

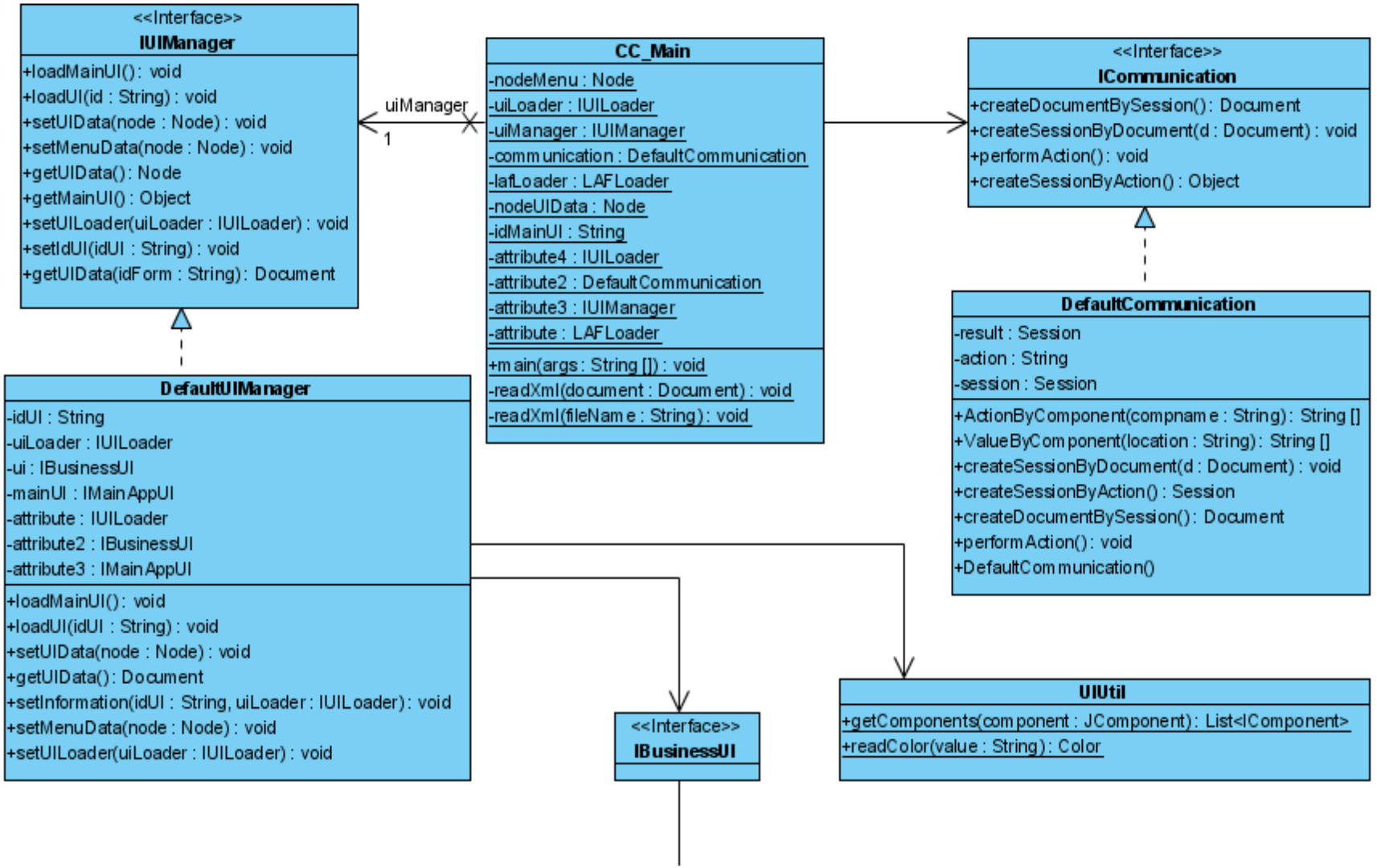
Diagrama de clases del diseño del CU Cargar Formulario

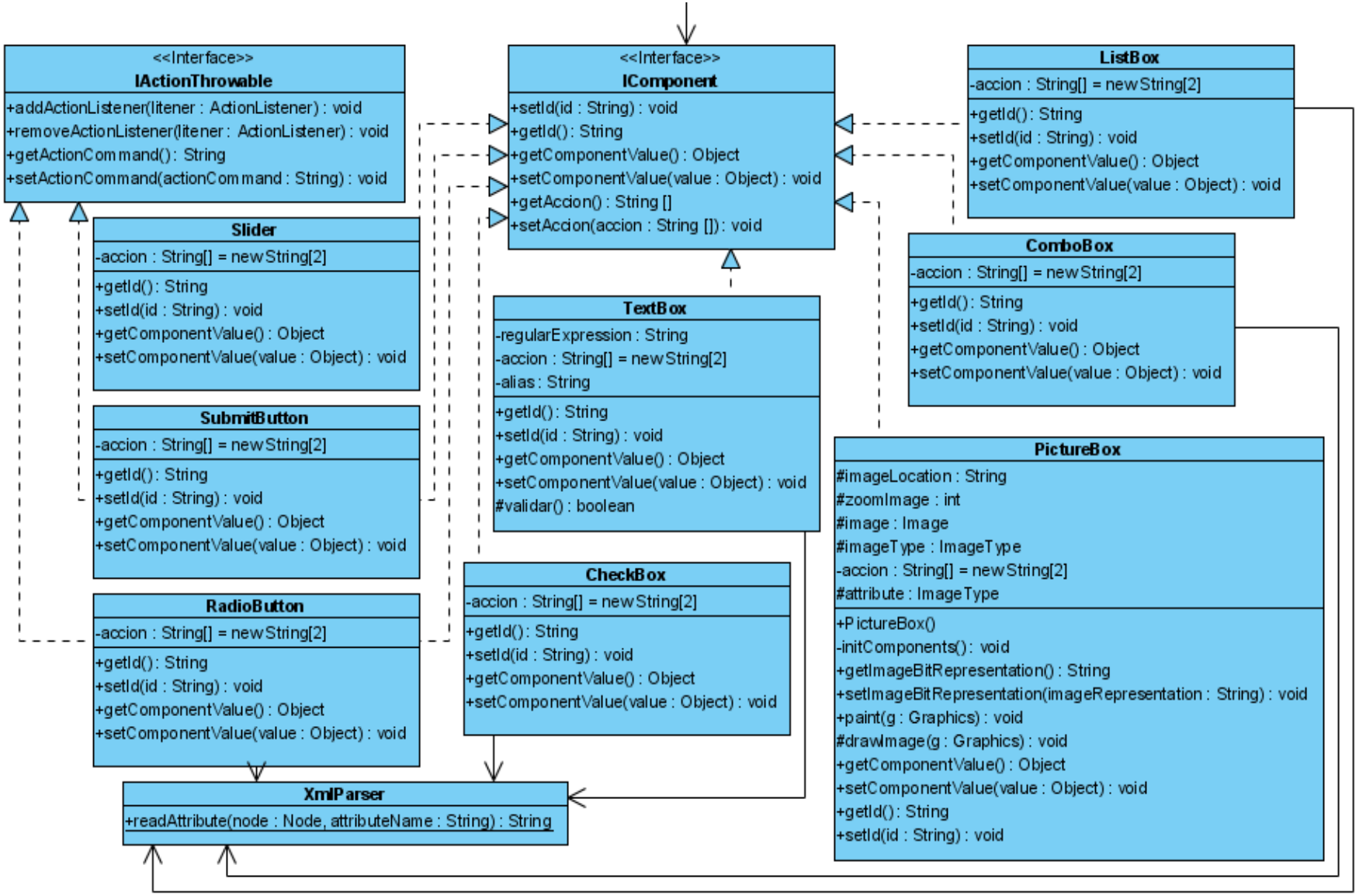




Anexo 2 Fig. 2 Diagrama de clases del diseño del CU Cargar Formulario.

Diagrama de clases del diseño del CU Capturar Datos

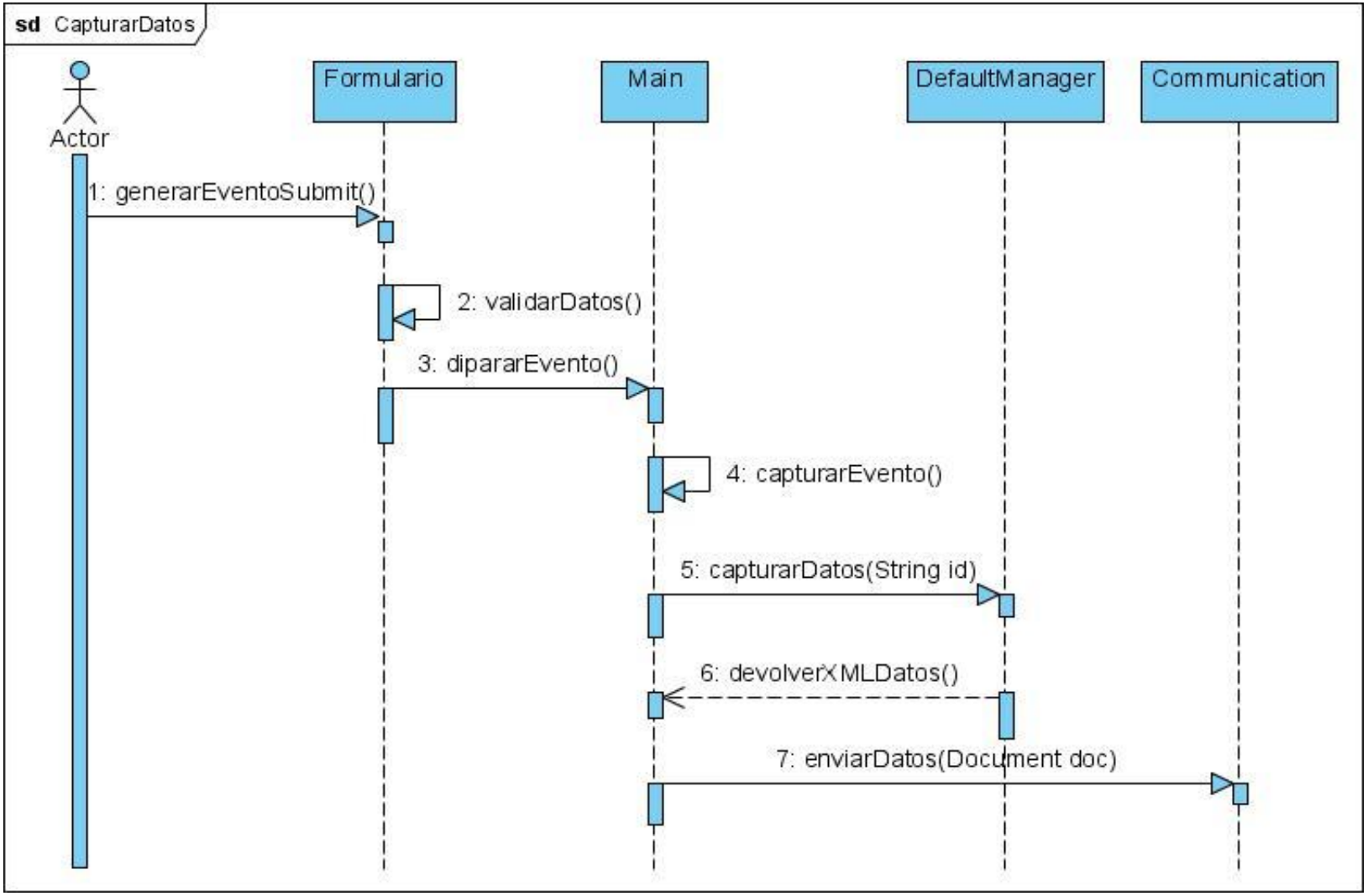




Anexo 2 Fig. 3 Diagrama de clases del diseño del CU Capturar Datos.

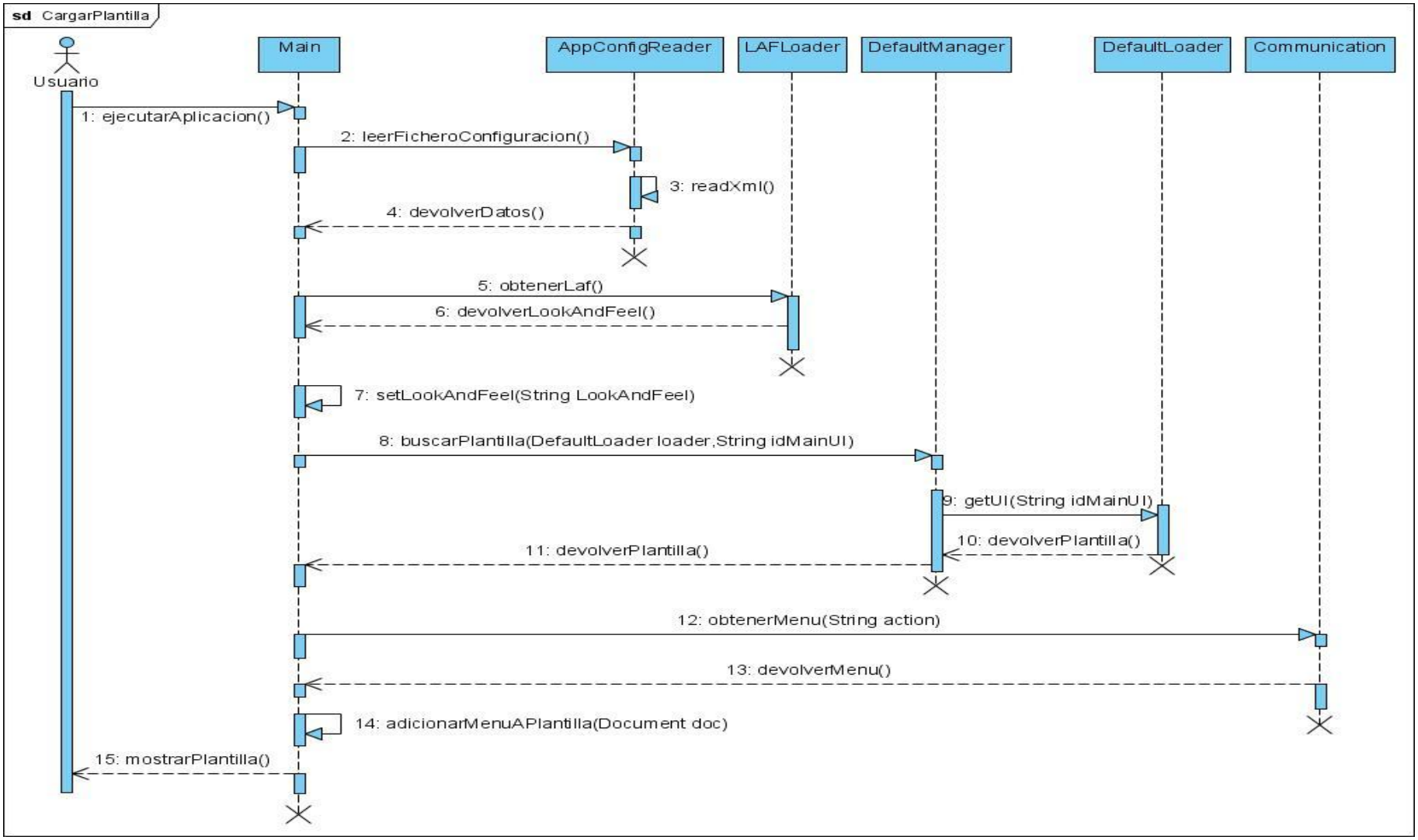
Anexo 3: Diagramas de Secuencia

Diagrama de secuencia del diseño del CU Capturar Datos



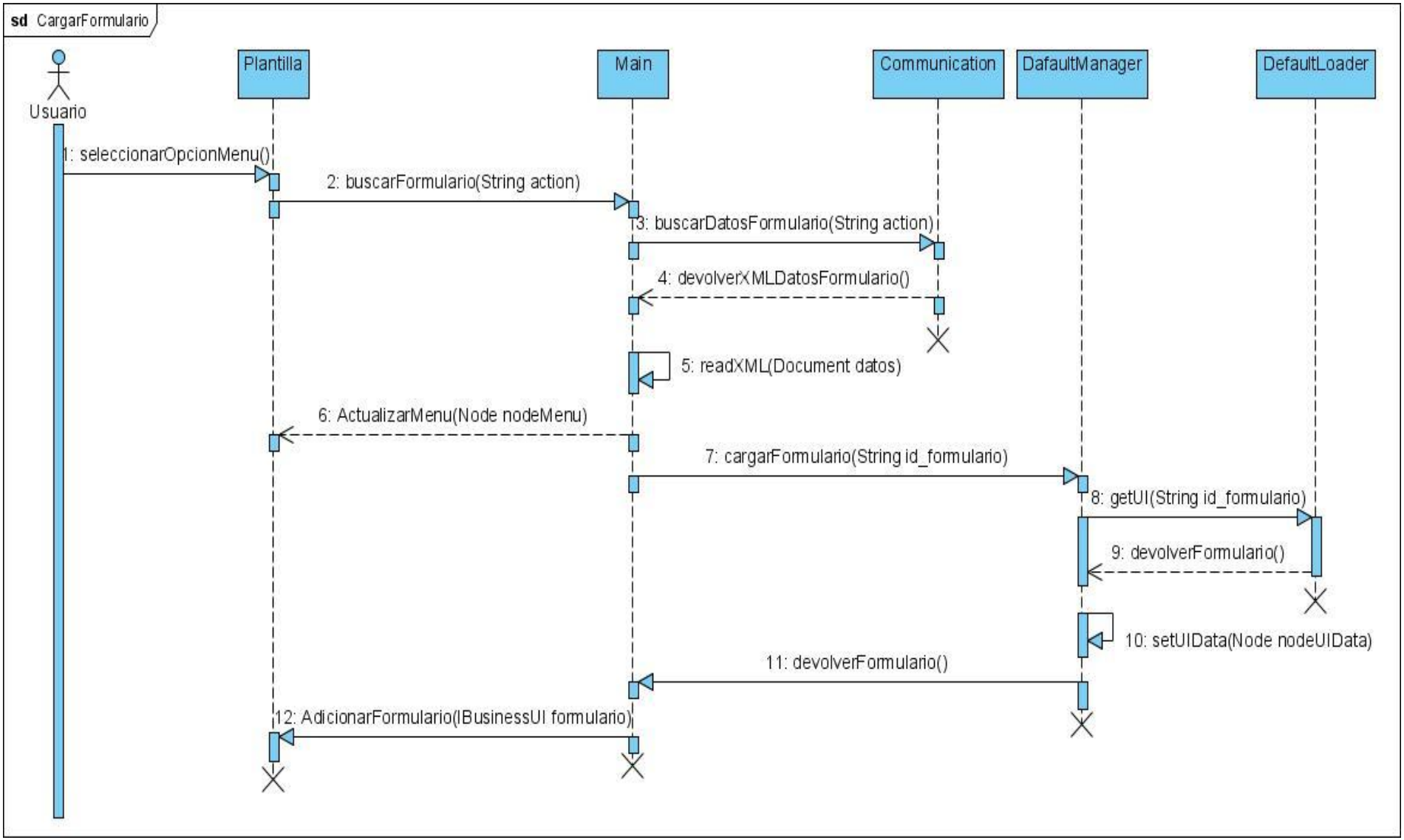
Anexo 3 Fig. 1 Diagrama de secuencia del diseño del CU Capturar Datos.

Diagrama de secuencia del diseño del CU Cargar Plantilla



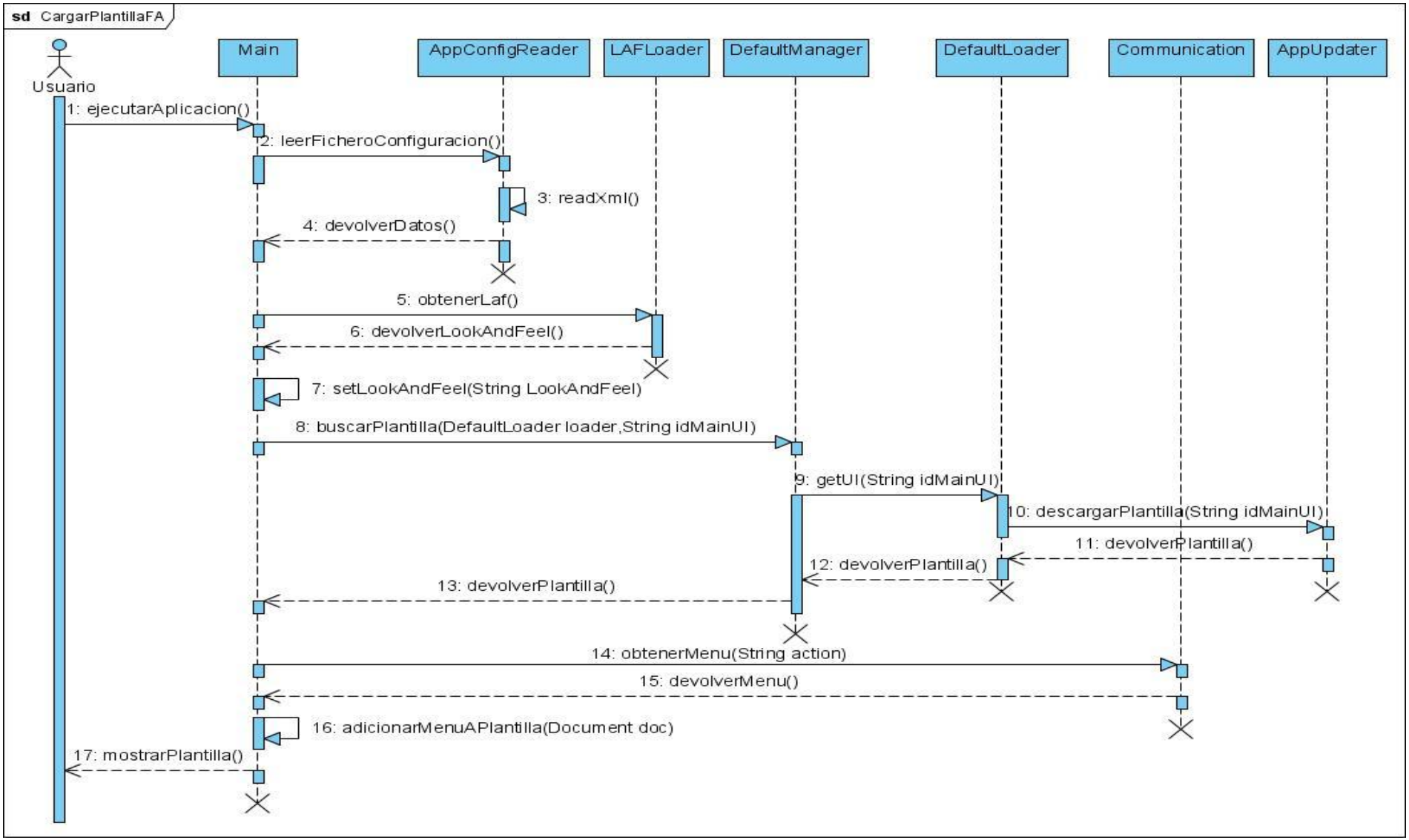
Anexo 3 Fig. 2 Diagrama de secuencia del diseño del CU Cargar Plantilla.

Diagrama de secuencia del diseño del CU Cargar Formulario



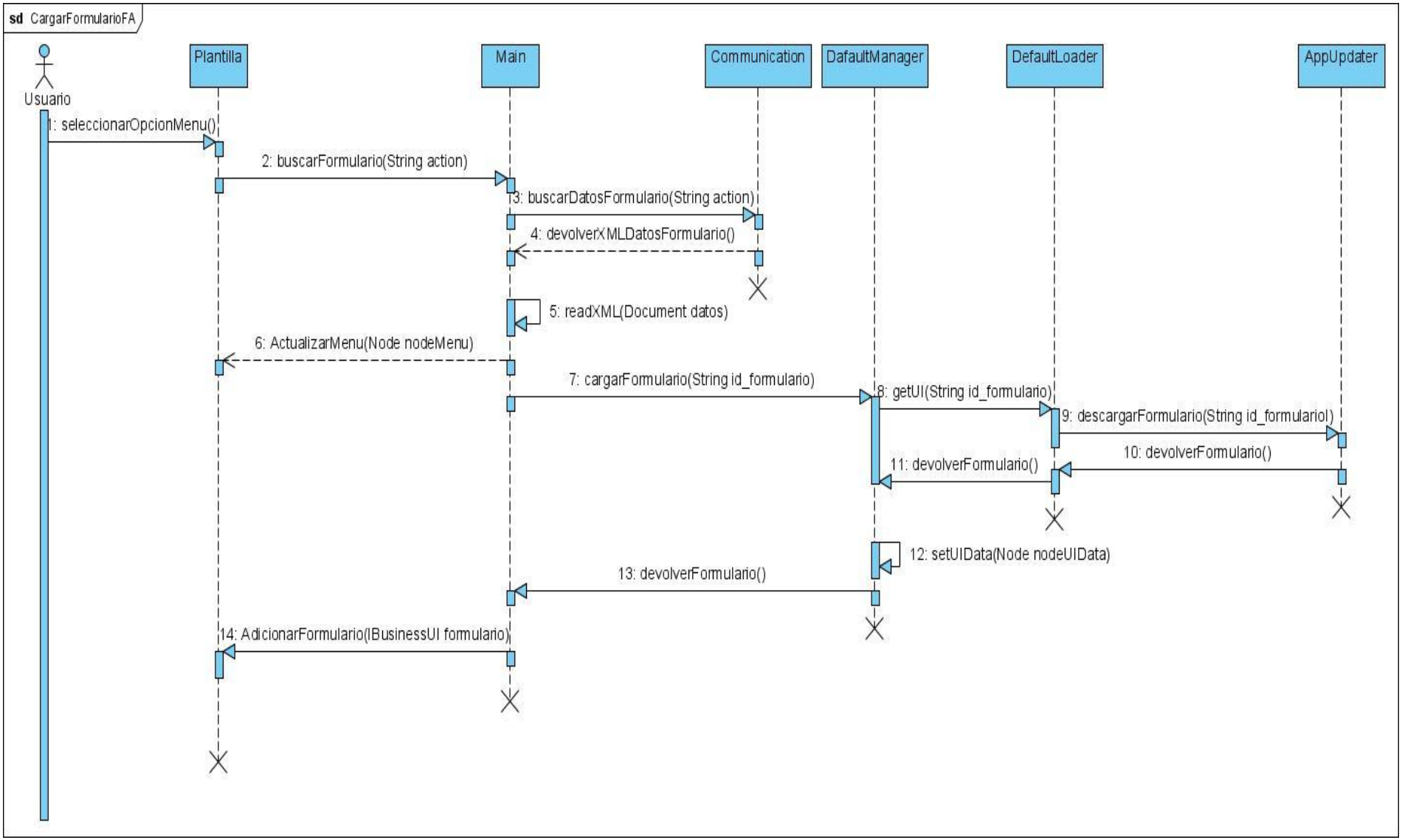
Anexo 3 Fig. 3 Diagrama de secuencia del diseño del CU Cargar Formulario.

Diagrama de secuencia del diseño del Flujo Alterno del CU Cargar Plantilla



Anexo 3 Fig. 4 Diagrama de secuencia del diseño del Flujo Alterno del CU Cargar Plantilla.

Diagrama de secuencia del diseño del Flujo Alterno del CU Cargar Formulario



Anexo 3 Fig. 5 Diagrama de secuencia del diseño del Flujo Alterno del CU Cargar Formulario.

Anexo 4: Descripción de las clases del diseño

Nombre: XMLConfigReader	
Tipo de clase : controladora	
Atributo	**Tipo**
mainInterfaceId	String
interfacesPath	String
Para cada responsabilidad:	
Nombre:	XMLConfigReader(String fileName)
Descripción:	Constructor al cual se la pasa como parámetro la dirección de un fichero de configuración que es usado para llamar al método readXML el cual inicializa los atributos de la clase.
Nombre:	readXml(String fileName) : void
Descripción:	Interpreta el XML de configuración especificado como parámetro, adicionándole al hashtable los pares llave-valor, con la información del XML.
Nombre:	getMainInterfaceId() : String
Descripción:	Devuelve el atributo *mainInterfaceId* de la clase.
Nombre:	getInterfacesPath() : String
Descripción:	Devuelve el atributo *interfacesPath* de la clase.

Anexo 4 Tabla 1. Descripción de la clase “XMLConfigReader”.

Nombre: UIUtil	
Tipo de clase : controladora	
Atributo	**Tipo**
-	-
Para cada responsabilidad:	
Nombre:	getComponents(JComponent component): List<IComponent>
Descripción:	Devuelve listado de los componentes de un formulario especificado por parámetro
Nombre:	readColor(String value):Color
Descripción:	Construye y retorna un objeto Color, a partir del parámetro especificado, el cual contiene los elementos que conforman a un objeto de este tipo.

Anexo 4 Tabla 2. Descripción de la clase “UIUtil”.

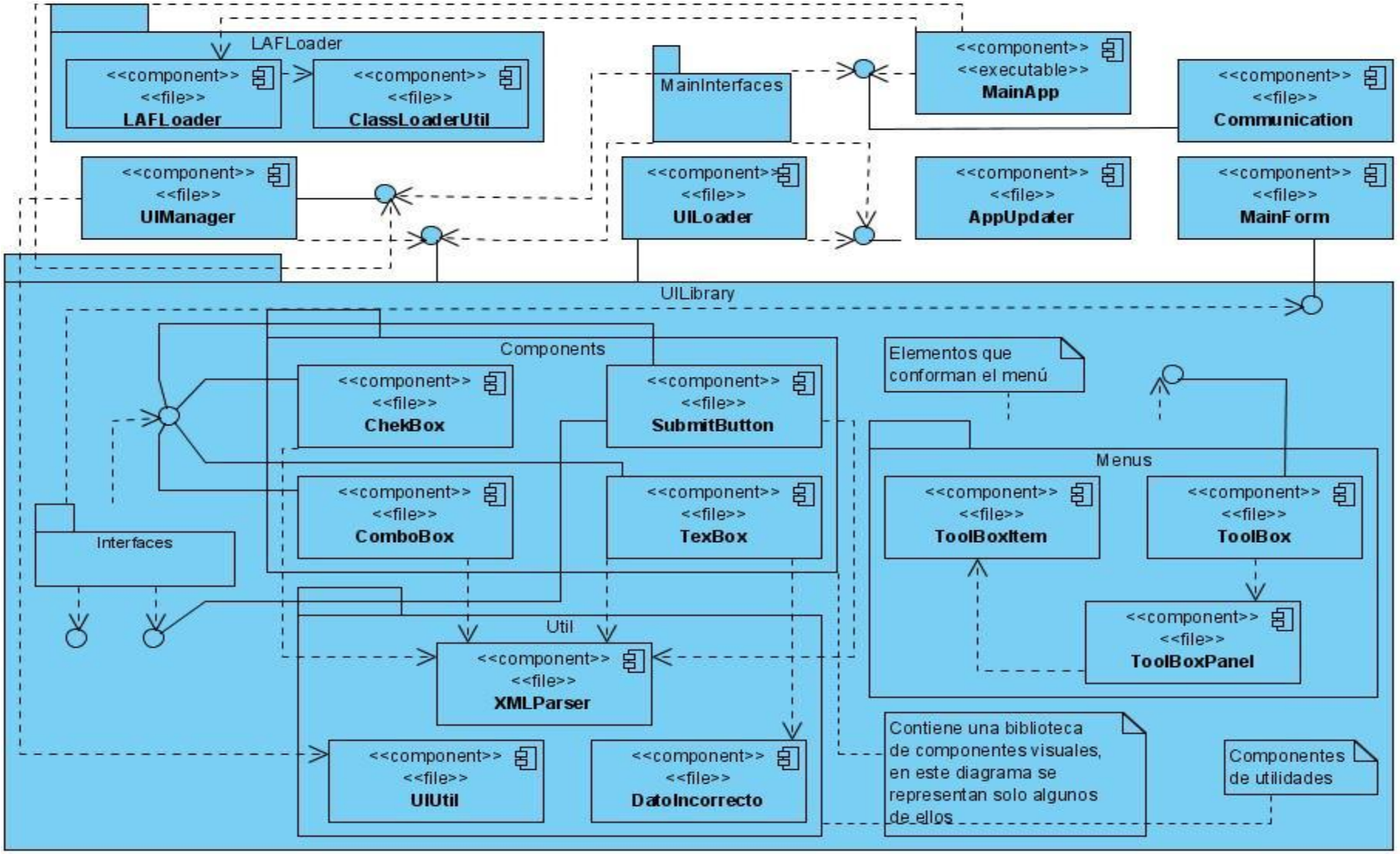
Nombre: XmlParser	
Tipo de clase : controladora	
Atributo	**Tipo**
-	-
Para cada responsabilidad:	
Nombre:	readAttribute(Node node, String attributeName): String
Descripción:	Devuelve el valor del atributo *attributeName* pasado por parámetro en el nodo *node* pasado también por parámetro o *null (nulo)* si el nodo no posee dicho atributo.

Anexo 4 Tabla 3. Descripción de la clase "XmlParser".

Nombre: DatoIncorrecto	
Tipo de clase : controladora	
Atributo	**Tipo**
Message	String
Para cada responsabilidad:	
Nombre:	DatoIncorrecto(String mensage)
Descripción:	Inicializa el atributo *message* de la clase asignándole el valor del parámetro *mensaje*.
Nombre:	getMessage():String
Descripción:	Devuelve el atributo *message* de la clase.
Nombre:	setMessage(String mensaje): void
Descripción:	Modifica el valor del atributo *message* de la clase asignándole el valor del parámetro *mensaje*.

Anexo 4 Tabla 4. Descripción de la clase "DatoIncorrecto".

Anexo 5: Diagrama de componentes general de la aplicación



Anexo 5 Fig. 1 **Diagrama de componentes de la aplicación.**

Anexo 6: Matriz de Casos de Prueba

Matriz del caso de prueba para el caso de uso Cargar Formulario

Id escenario	**Escenario**	**Variable 1** Identificador del formulario	**Variable 2** Listado de propiedades	**Variable 3** Listado de valores	**Variable 4** Acciones de los componentes	**Variable 5** Datos del Menú	**Variable 7** Formulario Repositorio local	**Variable 8** Formulario Repositorio externo	**Respuesta del Sistema**	**Resultado de la Prueba**
EC 1	Datos correctos para búsqueda en el repositorio local	V	V	V	V	V	V	I(No se utiliza)	El sistema carga el formulario con la información correspondiente.	*Satisfactorio*
EC 2	Datos correctos para búsqueda en el repositorio externo.	V	V	V	V	V	I(No se encuentra el formulario)	V	El sistema carga el formulario con la información correspondiente.	*Satisfactorio*
EC3	Datos incorrectos	I (información no suministrada)	V	V	V	V	V	I(No se utiliza)	El sistema no carga el formulario.	*Satisfactorio*

	correctos	V	I (información no suministrada)	V	V	V	V	I(No se utiliza)	El sistema carga el formulario con las propiedades de sus componentes por defecto.	*Satisfactorio*
		V	V	I (información no suministrada)	V	V	V	I(No se utiliza)	El sistema carga el formulario con sus componentes sin valores.	*Satisfactorio*
		V	V	V	I (información no suministrada)	V	V	I(No se utiliza)	El sistema carga el formulario con sus componentes sin acciones.	*Satisfactorio*
		V	V	V	V	I (información	V	I(No se utiliza)	El sistema carga el	*Satisfactorio*

						no suministrada)			formulario sin actualizar el menú de la plantilla.	
		V	V	V	V	V	V	I(No se utiliza)	El sistema no carga el formulario.	*Satisfac torio*

Anexo 6 Tabla 1. Matriz del caso de prueba para el caso de uso Cargar Formulario.