Universidad de las Ciencias Informáticas

EasyPol: Propuesta de Componentes Genéricos Reutilizables para Sistemas de Investigación Policial.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERÍA INFORMÁTICA

Autor
Reiniel Herrera Pereda.

Tutores
Ing. Susel Ruiz Durán.
Ing. Jorge Amado Soria Ramírez.

Ciudad de la Habana
Junio 2009

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 8 de la Universidad de las Ciencias Informáticas; así como a dicho centro para que hagan el uso que estimen pertinente con este trabajo.

Para que así conste firmo la presente a los ____ días del mes de Junio del año 2009.

Reiniel Herrera Pereda	**Ing. Susel Ruiz Durán**	**Ing. Jorge A. Soria Ramírez**
_____	_____	_____
Firma del Autor	Firma del Tutor	Firma del Tutor

A Susel, por ser mi tutora, y más que eso mí amiga y hermana.

A Amado, por estar en los momentos oportunos, y por ser, para mí, un ejemplo a seguir.

A Yander por el impulso inicial, los amenos debates y soportarme las necedades.

A Yadira, mi novia, por la comprensión y ayuda que me brindo en los momentos que más lo necesitaba.

A Tomás, mi amigo.

A todo el equipo del proyecto CICPC que se preocuparon por la marcha de este trabajo.

A Luis Emilio, Adonys, Jose Carlos, Yander y Eric (*el hormigón*) por forzarme a divertirme de vez en cuando, y por todos los momentos difíciles y alegres que pasamos juntos.

Al proyecto CICPC, por ponerme metas nuevas y difíciles, por ser, en gran medida, el responsable de muchas noches de desvelo de estudio y trabajo, por permitirme conocer muchos de los mejores amigos que tengo, y por mostrarme el camino que quiero seguir en la vida.

A todas aquellas personas que de una forma u otra han hecho posible que se haga realidad este sueño.

A Raidel, mi querido hermano, por toda la confianza depositada, las preocupaciones, los consejos y la ayuda incondicional que siempre me ha brindado.

Pero especialmente a mis padres, por criarme y educarme como lo hicieron. Por formar al hombre que soy. Por siempre guiarme por el camino correcto y enseñarme que en la vida las cosas hay que ganárselas con trabajo y sudor. Por todo el amor, comprensión, ayuda y apoyo que me han dado durante toda mi vida. Por estar siempre ahí, y por ser sencillamente los mejores padres y amigos que alguien puede desear. Para ustedes mis respetos, gratitud e infinito amor y agradecimiento.

A mis padres, las personas más especiales que he tenido en toda mi vida. Ellos me han mostrado el verdadero camino a seguir, y siempre han estado ahí en todos los momentos. A ustedes va dedicado este trabajo, para que se sientan orgullosos de sus dos hijos, y para que hagan realidad su sueño.

RESUMEN

Este trabajo presenta una propuesta de componentes genéricos reutilizables que soportan el control de flujos de estados, el manejo de componentes y funcionalidades básicas de la investigación, y el manejo de extensiones comunes de elementos investigativos (personas, objetos, vehículos y armas), como funcionalidades primarias para la automatización de los procesos centrales de una institución policial.

Este documento recoge los resultados de todo el trabajo investigativo realizado. Se identifican y describen los procesos centrales de las organizaciones policiales, especialmente aquellos que se van a automatizar; se describen sistemas informáticos similares que se han desarrollado en la universidad y en otros centros, y se propone, como solución a la situación problémica descrita, la creación de componentes genéricos que faciliten y potencien el desarrollo de sistemas policiales de elevada calidad en un menor tiempo. Posteriormente se hace un análisis comparativo acerca de las tecnologías existentes y se seleccionan las más apropiadas. Se muestran los resultados del diseño de la propuesta y finalmente se incluye una valoración de la factibilidad del proyecto.

ÍNDICE

INTRODUCCIÓN

La magnitud y complejidad del tema de la seguridad ciudadana indica, de acuerdo con los fracasos que han ocurrido en los últimos años, que no existen atajos fáciles o recetas milagrosas para su solución. Durante la última década, este fenómeno está entre las tres principales preocupaciones en la mayoría de las sociedades en Latinoamérica. (1)

Teniendo en cuenta la actual y agravante situación mundial con respecto al tema de la inseguridad social, la Universidad de las Ciencias Informáticas (UCI en lo adelante), está manejando la idea de confeccionar un polo productivo entorno al desarrollo de Sistemas Policiales. En la UCI, se han desarrollado varios proyectos que tienen como meta principal contribuir al mejoramiento y mantenimiento de la seguridad ciudadana en determinados países latinoamericanos, automatizando procesos de vital importancia. El estudio de los negocios existentes en el Cuerpo de Investigaciones Científicas, Penales y Criminalísticas (CICPC en lo adelante), y de Cuerpos Policiales, pertenecientes a la República Venezolana, han revelado procesos que constituyen, generalmente, el núcleo funcional de las organizaciones policiales entre los que se puede mencionar: *El Flujo de Gestión Documental, La Sustanciación de Expedientes, y El Manejo de Elementos Investigativos.*

El desarrollo de los sistemas correspondientes a estas organizaciones ha demostrado que existen dificultades para automatizar los procesos centrales mencionados anteriormente, ya que no se mantiene una línea única de trabajo para implementar estas características comunes, con la calidad, flexibilidad y rendimiento que aplicaciones de este tipo demandan, lo que conlleva a la existencia de gran dependencia entre los procesos generales y específicos del negocio, ya que procesos centrales como la *gestión documental* poseen una fuerte dependencia de los documentos específicos que son manejados; a la poca mantenibilidad del sistema dada la forma heterogénea de manejar procesos comunes; y a una reducida tolerancia a cambios, lo que provoca que un cambio pequeño pueda tener un impacto considerablemente grande en términos de desarrollo de aplicaciones. Todas estas cuestiones hacen el desarrollo largo, tedioso y lejos de lograr un alto grado de motivación en los desarrolladores, son motivos para mantenerse alejados, lo que provoca, en reiteradas ocasiones, retrasos en los cronogramas de entrega.

Por todo lo planteado, se decidió generalizar y encapsular e*l Flujo de Gestión Documental, la Sustanciación de Expedientes, y el Manejo de Elementos Investigativos*, procesos centrales de las organizaciones policiales, como vía para

facilitar y agilizar el desarrollo de Sistemas Policiales: facilitar, porque una solución general que modele la automatización de los principales flujos de trabajo y posea facilidades de personalización, permitiría a los desarrolladores enfocarse más en los requerimientos específicos de su sistema; y agilizar, porque no se tendría que asignar recursos y tiempo a encontrar una solución para la automatización de estos procesos generales: simplemente, en el peor de los casos, bastaría con personalizar una solución existente.

Por tanto el **problema** a resolver queda formulado a modo de interrogante de la siguiente forma: ¿Cómo facilitar y agilizar el desarrollo de Sistemas de Investigación Policial?

El **objeto de estudio** lo constituyen las funcionalidades necesarias para dar soporte a los procesos estándares de las organizaciones policiales.

El **campo de acción** que abarca las funcionalidades que soportan el Flujo de Gestión Documental, la Sustanciación de Expedientes y el Manejo de Elementos Investigativos.

El **objetivo general** de esta investigación es desarrollar componentes genéricos que faciliten y agilicen el desarrollo de funcionalidades centrales (específicamente Flujo de Gestión Documental, la Sustanciación de Expedientes y el Manejo de Elementos Investigativos) de los sistemas de investigación policial, siendo lo suficientemente abiertos y configurables para permitir su reutilización.

De ahí se derivan los siguientes **objetivos específicos**:

- Desarrollar un componente para el control de flujos de estados, permitiendo la configuración de los estados, transiciones y restricciones, y que incluya un soporte para notificación a interesados ante la ocurrencia de eventos señalados. Este componente podrá usarse como base para el proceso de Gestión Documental, así como para el control de estado de los Elementos Investigativos.
- Desarrollar un componente que incluya los conceptos y funcionalidades básicas relativas a la investigación y sus componentes (expedientes, elementos investigativos, documentos). Entre las funcionalidades más relevantes se pueden mencionar: la sustanciación de expedientes y la gestión de relaciones documento-documento, expediente-elemento investigativo, documento-elemento investigativo. Este componente podrá usarse como base para el proceso de Sustanciación de Expedientes y para el Manejo de Elementos Investigativos.

- Desarrollar un componente que incluya extensiones comunes de Elementos Investigativos, tales como armas, objetos, vehículos, personas y otros, y que podrá usarse en caso de que se desee incluir en un sistema policial alguno de los conceptos antes mencionados.
- Realizar pruebas unitarias y de integración a los componentes desarrollados, para comprobar su correcto funcionamiento y evaluar la factibilidad de su uso.

Con vistas al cumplimiento de los objetivos se propone la realización de las siguientes **tareas**:

- Estudiar los procesos elementales del negocio en organizaciones policiales.
- Estudiar los métodos empleados para desarrollar las funcionalidades que soportan los procesos centrales de los sistemas policiales en la UCI.
- Seleccionar las herramientas para llevar a cabo el desarrollo de los componentes y elegir la plataforma en la que serán desarrollados, fundamentando dicha elección.
- Seleccionar la Metodología de Análisis y Diseño de Sistemas Informáticos que será utilizada, para facilitar la creación y garantizar la calidad de los componentes a desarrollar.
- Analizar, diseñar e implementar los componentes propuestos.
- Desarrollar una aplicación sencilla en la que sean puestos en práctica los componentes creados.
- Evaluar los resultados de la aplicación de los componentes desarrollados.

Esta investigación expondrá procesos que constituyen el núcleo funcional de las organizaciones policiales. El desarrollo de componentes que permitan automatizar estos procesos y su posterior reutilización en la confección de sistemas policiales permitirá, si se generaliza su uso, disminuir los tiempos de desarrollo, y establecer una forma única para desarrollar sistemas de este tipo, contribuyendo al fortalecimiento de la comunidad de desarrollo de la UCI que está vinculada a la creación de sistemas de investigación policial. Todas estas variables, constituyen elementos que contribuirán de manera más rápida a la entrega de un producto que responda a las necesidades de la organización cliente, y que puede ser reajustado fácilmente, según las variaciones de los requerimientos del negocio, lo que tiene una repercusión positiva en la población en sentido general, ya que constituye un arma más para luchar por la seguridad ciudadana.

El presente documento consta de cuatro capítulos:

En el Capítulo 1 se describen varios de los procesos elementales del negocio de las organizaciones policiales. Se analizan los métodos para el desarrollo de sistemas de investigación policial en la UCI y se identifican los principales problemas que motivan el desarrollo de este trabajo. Se describe brevemente la propuesta de solución y se fundamentan los objetivos que la misma se plantea. Se realiza un análisis respecto a las tendencias, tecnologías y herramientas existentes en la actualidad que deben ser consideradas para seleccionar aquellas que se van a utilizar en el proyecto. Finalmente se plantea dicha selección a modo de propuesta, con cada uno de sus aspectos debidamente fundamentado.

En el Capítulo 2 se describen los procesos actuales a través de un modelo de dominio, el cual sirve de base para clarificar los conceptos involucrados en los procesos centrales de los sistemas policiales. Se detallan los requisitos funcionales y no funcionales de la solución a desarrollar y se modela la misma en términos de casos de uso de sistema.

El Capítulo 3 aborda aspectos relacionados con la construcción de la solución propuesta. Se realiza el análisis de cada uno de los componentes a desarrollar estructurando los requisitos para facilitar su preparación, modificación y en general su mantenimiento. Finalmente se expone el diseño de la solución propuesta.

El Capítulo 4 y último, contiene la validación de la propuesta de solución con el desarrollo de una sencilla aplicación web donde son utilizados los componentes desarrollados, describiendo para ellos los requisitos funcionales que esta debe cumplir. Finalmente se realiza la evaluación de la factibilidad del uso de los componentes desarrollados.

CAPÍTULO 1

FUNDAMENTACIÓN TEÓRICA

1.1 Introducción.

En el presente capítulo se describen varios de los procesos elementales del negocio de las organizaciones policiales, se analizan los métodos para el desarrollo de sistemas de investigación policial en la UCI y se identifican los principales problemas que motivan esta investigación. Se plantea la propuesta de solución y se fundamentan los objetivos propuestos.

Además, se realiza un análisis de las tecnologías y tendencias que existen en la actualidad a nivel mundial y que pudieran ser útiles en el desarrollo de la propuesta de solución. Se tienen en cuenta los lenguajes de programación más utilizados para desarrollar sistemas policiales, las distintas metodologías de desarrollo de software, las herramientas CASE más utilizadas, así como el lenguaje de marcado extensible (XML). Finalmente, se seleccionan las más apropiadas teniendo en cuenta que las que se utilicen deben garantizar el cumplimiento de los intereses de los usuarios finales y de la universidad en general.

1.2 Fundamentación del Tema.

1.2.1 Organizaciones policiales.

El principal rol de estas instituciones es investigar crímenes en contra de las personas o que afecten el orden público, así como el arresto de sospechosos. A continuación se describen, de modo general, los principales procesos de varias de estas instituciones en la República Bolivariana de Venezuela y Cuba.

1.2.1.1 Cuerpo de Investigaciones Científicas, Penales y Criminalísticas.

Esta organización constituye la policía científica de Venezuela y cuenta fundamentalmente con tres áreas en las que, a grandes rasgos, se centra todo el peso del trabajo: Investigación Penal, Investigación Criminalística e Investigación de Ciencias Forenses.

El objetivo principal del CICPC es investigar y esclarecer delitos penalizados por la ley, por lo que todo el flujo de trabajo está orientado básicamente a la Investigación Penal;

el área de Criminalística está especializada en la realización de experticias sobre elementos que surgen durante la investigación, sitios donde ocurrieron los hechos, entre otros; y el área de Investigación de Ciencias Forenses, se encarga de practicar determinados estudios sobre los cadáveres, así como de realizar evaluaciones psicológicas, médicas y toxicológicas de individuos (entre otras).

Además, el CICPC posee otras áreas de soporte que contribuyen a lograr el objetivo principal de la institución. Entre ellas se encuentran los departamentos de Aprehensión, Dotación de Equipos Policiales, Investigación Interna, el Centro Telefónico de Atención al Ciudadano e Interpol.

La institución cuenta actualmente con un sistema informático ya obsoleto, en el que es almacenada y manejada una parte de la información referente a los casos que han sido y están siendo investigados. De mantener actualizada toda esta información se encargan los departamentos de análisis y control de la información policial.

El proceso de Investigación Penal, comienza cuando un ciudadano, formalmente, realiza la denuncia de un hecho punible, o lo que es lo mismo, que sea penalizado por la ley. Al tomarse una denuncia, se conforma un **Acta Procesal**, carpeta en la que serán archivados todos los documentos que contribuyen al correcto desarrollo de la investigación. Durante el proceso investigativo para la resolución de un caso, con frecuencia se realizan solicitudes para la realización de experticias, se crean informes dando respuesta a las solicitudes anteriores, se ejecutan órdenes legales emitidas por el(los) fiscal(es) que estén conduciendo la investigación, se colectan evidencias, entre otros procesos de vital importancia. Durante el transcurso de una investigación, se generan también otros documentos de sustanciación del Acta Procesal, cuyo único objetivo es enriquecer la investigación y no requieren una respuesta. Entre ellos podemos mencionar: órdenes legales como la orden de aprehensión, orden de allanamiento, orden de prohibición de salida del país, entre otras. Todos los documentos que son generados durante la investigación, son incluidos en el Acta Procesal, como constancia de todas las acciones legales realizadas.

El área de Investigación de Ciencias Forenses, tiene la responsabilidad de atender todas las solicitudes emitidas por el área de investigación penal, y, adicionalmente, procesa los cadáveres que sean encontrados. En ambos casos, se crea un **Expediente Tanatológico** en el que son archivados todos los estudios practicados, tales como protocolo de autopsia, ficha antropológica para cadáveres no identificados, odontodiagrama post mortem, certificados de identidad, entre otros.

El área de Investigación Interna, tiene la responsabilidad de velar por la disciplina de los funcionarios adscritos a este organismo y reunir elementos que permitan llegar a

una decisión justa, en caso de que se pruebe que el funcionario cometió una falta. En el transcurso de un proceso investigativo en este departamento, se confeccionan determinados documentos que respaldan e indican las acciones a desarrollar para esclarecer la presunta falta cometida por el o los funcionarios implicados. Entre ellas podemos mencionar las boletas de citación formal e informal, las actas de inspección a sitios de suceso, las actas de entrevista y los autos de nombramiento de abogados. Al iniciar cada investigación, se realiza una **Averiguación Preliminar**, y en caso que se demuestre la culpabilidad del o los implicados, se procede a crear un **Expediente Disciplinario**, donde son agrupadas las averiguaciones hechas preliminarmente, y las diligencias adicionales que sean necesarias.

Durante el transcurso de una investigación, tanto penal como interna, van saliendo a la luz elementos que están relacionados al hecho punible que está siendo investigado. Con ayuda de los archivos históricos, los investigadores pueden conocer si estos elementos se encontraban relacionados previamente a otro caso, o si solo guardan relación con el actual, además, pueden aportar información vital para la conclusión de la investigación. Por esta razón son, a menudo, objeto de experticias y estudios para determinar sus propiedades, autenticidad, origen, fabricación, entre otras características. Estos elementos investigativos pueden ser armas y vehículos, de cualquier tipo, marca o modelo; objetos, de cualquier forma, material u origen; y, adicionalmente, personas, que pueden ser testigos, imputados, presuntos imputados, víctimas, agraviados, denunciantes, entre otros.

El CICPC está organizado en despachos y dependencias que realizan funciones bien definidas, ya que existe un alto grado de especialización. Adicionalmente, y en reiteradas ocasiones, se requiere colaboración y ayuda de organismos y entidades externas. Por estos motivos, se hace sumamente necesaria la comunicación tanto interna de la organización, como externa con otras entidades. Esta comunicación se realiza a través de memorandos u oficios. En una dependencia o despacho, previo al envío de un documento, sea del tipo que fuere, a una entidad que pertenezca o no al CICPC, se le practica una revisión para comprobar que no contiene errores de ninguna índole. En caso de ser detectado algún tipo de incoherencia o falta, el documento es enviado nuevamente al funcionario que le dio origen, con instrucciones para realizar su corrección. Una vez que se encuentre libre de errores, debe ser aprobado por el jefe de despacho, o en su defecto, por alguien que autorice su salida. Solamente cuando se ha completado este proceso, el documento puede emitirse a su destino. El proceso para recibir documentos es bastante controlado: una vez estos arriban a la dependencia, son registrados en el libro de correspondencia y llevados hacia el

encargado de asignar la tarea contenida en el documento a un funcionario del despacho, quien se encargará de darle cumplimiento o respuesta, según el caso. Durante estos procesos de envío y recepción, los documentos pueden ser anulados por el funcionario responsable de permitir su salida del despacho, rechazados por su destinatario o redirigidos por el despacho destino hacia otro despacho. Además, todos los documentos que sean recibidos, emitidos, anulados, rechazados o redireccionados por un despacho quedan en un archivo como respaldo y constancia de su existencia.

1.2.1.2 Cuerpos Policiales de Venezuela.

Los Organismos o Cuerpos Policiales de la República de Venezuela son las instituciones encargadas de velar por el orden público. Se desempeñan fundamentalmente en tres áreas: realizar operativos policiales, tomar las denuncias de los ciudadanos que acudan a ellas, y registrar las reseñas de las personas que sean detenidas por algún delito o falta cometida. Los operativos policiales constituyen el mecanismo para controlar determinadas amenazas al bienestar social entre las que se puede mencionar incautar un cargamento de drogas y detener a un ciudadano involucrado en un determinado delito. Las denuncias constituyen hechos o faltas que son cometidas cotidianamente tales como robo o hurto de vehículos u objetos valiosos, entre otras, que son notificadas a las autoridades por los ciudadanos. Reseñar una persona consiste en tomar todos los datos de un ciudadano que ha sido detenido por un delito o falta, cometido a priori, o en la mayoría de los casos, en flagrancia.

Poseen la responsabilidad de brindar protección a la ciudadanía frente a situaciones que constituyan amenaza, vulnerabilidad o riesgo para su integridad física, sus propiedades, el disfrute de los derechos y el cumplimiento de sus deberes (10). Adicionalmente, cuentan con otras áreas que se encargan de brindar soporte en determinados aspectos tales como recursos humanos, y gestión y control de los equipos policiales.

En algunas de estas instituciones, existen aplicaciones informáticas que automatizan varios de los procesos llevados a cabo, pero de forma no centralizada ni única, lo que ocasiona que no se lleve un control a nivel nacional de los delitos, operativos y detenciones realizadas.

Los operativos policiales, tienen origen a través de una ordenanza del Ministerio del Interior y Justicia (MIJ), o por iniciativa propia de los funcionarios del organismo policial. Para el éxito de un operativo, se llevan a cabo varias tareas previas a su ejecución, tales como la planificación, donde se asignan recursos, tiempo de duración

y funcionarios responsables. Además, se definen los mecanismos que serán utilizados. Durante un operativo policial, se pueden reseñar, verificar y encontrar elementos (armas, vehículos, objetos) y personas que se encuentren solicitados por el CICPC. Adicionalmente, se pueden obtener sustancias ilícitas como drogas. Los resultados arrojados por los operativos, son registrados con el fin de que contribuyan a combatir el delito, y de informar al CICPC sobre el encuentro de elementos y personas solicitadas, para que puedan ser tomadas las medidas pertinentes.

El proceso de registrar una denuncia comienza cuando un ciudadano acude a una institución policial a presentar una falta o delito cometido por otro u otros ciudadanos. En el momento de ser tomada una denuncia, se determinan las infracciones de la ley cometidas a una hora y en una dirección determinada, conformando así un caso. Además, se toman los datos del denunciante, de los elementos (objetos, armas, vehículos), y personas (denunciados, víctimas, testigos), relacionados al hecho delictivo, los cuales son asociados a la denuncia que está siendo creada. Al registrarse un nuevo **caso**, es abierto un nuevo **expediente**, en el cual se almacenarán todas las **denuncias** realizadas en el caso en cuestión, y todos los elementos relacionados a estas. Estos expedientes son trasladados hacia una determinada fiscalía, en la cual se determinará si el caso es asignado a otra institución policial, o se mantiene en el organismo inicial, dados los progresos de este en la investigación.

Al realizar una detención, son tomados todos los datos del ciudadano detenido, y archivados en el expediente que contiene la información pertinente al caso en cuestión. Si el expediente no existía con anterioridad, es creado, permitiendo su posterior uso en las diferentes acciones necesarias para desarrollar la investigación, mientras esta competa a la institución policial.

1.2.1.3 MININT.

MININT es el acrónimo con que se conoce el *Ministerio del Interior de la República de Cuba*. Está dividido en órganos y estructuras que cumplen funciones de seguridad ciudadana y de establecimiento del orden interior, entre las que se pueden mencionar la Dirección General de la Policía Nacional Revolucionaria (PNR), Dirección de Atención a Ciudadanía, Dirección de Instrucción Penal (DIP), Dirección Técnica Investigativa (DTI, Criminalística e investigación criminal), entre otras.

Incluye asimismo diversos órganos de logística, preparación de la fuerza, entre otros. Además, posee empresas comerciales que brindan servicios de seguridad, tanto humana (Servicios Especializados de Protección, S.A., SEPSA) como técnica (SEISA,

ACERPROT), que incluyen cadenas de tiendas de venta a la población de aditamentos de seguridad. (15)

1.2.1.4 Flujo actual de los procesos involucrados en el campo de acción.

En los epígrafes anteriores, fueron descritos los principales procesos de varias organizaciones policiales. De forma general, los procesos fundamentales de estas organizaciones se pueden sintetizar de la siguiente manera:

- *Sustanciación de Expedientes*: se corresponde con la creación y actualización de expedientes en las distintas áreas investigativas, dentro de los cuales se archivan documentos u otros expedientes que contienen información importante para el desarrollo de la investigación.
- *Manejo de elementos investigativos*: concierne la creación, modificación y eliminación de elementos como armas, objetos, vehículos y personas, para contribuir al correcto desarrollo de la investigación. Incluye la gestión del flujo de estados por los que transitan los elementos investigativos en cada momento de los procesos de negocio en los que están involucrados.
- *Gestión Documental*: agrupa lo referente a la comunicación interna de la organización y con entidades externas a ella, así como el tratamiento a la documentación de funcionamiento interno, haciendo pasar a cada documento por procesos de revisión, aprobación, remisión, recepción, asignación u otros, según sea necesario.

1.2.2 Desarrollo de Sistemas Policiales.

En la UCI, han sido desarrollados los sistemas correspondientes a las organizaciones policiales venezolanas anteriormente descritas. Estos sistemas tienen como objetivo principal elevar la seguridad ciudadana en la población de Venezuela. En los dos epígrafes siguientes, son descritos, de manera general estos sistemas.

1.2.2.1 SIIPOL.

El sistema desarrollado para el CICPC tiene como nombre Sistema de Investigación e Información Policial (SIIPOL en lo adelante). Este sistema soporta decisiones estratégicas del MIJ y la Dirección general de la Institución, contribuye a controlar y organizar el trabajo en las dependencias, y mejora el nivel de respuesta a las necesidades de seguridad del ciudadano venezolano.

Este sistema cuenta con elementos de gestión documental, para orientar de manera más fácil la información al cliente. Estos elementos están agrupados en una *Agenda de Trabajo*, en la cual se dispone el manejo de solicitudes, informes, comunicaciones u

oficios en función del estado en que estos se encuentren, y del área a la que pertenezcan. Dichas áreas, generalmente, constituyen módulos del sistema, ya que engloban un conjunto de procesos bien definidos como la investigación penal, criminalística y forense.

En determinadas áreas investigativas, existen expedientes con el propósito de archivar todos los documentos creados a partir de los procesos fundamentales que se desempeñan en la misma. Para ejemplificar lo anteriormente planteado, se puede hacer mención del Acta Procesal, perteneciente al área de investigación penal, en la cual son archivados todos los documentos que se encuentren asociados a la investigación de un caso.

SIIPOL incluye varios elementos de investigación tales como armas, objetos, vehículos y personas, los cuales pueden ser incluidos, modificados, eliminados, y consultados. Estos elementos son manejados por un único modulo que se encarga de brindar los servicios necesarios a todos los restantes, permitiendo además, asociarlos a documentos, tales como denuncias, solicitudes, experticias, entre otros. Constituyen el núcleo de los procesos investigativos en esta organización, ya que están presentes de alguna forma, en casi la totalidad de ellos.

1.2.2.2 SIGEPOL.

El Sistema de Gestión Policial (SIGEPOL), fue definido como un sistema que tiene como objetivo la captura de la información sobre la gestión policial, que sirva de base a la toma de decisiones en la sala situacional del Centro de Tratamiento y Análisis de Información de Seguridad Ciudadana (CTAISC), a fin de que la información y el análisis de esta ayuden a combatir el delito en la República Bolivariana de Venezuela.

Este sistema consta de tres módulos principales. El módulo Denuncia, tiene la responsabilidad de brindar tratamiento a los hechos o faltas informados por los ciudadanos. Una vez tomada una denuncia es abierto un caso investigativo, el cual es almacenado dentro de un expediente, al igual que todos los documentos que son necesarios en el transcurso de la investigación. El módulo Operativos Policiales, es el encargado de llevar el control de todas las planificaciones, asignaciones y recursos destinados a realizar los operativos policiales, así como los resultados de estos: elementos recuperados, incautados entre otros. Finalmente, incluye el módulo de Reseñas, que posibilita llevar un control de todos los datos de los ciudadanos detenidos, con el objetivo de agilizar el proceso investigativo.

SIGEPOL, incorpora armas, objetos y vehículos, de distintas clasificaciones, marcas y modelos, para poder reflejar todos los elementos que están relacionados al hecho

investigado, y que, de alguna manera, pueden constituir factores importantes en la solución de un caso.

Adicionalmente, posee diversos reportes especializados que brindan información clara y actualizada sobre las denuncias, casos, expedientes, detenidos y operativos policiales realizados a nivel nacional, ya que cuenta con una base de datos central para todas las organizaciones policiales del país.

1.2.2.3 SAJO

SAJO, acrónimo del Sistema Automatizado Jurídico Operativo, perteneciente al Ministerio del Interior de la República de Cuba. Este sistema maneja el registro, control y seguimiento de los hechos delictivos denunciados en cualquier lugar del territorio nacional, así como otros casos que no constituyen delitos, como son: los índices de peligrosidad, las muertes, lesiones y daños por accidentes del tránsito, las muertes y lesiones accidentales, los ausentes a domicilio, los suicidios y las muertes naturales.

Constituye el medio automatizado que permite medir el trabajo de las unidades y el comportamiento del delito en los territorios del país, en diferentes períodos. Apoya y facilita el análisis de la situación delictiva a distintos niveles de Dirección.

Su objetivo esencial es el registro y control de todas las **Denuncias** de los hechos delictivos ocurridos en cualquier lugar del país, las cuales se registran en un **Expediente** (uno ya existente, si la denuncia se refiere a un hecho que está siendo investigado; nuevo en caso contrario). Dispone de un Número de Denuncia que se asigna automáticamente por la computadora centralizadamente a nivel provincial para identificar de forma única los hechos en todo momento, y que se conforma según el lugar que registra, el año en curso y el número consecutivo que comienza en 1 al inicio de cada año.

Cada denuncia puede tener asociado un conjunto de objetos de cualquier tipo, marca o modelo, los cuales pueden contribuir al desarrollo exitoso de la investigación, por lo que son registrados. Así mismo son registrados los datos identificativos de las personas relacionadas a la investigación: denunciantes, procesados, víctimas, detenidos, testigos, entre otros.

Emplea una base central única por cada provincia, a fin impedir que existan diferencias en los contenidos de las bases de datos de uno y otro lugar de la estructura.

1.2.2.4 Problemas encontrados.

En el SIIPOL, los elementos encontrados de gestión documental se encuentran de manera dispersa y desorganizada, ya que fueron pensados e implementados en

función de los procesos de negocio de la organización, y no en función de la gestión de documentos. Su implementación requirió un tiempo y esfuerzo bastante considerable ya que fue realizada cuando casi la totalidad de las funcionalidades estaban implementadas, por lo que fueron puestas en práctica varias alternativas que condujeron a la coexistencia de código muy diferente para las mismas funcionalidades.
Todo el proceso de la Agenda de Trabajo del SIIPOL, está basado en un flujo de estados por los que pasa cada documento manejado en ella. Las transiciones de estado son realizadas de manera explícita en el código, y varios procesos importantes dependen de estas, por lo que se puede asegurar que un pequeño cambio en el flujo de estados, cambiaría casi totalmente el sistema.
Otro de los problemas detectados es la existencia de diferentes mecanismos para sustanciar expedientes. Se puede ejemplificar esto con el Acta Procesal y el Expediente Tanatológico del SIIPOL; el primero es utilizado para archivar todo lo relativo a la investigación de un caso, el segundo para almacenar experticias y pruebas realizadas sobre cadáveres. Dichos expedientes poseen propósitos diferentes, pero, el principio de funcionamiento es el mismo: *un expediente es una carpeta para almacenar otras carpetas y/o documentos*. Al no tener en cuenta este principio, fueron diseñadas e implementadas diferentes vías para cumplir el mismo objetivo, y en algunos casos, *"reinventar la rueda"*, lo que provocó invertir tiempo en actividades que no lo requerían, dilatando así el cronograma de entrega del producto final.
En todo proceso de investigación policial, se encuentran relacionados elementos entre los que se pueden mencionar arma, objetos, vehículos y personas. Estos elementos constituyen el centro de toda organización policial ya que están relacionados de una forma u otra a todos los procesos que son llevados a cabo en la institución. En reiteradas ocasiones, por restricciones del negocio, de las tecnologías utilizadas, o simplemente por la capacidad, habilidad y diversidad de los equipos de desarrollo, a cada elemento investigativo se le otorga un tratamiento similar, pero codificado diferente, lo que provoca poca mantenibilidad del sistema.

1.2.3 Propuesta de Solución.

Teniendo en cuenta la situación problémica planteada se propone la elaboración de componentes genéricos capaces de modelar el Flujo de Gestión Documental, la Sustanciación de Expedientes, y el Manejo de Elementos Investigativos como procesos centrales de un sistema de investigación policial. Estos componentes incluirán las siguientes funcionalidades:

- Tratamiento único a la creación y sustanciación de los expedientes, basado en la premisa de que todo expediente es un contenedor de información, capaz de almacenar otros expedientes y/o documentos.
- Estandarización del modelo general de respuesta a un documento. Cada solicitud, excepto aquellas que son anuladas, deben tener asociada una respuesta, esto es conocido como el par Solicitud-Respuesta.
- Notificación a los elementos contenedores, como los expedientes, al ocurrir alguna acción de interés sobre un elemento que esté contenido en ellos, por ejemplo las personas involucradas.
- Soporte para la relación de los elementos investigativos con contenedores asociados.
- Soporte para relaciones entre los elementos investigativos.
- Creación de extensiones comunes de elementos de investigación, como armas, objetos, personas y vehículos.
- Definición y gestión de estado para los elementos investigativos.
- Definición y gestión del proceso de gestión documental, basado en un flujo de estados.
- Facilitación de la configuración del flujo de estados para cualquier modificación en el proceso de gestión documental y para los elementos investigativos.

1.2.4 Objetivos Propuestos.

Para cumplimentar la propuesta de solución planteada en el epígrafe anterior, y en respuesta a la situación problémica, son propuestos un conjunto de objetivos expuestos a continuación.

1.2.5 Objetivo General.

El **objetivo general** de esta investigación es desarrollar componentes genéricos que faciliten y agilicen el desarrollo de funcionalidades centrales (específicamente Flujo de Gestión Documental, la Sustanciación de Expedientes y el Manejo de Elementos Investigativos) de los sistemas de investigación policial, siendo lo suficientemente abiertos y configurables para permitir su reutilización.

1.2.6 Objetivos Específicos.

- Desarrollar un componente para el control de flujos de estados, permitiendo la configuración de los estados, transiciones y restricciones, y que incluya un soporte para notificación a interesados ante la ocurrencia de eventos señalados. Este componente podrá usarse como base para el proceso de Gestión Documental, así como para el control de estado de los Elementos Investigativos.
- Desarrollar un componente que incluya los conceptos y funcionalidades básicas relativas a la investigación y sus componentes (expedientes, elementos investigativos, documentos). Entre las funcionalidades más relevantes se pueden mencionar: la sustanciación de expedientes y la gestión de relaciones documento-documento, expediente-elemento investigativo, documento-elemento investigativo. Este componente podrá usarse como base para el proceso de Sustanciación de Expedientes y para el Manejo de Elementos Investigativos.
- Desarrollar un componente que incluya extensiones comunes de Elementos Investigativos, tales como armas, objetos, vehículos, personas y otros, y que podrá usarse en caso de que se desee incluir en un sistema policial alguno de los conceptos antes mencionados.
- Realizar pruebas unitarias y de integración a los componentes desarrollados, para comprobar su correcto funcionamiento y evaluar la factibilidad de su uso.

1.3 Tendencias y Tecnologías Actuales a Considerar.

1.3.1 Metodologías de Desarrollo de Software.

Las metodologías de desarrollo de software, hoy en día, constituyen uno de los temas más polémicos en el mundo del desarrollo de aplicaciones. Su uso persigue, como objetivo principal, lograr el éxito rotundo de los proyectos de producción de software, para lo cual imponen un proceso disciplinado con el fin de hacerlo más predecible y eficiente.

El desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Por una parte tenemos aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán, las cuales son conocidas como “metodologías pesadas o formales”. Otra propuesta es centrarse en otras dimensiones, como por ejemplo el

factor humano o el producto software, siendo esta la filosofía de las llamadas "metodologías ágiles", las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. (5) Se analizarán a continuación varias de las más conocidas, sus características, ventajas y desventajas.

1.3.1.1 El Proceso Unificado de Rational (RUP)

RUP son las siglas en inglés de *Rational Unified Process*. Es una de las metodologías más amplias y generales que existen, ya que no es un sistema con pasos firmemente establecidos, sino un conjunto de procedimientos adaptables al contexto y necesidades de cada organización o proyecto. Es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo a través del lenguaje unificado de modelado (UML). El ciclo de vida RUP es una implementación del *desarrollo en espiral*[1], que organiza las tareas en fases, iteraciones y disciplinas.

Cada fase representa un estado del proyecto, y produce un hito que sirve de entrada a la próxima fase. Todos los flujos o disciplinas se aplican en todas las fases, si bien algunos tienen más carga de trabajo que otros en determinada fase.

El ciclo de vida de RUP se caracteriza por ser:

- Dirigido por casos de uso, que reflejan lo que los usuarios futuros necesitan y desean, guiando el proceso de desarrollo ya que los demás artefactos representan la realización de los casos de uso.
- Centrado en la arquitectura, que muestra la visión común del sistema completo y describe los elementos del modelo que son más importantes para su construcción.
- Iterativo e incremental, lo que significa que cada fase se desarrolla en iteraciones que involucran actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros, obteniendo un producto con un determinado nivel que irá creciendo incrementalmente en cada iteración.

[1] El **Desarrollo en Espiral** es un modelo de **ciclo de vida** desarrollado por Barry Boehm en 1985, utilizado generalmente en la Ingeniería de Software. Las actividades de este modelo se conforman en una espiral, cada bucle representa un conjunto de actividades. Las actividades no están fijadas a priori, sino que las siguientes se eligen en función del análisis de riesgo, comenzando por el bucle interior.

1.3.1.2 Programación Extrema (XP)

XP, siglas en inglés de *Extreme Programming*, se centra en prácticas y principios concretos para la implementación de un proyecto: comunicación, simplicidad, retroalimentación y coraje. Se adapta a procesos con requerimientos inestables, de alto riesgo, para pequeños equipos de desarrollo o que posean poca experiencia. Una de sus principales innovaciones es integrar las pruebas como parte del proceso de desarrollo.

Constituye una perspectiva deliberada y disciplinada al desarrollo de software. Es exitosa debido a que enfatiza la satisfacción del cliente. Esta metodología está diseñada para entregar el software que los clientes necesitan cuando lo necesitan. Le entrega el poder a los desarrolladores para responder con confianza a los requerimientos cambiantes del consumidor final, incluso cuando el proyecto ya está en etapas avanzadas.

Los desarrolladores que emplean XP se comunican con sus clientes y entre ellos mismos, para mantener un diseño limpio y simple. Obtienen retroalimentación probando el software desde el primer día. Entregan el sistema a los clientes lo más pronto posible, e implementan los cambios a medida que son sugeridos.

1.3.1.3 Scrum

Scrum es un proceso ágil y liviano que sirve para administrar y controlar el desarrollo de software. El desarrollo se realiza en forma iterativa e incremental (una iteración es un ciclo corto de construcción repetitivo). Cada ciclo o iteración termina con una pieza de software ejecutable que incorpora nueva funcionalidad. Las iteraciones en general tienen una duración entre 2 y 4 semanas.

Se focaliza en priorizar el trabajo en función del valor que tenga para el negocio, maximizando la utilidad de lo que se construye y el retorno de inversión. Está diseñado especialmente para adaptarse a los cambios en los requerimientos. Los requisitos y las prioridades se revisan y ajustan durante el proyecto en intervalos muy cortos y regulares. De esta forma se puede adaptar en tiempo real el producto que se está construyendo a las necesidades del cliente. Se busca entregar software que realmente resuelva las necesidades, aumentando la satisfacción del cliente.

El equipo de desarrollo se enfoca en una única cosa: construir software de calidad. Por otro lado, la gestión de un proyecto Scrum se focaliza en definir cuáles son las características que debe tener el producto a construir (qué construir, qué no y en qué orden) y en remover cualquier obstáculo que pudiera entorpecer la tarea del equipo de

desarrollo. Se busca que los equipos sean tan efectivos y productivos como sea posible.

Tiene un conjunto de reglas y está basado en los principios de inspección continua, adaptación, auto-gestión e innovación. El cliente se entusiasma y se compromete con el proyecto dado que ve crecer el producto iteración a iteración y encuentra las herramientas para alinear el desarrollo con los objetivos de negocio de su empresa. Por otro lado, los desarrolladores encuentran un ámbito propicio para desarrollar sus capacidades profesionales y esto resulta en un incremento en la motivación de los integrantes del equipo. (2)

1.3.1.4 Desarrollo de Componentes de Software Reutilizables (CSR)

En la actualidad, no existe una metodología patentizada y validada para el desarrollo de componentes de software reutilizable. Sin embargo, diversas empresas y universidades han personalizado el Proceso Unificado de Desarrollo según sus necesidades, logrando un modelo a seguir válido, y que garantiza el cumplimiento de los objetivos trazados. Entre ellas podemos mencionar el Método *WATCH-Component*, marco metodológico orientado al desarrollo de aplicaciones empresariales caracterizadas por estar basadas en la reutilización de componentes, emplear tecnología web y ser de pequeña a mediana escala. Consta de tres modelos: el modelo de producto, que captura las propiedades de los CSR; el modelo del proceso, que describe las actividades necesarias para realizar la producción; y el modelo del grupo de desarrollo, que describe la estructura y los roles del personal que participa en el proyecto de desarrollo. (5)

1.3.1.5 Análisis comparativo.

RUP constituye una metodología formal, ampliamente configurable, tan general, que puede ser aplicada a casi cualquier proyecto de desarrollo de software, permitiendo adaptar sus artefactos, roles, y modelos a las especificidades del proyecto que esté siendo desarrollado. *Extreme Programming* es una metodología ágil, que requiere interacción con el cliente en todo momento, que es utilizada principalmente en proyectos donde se automaticen procesos con requerimientos inestables, y que tiene como meta entregar el producto al cliente lo más rápido posible. Scrum es un proceso ágil y liviano, que plantea el desarrollo de iteraciones cortas, facilitando la adaptación a cambios en los requerimientos, los que son revisados y ajustados durante períodos muy cortos y regulares de tiempo; y que requiere interacción con el cliente, aunque de forma moderada.

1.3.1.6 Metodología seleccionada.

Para cumplir los objetivos de esta investigación, se hace necesario utilizar una metodología altamente personalizable, que proporcione facilidades para su configuración, y que no requiera una constante interacción con el cliente. Además, que proporcione artefactos formales que posibiliten la correcta documentación de los componentes desarrollados para su posterior utilización. Por estas razones, la metodología seleccionada para guiar el proceso de desarrollo de la solución propuesta en esta investigación es RUP, la cual será personalizada, según las necesidades de este proyecto.

1.3.2 Lenguajes de Programación.

En la actualidad existen múltiples lenguajes de programación, para aplicaciones de escritorio, para aplicaciones web, para inteligencia artificial, entre otros muchos tipos de aplicaciones que se pudieran mencionar. Teniendo en cuenta que, generalmente, los sistemas policiales utilizan una arquitectura distribuida cliente-servidor se caracterizarán brevemente los lenguajes más robustos y utilizados en la UCI, con el fin de seleccionar uno de ellos para la elaboración de la solución propuesta.

- Java es un lenguaje simple, orientado a objetos, distribuido, robusto, seguro, de arquitectura neutra, portátil, de alto desempeño, dinámico y que tiene soporte para hilos múltiples. Posee gran cantidad de librerías que permiten y facilitan el rápido desarrollo de aplicaciones, ya que es libre y de código abierto.
- PHP es un lenguaje interpretado de propósito general ampliamente usado, que está diseñado especialmente para desarrollo web y puede ser incrustado dentro de código HTML. Generalmente se ejecuta en un servidor web, tomando el código escrito en PHP como su entrada y creando páginas web como salida. Puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas sin costo alguno.
- C# permite el desarrollo de aplicaciones en forma rápida. Es un lenguaje que se integra bien con el desarrollo de aplicaciones Web, XML y muchas de las tecnologías emergentes. Combina las mejores ideas de lenguajes como C, C++ y Java con las mejoras de productividad de *.NET Framework* de *Microsoft* y brinda una experiencia de codificación muy productiva tanto para los nuevos programadores como para los veteranos.

Una vez analizadas las principales características de los lenguajes antes mencionados, se decidió que la solución propuesta se desarrollará en lenguaje de programación Java, ya que este ofrece una amplia gama de componentes reutilizables

que facilitan y agilizan el desarrollo, se puede ejecutar en cualquier sistema (siempre y cuando dicho sistema posea una implementación de la máquina virtual de Java), es libre, robusto y de muy alto nivel, lo que posibilita que el desarrollador se centre más en cubrir los requerimientos del negocio, que en las trabas y especificidades del lenguaje. Adicionalmente, es necesario tomar en consideración el hecho de que los sistemas policiales desarrollados en la UCI hasta el momento han sido implementados en este lenguaje.

1.3.3 Entorno de Desarrollo.

Eclipse, es un entorno de desarrollo integrado, de código abierto, multiplataforma, para desarrollar lo que se conoce como "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. El entorno de desarrollo integrado (IDE) de eclipse emplea módulos (en inglés, *plugins*) para proporcionar toda su funcionalidad al frente de la plataforma de cliente rico, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Esta propia arquitectura de módulos, permite escribir cualquier extensión deseada en el ambiente. Provee soporte para Java y CVS.

Para la elaboración de la solución propuesta se utilizará el **Red Hat Developer Studio**, un entorno de desarrollo integrado basado en Eclipse, que combina productos aportados a *Red Hat* por *Exadel* en marzo de 2007 -*Exadel Studio Pro*, *RichFaces* y *Ajax4jsf*- con software de *middleware JBoss*, como *JBoss Seam* e *Hibernate*, dentro de un potente entorno de desarrollo para aplicaciones SOA, Ajax y Java empresariales.

1.3.4 Herramientas CASE.

Las herramientas CASE (siglas en inglés de *Computer Aided Software Engineering*) son un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores durante todos los pasos del ciclo de vida del software. Constituyen un soporte automatizado para el desarrollo y mantenimiento de software. Se pueden ver como la unión de las herramientas automáticas de software y las metodologías de desarrollo de software formales. Existen múltiples herramientas de este tipo especializadas con diferentes propósitos: para la fase de diseño, para generar código, algunas tienen una visión de desarrollo orientada a procesos sin la capacidad de modelamiento, mientras otras proveen el modelamiento sin incluir los procesos de Análisis o Diseño. Entre las herramientas CASE más conocidas y utilizadas a escala internacional, podemos mencionar *ERWIN*, *EasyCASE*,

OracleDesigner, *PowerDesigner*, *Rational Rose* y *Visual Paradigm*. A continuación se analizarán las más utilizadas en la UCI.

1.3.4.1 Rational Rose

Es una de las más poderosas herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos. Se utiliza para modelar un sistema antes de proceder a construirlo. Cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases. Permite analizar patrones ANSI C++, Rose J y Visual C++ basado en "*Design Patterns: Elements of Reusable Object-Oriented Software*". Soporta ingeniería inversa para algunos de los conceptos más comunes de Java 1.5, la generación de código Ada, ANSI C ++, C++, CORBA, Java y Visual Basic, con capacidad de sincronización modelo - código configurables, Enterprise Java Beans™ 2.0, modelado UML para trabajar en diseños de base de datos con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos. Además posee capacidad para integrarse con cualquier sistema de control de versiones *SCC-compliant*, incluyendo a *Rational ClearCase* y permite la publicación web y generación de informes para optimizar la comunicación dentro del equipo. (4)

1.3.4.2 Visual Paradigm.

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Los sistemas de modelado UML ayudan a una más rápida construcción de aplicaciones de calidad y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, ingeniería inversa, generar código a partir de diagramas, generación de objetos a partir de bases de datos, generación de bases de datos a partir de diagramas de entidad relación y generar documentación. Permite interoperabilidad con modelos UML2 (metamodelos UML 2.x para plataforma Eclipse). Se integra con Visio. Posee una plataforma, llamada SDE, capaz de integrarse con *Eclipse*, *NetBeans*, *Oracle JDeveloper*, *JBuilder*, *IntelliJ IDEA*, *WebLogic Workshop*, *Microsoft Visual Studio*, entre otras.

1.3.4.3 Herramienta seleccionada.

La herramienta seleccionada para modelar y mantener la solución es el *Visual Paradigm* para UML en su versión 6.0. La decisión se basa, fundamentalmente, en el hecho de que la herramienta posee una plataforma denominada SDE capaz de

integrarse con el entorno de desarrollo seleccionado, permitiendo realizar el proceso de ingeniería en ambos sentidos e incluye muchas funcionalidades para el lenguaje Java.

1.3.5 Lenguaje de marcado extensible (XML)

XML, siglas en inglés de *E**x**tensible **M**arkup **L**anguage*, es un metalenguaje extensible de etiquetas, desarrollado por el *World Wide Web Consortium* (W3C). Es una simplificación y adaptación del SGML[2] y permite definir la gramática de lenguajes específicos. No es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. 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 entre otras muchas aplicaciones.

1.3.6 Herramientas seleccionadas para desarrollar la solución.

A modo de resumen de todo lo planteado en los epígrafes anteriores de este capítulo, se puede concluir que la solución propuesta se desarrollará en lenguaje de programación Java, sobre el entorno de desarrollo integrado (IDE) *Red Hat Developer Studio*. Como herramienta de modelado se utilizará el *Visual Paradigm for UML 6.0*, y para guiar todo el proceso de desarrollo se utilizará RUP. Se utilizará el XML para realizar las configuraciones pertinentes y necesarias para alistar el ambiente de trabajo y el desarrollo de los componentes propuestos.

1.3.7 Herramientas para el desarrollo de la aplicación de muestra.

La aplicación de muestra será una aplicación web, donde se apliquen los componentes desarrollados. Para su implementación se utilizará el mismo entorno, tecnologías, herramientas, lenguajes y metodología seleccionados en el epígrafe anterior. Haciendo uso de la potente arquitectura del IDE seleccionado, se utilizarán *frameworks* que faciliten y aceleren el desarrollo tales como: *Spring*, *Hibernate* y *JSF*. Como sistema gestor de base de datos, se utilizará *PostgreSQL*, ya que es un gestor libre, de código abierto y altamente eficiente. Además, se utilizarán las hojas de estilo en cascada (CSS) para el diseño de la interfaz de usuario. Para desplegar la aplicación final, se utilizará el servidor web *Apache Tomcat* en su versión 6.0.

[2] **SGML** son las siglas de ***S**tandard **G**eneralized **M**arkup **L**anguage* o "Lenguaje de Marcado Generalizado". Consiste en un sistema para la organización y etiquetado de documentos. La Organización Internacional de Estándares (ISO) normalizó este lenguaje en 1986.

1.4 Conclusiones.

En este capítulo se ha descrito el objeto de estudio de este trabajo, determinándose los problemas que presentan. Se ha demostrado la necesidad de cambio y se ha planteado una propuesta de solución, junto a los objetivos generales y específicos.

Se realizó un análisis de las tecnologías a utilizar en el desarrollo de la propuesta de solución. Se fundamentó la elección del lenguaje de programación, las herramientas, el entorno de desarrollo, la metodología de desarrollo de software, así como el uso de otras tecnologías. Finalmente se planteó la propuesta que incluye dichos aspectos.

A partir de este capítulo, se abordará el desarrollo de la propuesta de solución.

CAPÍTULO

DESCRIPCIÓN DE LA PROPUESTA DE SOLUCIÓN

2.1 Introducción.

En el presente capítulo se describe la propuesta de solución. Se presenta un modelo de dominio para clarificar los conceptos involucrados en los procesos centrales de los sistemas policiales descritos en el capítulo anterior. Posteriormente se detallan los requisitos funcionales y no funcionales de la solución a desarrollar y se modela la misma en términos de casos de uso de sistema.

2.2 Modelo de Procesos.

Como punto de partida, es necesario aclarar que los procesos involucrados en esta investigación, no representan procesos "reales" de negocio, existentes en las organizaciones policiales estudiadas, sino procesos genéricos que se manifiestan de manera directa o indirecta en su trabajo cotidiano. Constituyen una generalización de las principales actividades desarrolladas por estas instituciones, obtenida como resultado del estudio de los correspondientes negocios. Por estas razones, se decidió realizar un modelo de dominio para proporcionar una mayor comprensión del área de acción de la solución propuesta.

2.2.1 Actores.

Los actores del negocio son aquellas personas o sistemas que obtienen un resultado de valor a partir de uno o varios procesos del negocio. Los procesos que constituyen objeto de estudio en el presente trabajo no se corresponden con procesos existentes en un negocio determinado, sino que son un conjunto de actividades repetitivas que son fuertes candidatas a la automatización en todo proceso de informatización de un sistema policial. Por estos motivos no se identificaron **actores de negocio** que inicialicen e interactúen en los procesos comprendidos en esta investigación.

2.2.2 Descripción de los procesos.

A continuación se describen los procesos genéricos de las instituciones policiales que constituyen el centro de este trabajo.

- La **Gestión Documental** pretende el tratamiento integral, consistente y fiable de los documentos y la información que se genera en los procesos de negocio. Durante las actividades cotidianas llevadas a cabo, son creados, modificados, y archivados documentos que divergen tanto en su tipo y finalidad, como en la información que contienen. Todo el trabajo de las instituciones policiales está fuertemente atado a documentos legales, por lo que, estos son sometidos a varias revisiones a diferentes niveles con el fin de minimizar los errores que puedan contener. Generalmente, una institución de esta índole está dividida en varias sub-organizaciones. Es sumamente necesario mantener la comunicación entre ellas, así como con organismos e instituciones externas. Básicamente, este proceso es iniciado con la creación de un documento, punto en el que comienzan los procesos de revisiones y correcciones del mismo, hasta que es enviado hacia su destino y archivado en la sub-organización origen como parte de la documentación histórica generada por esta. Una vez que el documento llega a su destino, puede tomar varios caminos como ser enviado a otra sub-organización o ser rechazado, por solo mencionar algunos ejemplos.

- En las instituciones policiales, generalmente, la actividad mínima que se realiza es la toma de una denuncia, hecho a partir del cual es abierto un expediente. Dicho expediente constituye el “archivo” o “carpeta” donde serán ubicados todos los elementos que tributen de alguna manera al desarrollo de la investigación. Estos elementos pueden ser documentos, evidencias, estudios practicados por expertos sobre determinadas sustancias, o sencillamente otros archivos o carpetas. Por tanto, de forma general, los **Expedientes**, sean investigativos o no, constituyen una manera simple de mantener relacionados y agrupados elementos que tributan a un mismo objetivo, y su enriquecimiento o **Sustanciación** garantiza el cumplimiento de su objetivo. La actividad de sustanciar un expediente comienza una vez que este es creado y solo termina cuando es archivado.

- Como actividad derivada y complementaria de las anteriormente expuestas, surge la asociación de **Elementos Investigativos**. Estos elementos no son más que personas u objetos tangibles, que están asociados a la investigación o pueden contribuir al exitoso desenlace de la misma. Tanto al crear un expediente, como un documento, se hace necesario recabar toda la información posible que se encuentre relacionada a los hechos ocurridos. Los objetos y personas asociados, poseen una “vida” dentro de cada investigación, es decir, una vez que estos son detectados, transitan por una serie de “estados” que indican en cada momento las

actividades que están siendo realizadas sobre ellos. Adicionalmente, pueden estar relacionados entre sí: una persona puede ser propietaria de un arma o vehículo, por solo exponer algún ejemplo. Una vez que una investigación es concluida, los elementos asociados a ella, son archivados para contar con una fuente para referencias futuras.

2.3 Modelo de Dominio.

De modo general, se pueden representar los principales conceptos presentes en los procesos antes descritos, así como las principales relaciones entre ellos, como se muestra en la *figura1*.

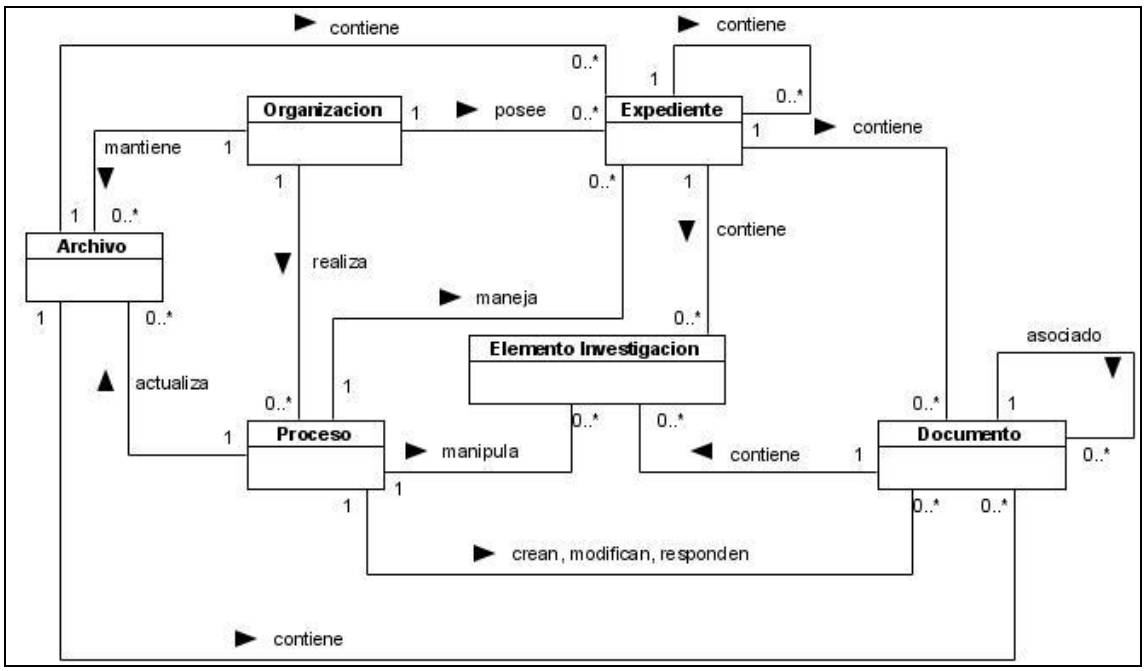


Figura 1. Modelo de Dominio.

2.4 Modelo de la Propuesta de Solución.

A partir de este punto se comienza a modelar la solución propuesta. Para ello se identifican sus requisitos, tanto funcionales como no funcionales, y se modelan los funcionales en términos de casos de uso del sistema.

2.4.1 Requisitos Funcionales.

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir.

R1 Cambiar Estado de Documento.

1.1 Recibir la petición de cambio de estado de un documento dada una acción determinada.

1.2 Verificar que se puede efectuar el cambio solicitado dadas las restricciones impuestas para cada transición de estado.

1.3 Ejecutar el cambio de estado o, en caso contrario, prevenir la realización de la acción en cuestión.

1.4 Notificar a los interesados de la ocurrencia de la transición de estado del documento en cuestión.

R2 Sustanciar Expedientes.

2.1 Registrar relación de pertenencia de un documento a un expediente.

2.2 Registrar relación de composición entre un expediente "padre" y un expediente "hijo".

2.3 Registrar relación de asociación de elementos investigativos a un expediente.

R3 Estandarizar el Modelo General de Relaciones de un Documento.

3.1 Permitir la asociación de documentos a otros documentos, como base para la relación “solitud-respuesta”.

3.2 Registrar relación de asociación de documentos a un expediente.

R4 Estandarizar el Modelo General de Relaciones de Elementos Investigativos.

4.1 Relacionar elementos investigativos con contenedores asociados (documentos, expedientes, etc.).

4.2 Relacionar elementos investigativos entre sí.

R5 Crear extensiones comunes para elementos investigativos.

5.1 Crear extensiones comunes de elementos investigativos, tales como: armas, objetos, vehículos y personas.

R6 Cambiar Estado de Elementos Investigativos.

6.1 Recibir la petición de cambio de estado de un elemento dada una acción determinada.

6.2 Verificar que se puede efectuar el cambio solicitado dadas las restricciones impuestas para cada transición de estado.

6.3 Ejecutar el cambio de estado o, en caso contrario, prevenir la realización de la acción en cuestión.

6.4 Notificar a los interesados de la ocurrencia de la transición de estado del elemento en cuestión.

R7 Estandarizar el Manejo de la Información Histórica de Elementos Investigativos.

7.1 Proporcionar un modelo general para el manejo de información histórica de los elementos investigativos.

R8 Notificar Ocurrencia de Eventos a Elementos Interesados.

8.1 Al ser ejecutada una acción que modifica los datos de un elemento en particular, enviar una notificación a los interesados en monitorear la ocurrencia de dicho evento.

2.4.2 Requisitos No Funcionales.

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Son importantes para que clientes y usuarios puedan valorar las características no funcionales del producto, puesto que las propiedades no funcionales, como cuán usable, seguro, conveniente y agradable es el sistema, pueden marcar la diferencia entre un producto bien aceptado y uno con poca aceptación.

Existen varios tipos de requisitos no funcionales. A continuación se presentan los correspondientes a la solución propuesta.

Software

- La solución requerirá la Maquina Virtual de Java en su versión 1.4 o superior.

Diseño e implementación

- Se utilizará el lenguaje Java o extensiones de este.
- Como estándar de codificación se utilizarán las convenciones de código inherentes del lenguaje java *(Java Code Conventions***)**.
- Se debe generar la documentación de todas las clases, métodos y recursos creados.
- La solución debe ser extensible de acuerdo a las necesidades de uso.

- La solución debe permitir una configuración flexible para satisfacer las necesidades de diferentes negocios.
- La solución debe poder integrarse con *frameworks* ampliamente utilizados como *Spring*.
- La solución deberá poder ejecutarse en diferentes ambientes de desarrollo.

Confiabilidad

- La solución debe brindar garantía de un tratamiento adecuado de las excepciones. Además todas las operaciones de importancia deben ser mostradas en forma de *logs* en la consola *Java*.

Portabilidad

- ❖ La solución debe ser independiente de la plataforma en que se utilice, propiedad que hereda del lenguaje utilizado.

Seguridad

- ❖ La solución debe delegar el aspecto de la seguridad en otros componentes de cada aplicación “cliente” específica.

Usabilidad

- ❖ La solución debe ser de fácil comprensión, configuración y utilización por los desarrolladores en la confección de sistemas policiales desarrollados en Java.

Soporte

- ❖ La solución debe brindar garantía de funcionamiento, adaptación y configuración. Adicionalmente se debe proveer toda la documentación correspondiente y el código fuente para que se potencie su extensión y evolución posterior.

2.4.3 Actores del Sistema.

Los actores de un sistema son agentes externos: aquellas personas o sistemas que interactúan con él. En la siguiente tabla se describen los actores de la solución propuesta.

Tabla 1. Actores del sistema.

Actor	Justificación
Desarrollador	Es el encargado de realizar todas las operaciones de configuración de los componentes.
Componente Externo	Representa a los recursos o clases del sistema que esté utilizando los componentes e interactuarán directamente con estos.

2.4.4 Diagrama de Casos de Uso del Sistema.

Los casos de uso son funcionalidades, o fragmentos de ellas, correspondientes a los procesos automatizados. En ellos se describe la secuencia determinada de eventos que realiza un actor en interacción con la aplicación. En la *figura2* se muestra la representación UML de los Casos de Uso de la propuesta de solución.

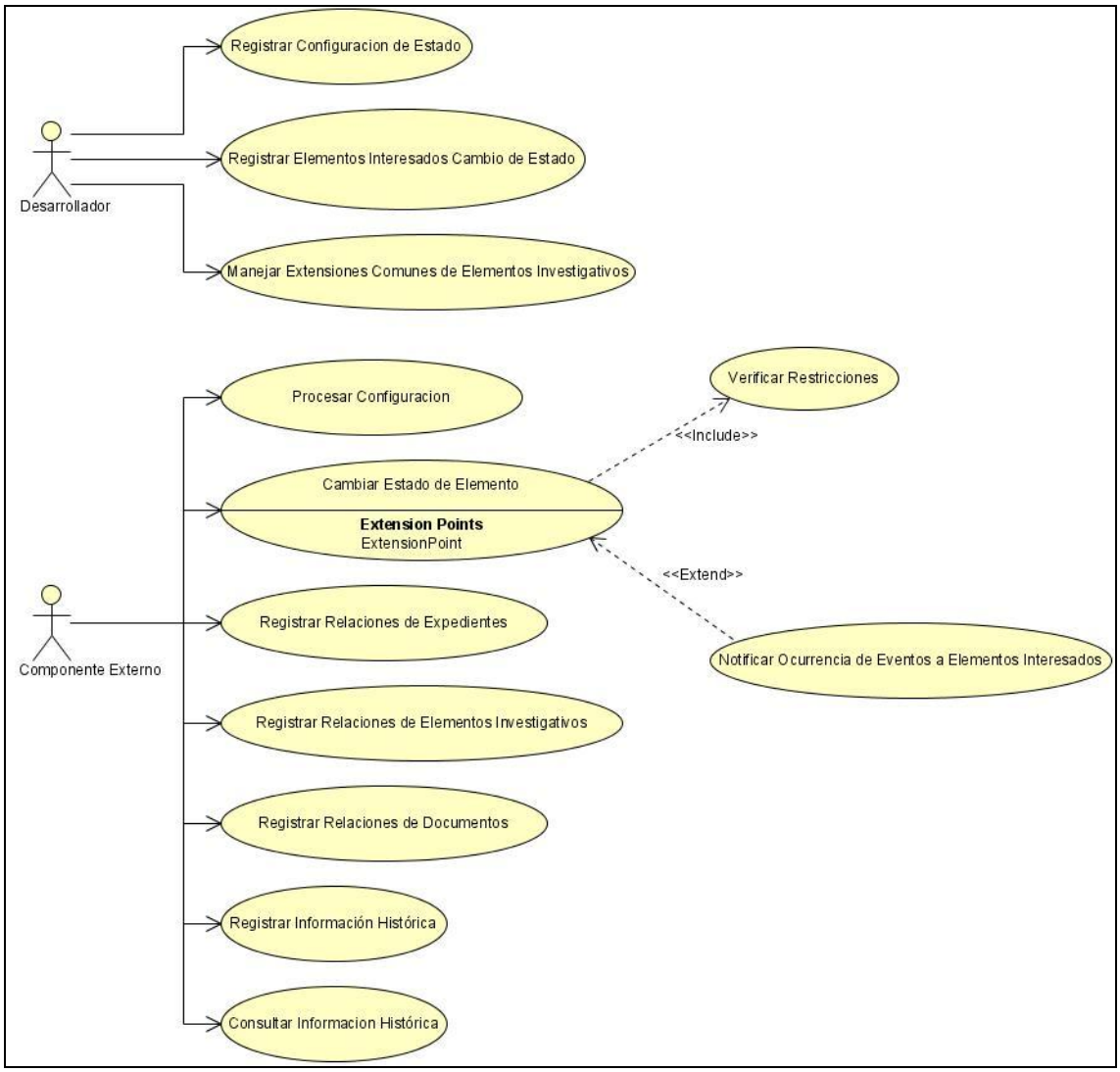


Figura2. Diagrama de Casos de Uso del Sistema.

2.4.5 Descripción de los Casos de Uso.

En este epígrafe se proporciona una descripción resumida de los casos de uso representados anteriormente.

- **CU Registrar Elementos Interesados Cambio de Estado.**

  El caso de uso inicia cuando el actor registra uno o varios elementos interesados en recibir una notificación cuando ocurra un cambio de estado en algún elemento en particular. El actor introduce los registros pertinentes en el archivo de configuración correspondiente a las definiciones y transiciones del estado en cuestión. El sistema valida las configuraciones introducidas indicando algún error en caso de que exista, finalizando así el caso de uso.

- **CU Registrar Configuración de Estado.**

  El caso de uso comienza cuando el actor accede a la opción que le permite configurar todos los datos asociados a un estado, que son necesarios para poder ejecutar una transición determinada. Por cada transición introduce la acción que la desencadenará, el nuevo estado al que transitará el elemento, y las restricciones que deben cumplirse para poder efectuarla satisfactoriamente cumpliendo con los requisitos del negocio. El sistema valida los datos introducidos indicando algún error en caso de que exista, finalizando así el caso de uso.

- **CU Procesar Configuración.**

  Este caso de uso comienza cuando el actor desea cargar las definiciones de estado y elementos interesados contenidos en un archivo de configuración. El sistema localiza el archivo y procesa la información contenida en él, obteniendo como resultado un código intermedio. Posteriormente, convierte el código intermedio en el modelo convenido por los desarrolladores para representar el negocio en el mundo objetual. El caso de uso finaliza una vez procesada toda la información, o al reportar algún error ocurrido durante el proceso.

- **CU Cambiar Estado de Elemento.**

  Este caso de uso agrupa las actividades necesarias para efectuar un cambio de estado en un elemento. Inicia cuando se realiza una acción que implica un cambio de estado. El sistema obtiene la configuración correspondiente al estado del elemento en cuestión, verifica que exista una acción que se corresponda con la realizada sobre el documento y que implique un cambio de estado, chequea las restricciones impuestas a la transición (ver **CU Verificar Restricciones**), y ejecuta el cambio de estado, notificando a los elementos interesados (ver **CU Notificar**

Ocurrencia de Eventos a Elementos Interesados) (en caso de que existan) de la transición realizada. En caso de no ser posible ejecutar la transición debido a alguna razón o error ocurrido, informará que no es posible la realización de la acción en cuestión. En cualquiera de los dos casos, finaliza el caso de uso.

- **CU Verificar Restricciones.**

  Inicia cuando el caso de uso base requiere realizar la verificación de las restricciones impuestas para efectuar una transición de estado determinada. El sistema obtiene las restricciones definidas en el archivo de configuración, y realiza las comprobaciones necesarias para cada una de las restricciones. Una vez terminado el proceso de verificación informa el resultado del mismo: puede o no efectuarse el cambio de estado. En caso que haya ocurrido algún error durante el proceso, este será informado, no permitiendo realizar la transición. En cualquier caso, finaliza el caso de uso.

- **CU Notificar Ocurrencia de Eventos a Elementos Interesados.**

  Este caso de uso engloba la secuencia de eventos que ocurren cuando se realiza una modificación a un elemento o a su estado, y se desea notificar este hecho a uno o varios elementos interesados. Inicia cuando el CU base realiza una acción que requiere ser informada a los elementos pertinentes. Posee dos secciones fundamentales: **Notificar Cambio de Estado,** referente a las notificaciones requeridas al efectuarse una transición de estado; y **Notificar Modificaciones**, que engloba la notificación de otras modificaciones generales realizadas. En ambos casos el sistema obtiene los elementos interesados en ser notificados (en caso de que existan) al ocurrir la acción en cuestión y notifica el evento a cada uno de ellos, informando algún error o requisito que lo impida. Finaliza así el caso de uso.

- **CU Manejar Extensiones Comunes de Elementos Investigativos.**

  Inicia cuando un desarrollador decide crear extensiones comunes para elementos de investigación tales como armas, objeto, vehículos y personas. El sistema le proporciona una distribución predeterminada de estos, con determinados atributos comunes, que ya incluyen el manejo del "flujo de vida" mediante el manejo de estados, y están suscritos al proceso de envío de notificaciones a interesados. El actor realiza las configuraciones necesarias para personalizar dicha distribución, adiciona elementos interesados (**CU Registrar Elementos Interesados**) y/o

adiciona configuraciones de estado (**CU Registrar Configuración de Estado**), finalizando así el caso de uso.

- **CU Registrar Relaciones de Expedientes.**

  El caso de uso inicia cuando el actor decide sustanciar un expediente relacionando elementos investigativos, documentos u otros expedientes. El sistema le brinda un marco predeterminado que soporta estas relaciones, con atributos básicos y comunes, y adicionalmente la opción de personalizarlas. El actor utiliza o personaliza las relaciones necesarias, quedando estandarizada la manera de adicionar y tratar las relaciones de un expediente determinado, finalizando de esta forma el caso de uso.

- **CU Registrar Relaciones de Elementos Investigativos.**

  El caso de uso inicia cuando el actor decide adicionar relaciones de elementos investigativos con contenedores asociados, o con otros elementos investigativos. El sistema le brinda un marco predeterminado que soporta estas relaciones, con atributos básicos y comunes, y adicionalmente la opción de personalizarlas. El actor utiliza o personaliza las relaciones necesarias, quedando estandarizada la forma de adicionar y tratar las relaciones entre elementos de este tipo, y estableciendo un marco para tratar de manera general la asociación de elementos de investigación a contenedores, finalizando de esta forma el caso de uso.

- **CU Registrar Relaciones de Documentos.**

  El caso de uso inicia cuando el actor decide adicionar relaciones a un documento con elementos investigativos u otros documentos. El sistema le brinda un marco predeterminado que soporta estas relaciones, con atributos básicos y comunes, y adicionalmente la opción de personalizarlas. El actor utiliza o personaliza las relaciones necesarias, quedando estandarizada la forma de adicionar y tratar las relaciones a un documento determinado y estableciendo un marco para tratar de manera general la respuesta al mismo, finalizando de esta forma el caso de uso.

- **CU Registrar Información Histórica.**

  Este caso de uso inicia cuando ocurre un cambio en un atributo de una entidad y es necesario almacenar su valor anterior. El sistema detecta que se ha producido un cambio en un atributo de interés y obtiene la antigua información que este contenía así como toda la información relacionada.

- **CU Consultar Información Histórica.**

  Este caso de uso inicia cuando el actor necesita obtener todos los cambios realizados a un objeto, o a un atributo de un objeto, en un momento determinado. El sistema obtiene las modificaciones realizadas en esa fecha, finalizando así el caso de uso.

2.5 Priorización de Casos de Uso.

La priorización de casos de uso es una actividad inicial importante para la selección de los escenarios y/o casos de uso que se realizaran en una iteración. Como resultado de la realización de esta actividad, fueron seleccionados determinados casos de uso de los descritos en el capítulo anterior, atendiendo a que representan funcionalidades significativas centrales y que poseen una cobertura arquitectónica sustancial, debido a que necesitan muchos elementos arquitectónicos, o que tensionan o ilustran un punto delicado de la arquitectura. Los restantes casos de uso serán desarrollados en versiones posteriores de los componentes, ya que necesitan de los casos de uso priorizados para su correcta realización. A continuación se muestran los casos de uso seleccionados.

- CU Registrar Elementos Interesados Cambio de Estado.
- CU Registrar Configuración de Estado.
- CU Procesar Configuración.
- CU Cambiar Estado de Elemento.
- CU Verificar Restricciones.
- CU Notificar Ocurrencia de Eventos a Elementos Interesados.
- CU Manejar Extensiones Comunes de Elementos Investigativos.
- CU Registrar Relaciones de Expedientes.
- CU Registrar Relaciones de Elementos Investigativos.
- CU Registrar Relaciones de Documentos.

2.6 Conclusiones

En este capítulo se realizó una descripción de la propuesta de solución a través de la modelación del dominio propuesto; el planteamiento de los requisitos funcionales y no funcionales de los componentes que se van a desarrollar y la modelación de sus funcionalidades en términos de casos de uso de sistema. A partir de este punto se comenzará a construir el sistema que constituye la propuesta de solución.

3 CAPÍTULO

CONSTRUCCIÓN DE LA PROPUESTA DE SOLUCIÓN

3.1 Introducción.

En el presente capítulo se agrupan los casos de uso descritos en el capítulo anterior en tres componentes teniendo en cuenta la afinidad y cohesión de las funcionalidades que se necesita implementar. Se estructuran los requisitos por cada uno de los componentes para facilitar su preparación, modificación y en general su mantenimiento. Se realiza el análisis de cada uno de los componentes a desarrollar y finalmente se expone el diseño de la solución propuesta.

3.2 Modelo de Análisis.

Durante el análisis se describe lo que requiere el cliente, se establece la base para la creación de un diseño de software y se definen un conjunto de requisitos que puedan ser validados, obteniendo una visión del sistema que se preocupa de dar cumplimiento a los requisitos funcionales del mismo. *El modelo de análisis ilustra información, funcionamiento y comportamiento dentro del contexto de los elementos del modelo de objetos.* (16). A continuación se presentan varias vistas de clases resultantes del análisis del dominio de cada uno de los componentes a desarrollar, como una primera aproximación al diseño de la solución propuesta, y la función que cumple cada una de estas clases.

3.2.1 Gestión de Estados.

La Gestión Documental y el "ciclo de vida" de los Elementos Investigativos están basados en hacer transitar a un elemento de un estado a otro, al ocurrir una acción determinada y siempre que se cumplan un conjunto de restricciones. El estado en un elemento significa una característica única que se corresponde con su ubicación en un momento determinado dentro del proceso de negocio.

El componente para manejar las transiciones de estados engloba los casos de uso ***Registrar Configuración de Estado***, **Registrar Elementos Interesados Cambio de Estado, Procesar Configuración*, *Cambiar Estado de Elemento***, ***Verificar Restricciones*** y la sección **Notificar Cambio de Estado** del **CU Notificar Ocurrencia**

de Eventos a Elementos Interesados. De esta forma se garantiza el cumplimiento de los requisitos funcionales *R1* y *R6*. A continuación se muestra una vista de clases resultantes del análisis de este componente.

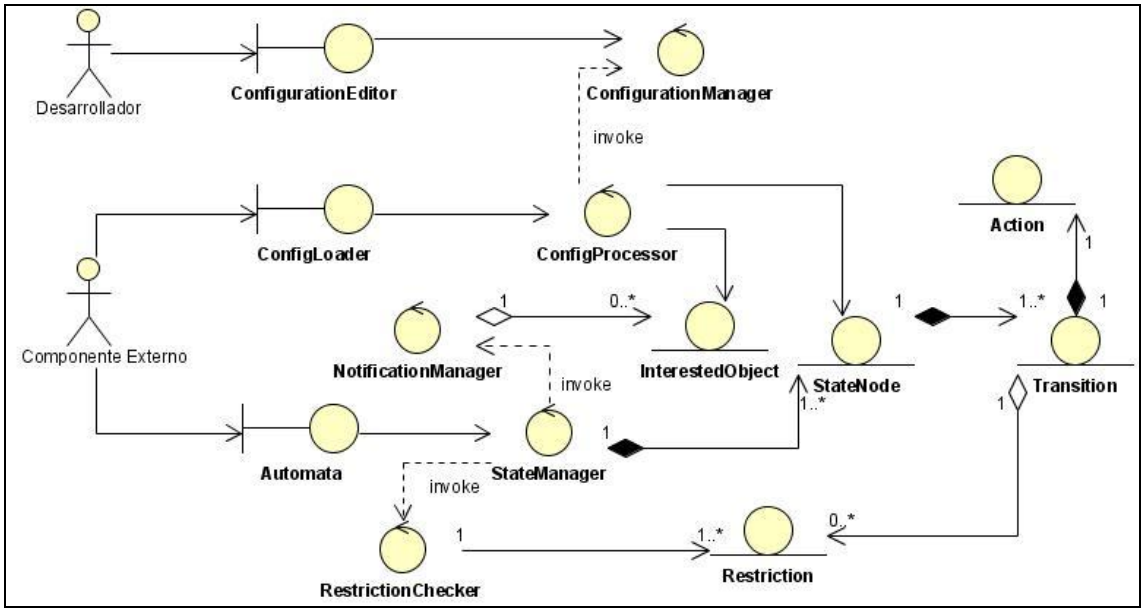


Figura3. Vista de Clases de Análisis. Componente Gestión de Estados.

Tabla 2. Clases de Análisis del Componente de Gestión de Estados.

Nombre	Clasificación	Función
ConfigurationEditor	Interfaz	Permitir al desarrollador configurar los estados y transiciones por los que transitará un elemento determinado.
ConfigurationManager	Control	Encapsular la validación y carga de las configuraciones plasmadas en los archivos de configuración.
ConfigLoader	Interfaz	Brindar funcionalidades para leer los archivos de configuración y procesar toda la información contenida en ellos.
ConfigProcessor	Control	Convertir toda la información proporcionada por el *ConfigurationManager* hacia el modelo objetual representativo del dominio del problema.
StateManager	Control	Procesar toda la información y realizar el cambio de estado de un elemento, o prevenir el mismo.
NotificationManager	Control	Enviar notificación del cambio ejecutado sobre un elemento determinado a los elementos interesados en la ocurrencia de este evento.

RestrictionsChecker	Control	Verificar que se cumplan todas las restricciones para ejecutar una transición de un estado a otro.
Automata	Interfaz	Especificar el contrato de uso del componente. Constituye el medio que expondrá el componente para ser utilizado por otros componentes o sistemas.
InterestedObject	Entidad	Constituye un concepto abstracto para representar cualquier elemento interesado en ser notificado al ocurrir un cambio de estado. Se incluyó en esta vista para facilitar una mejor comprensión del modelo.
StateNode	Entidad	Representa el estado en que puede encontrarse un elemento. Está compuesto, fundamentalmente por transiciones, que indican los posibles cambios de estado que pueden ocurrir.
Transition	Entidad	Almacena la información referente a un cambio de un estado a otro. Contiene la acción que lo desencadenará, y el nuevo estado al que transitará, así como las restricciones que deben ser cumplidas para efectuar la transición.
Restriction	Entidad	Encapsula la información de una condición que debe cumplirse para poder ejecutar una transición de estados.
Action	Entidad	Corresponde a una acción específica que provoca un cambio de estado.

Los paquetes son un mecanismo de organización de elementos que proporcionan una vista general del modelo, agrupando este en elementos o componentes más pequeños que colaboran entre sí. Sus contenidos deben estar fuertemente relacionados, minimizando las dependencias entre ellos y garantizando de esta forma una alta cohesión y un bajo acoplamiento. Se deben crear sobre la base de los requerimientos funcionales y el dominio del problema. En la *figura4* se muestra la estructura inicial de paquetes obtenida para organizar los elementos que permitirán la realización de los casos de uso contenidos en este componente. Esta organización se realizó teniendo en cuenta las funcionalidades que debe soportar.

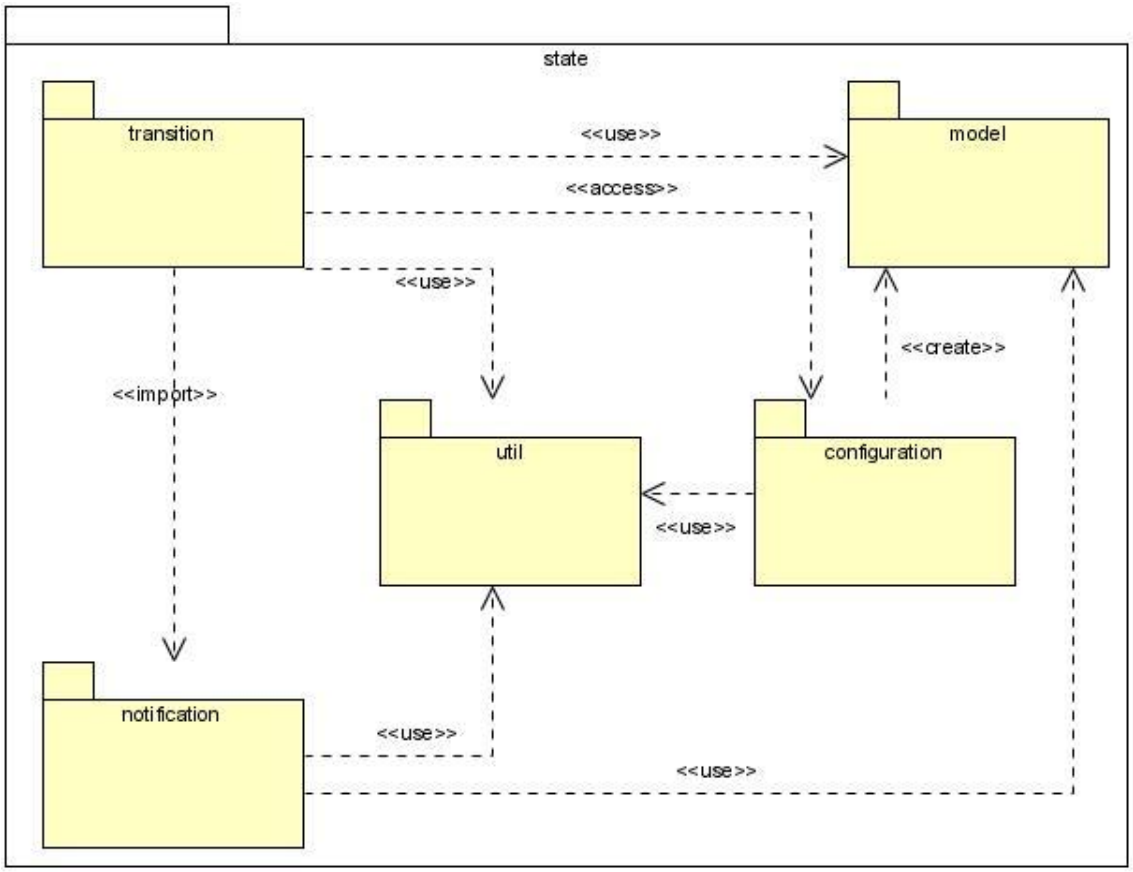


Figura4. Vista de Paquetes Análisis. Componente Gestión de Estados.

Tabla 3. Paquetes. Componente Gestión de Estados.

Nombre	Función
transition	Agrupa todas las clases y elementos necesarios para realizar una transición de estado. Incluye además el chequeo de restricciones, y detecta la acción que puede causar un cambio de estado.
configuration	Encapsula toda la lógica de creación y actualización de las configuraciones de estado. Creando todas las configuraciones en ficheros **xml** para permitir un alto grado de flexibilidad.
model	Contiene las clases del dominio en las que estará basado todo el componente. Sus valores son obtenidos al ser procesadas las configuraciones.
util	Constituye un paquete de servicios ya que contiene un conjunto de clases y coherentes acciones relacionadas funcionalmente, que se utilizan en varios escenarios de los casos de uso.
notification	Engloba todos los componentes necesarios para notificar la ocurrencia de determinados eventos sobre un elemento.

Los diagramas de colaboración destacan la organización de los objetos que participan en una interacción, y la secuencia de acciones que se realizan en ella. A continuación se muestran los diagramas de colaboración correspondientes a los casos de uso contenidos en este componente.

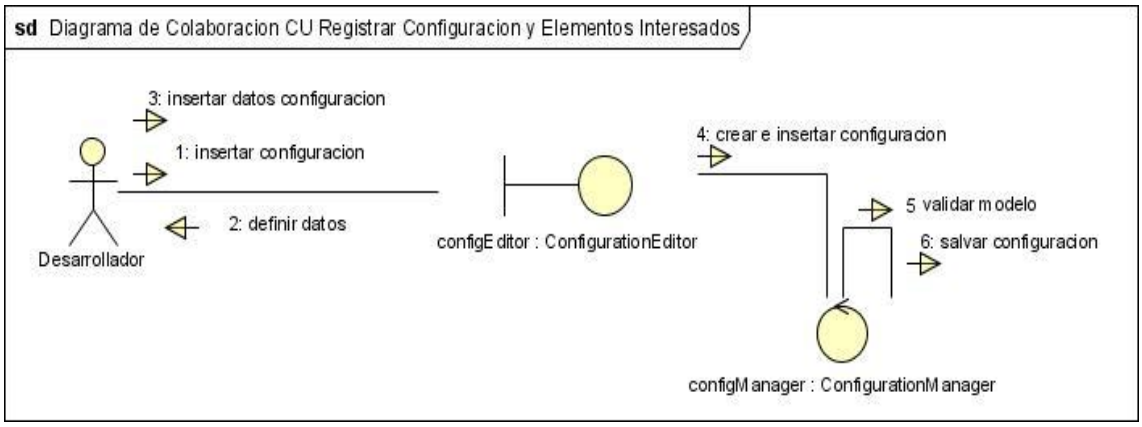


Figura5. Diagrama de Colaboración. CU Registrar Configuración de Estado y Registrar Elementos Interesados Cambio de Estado.

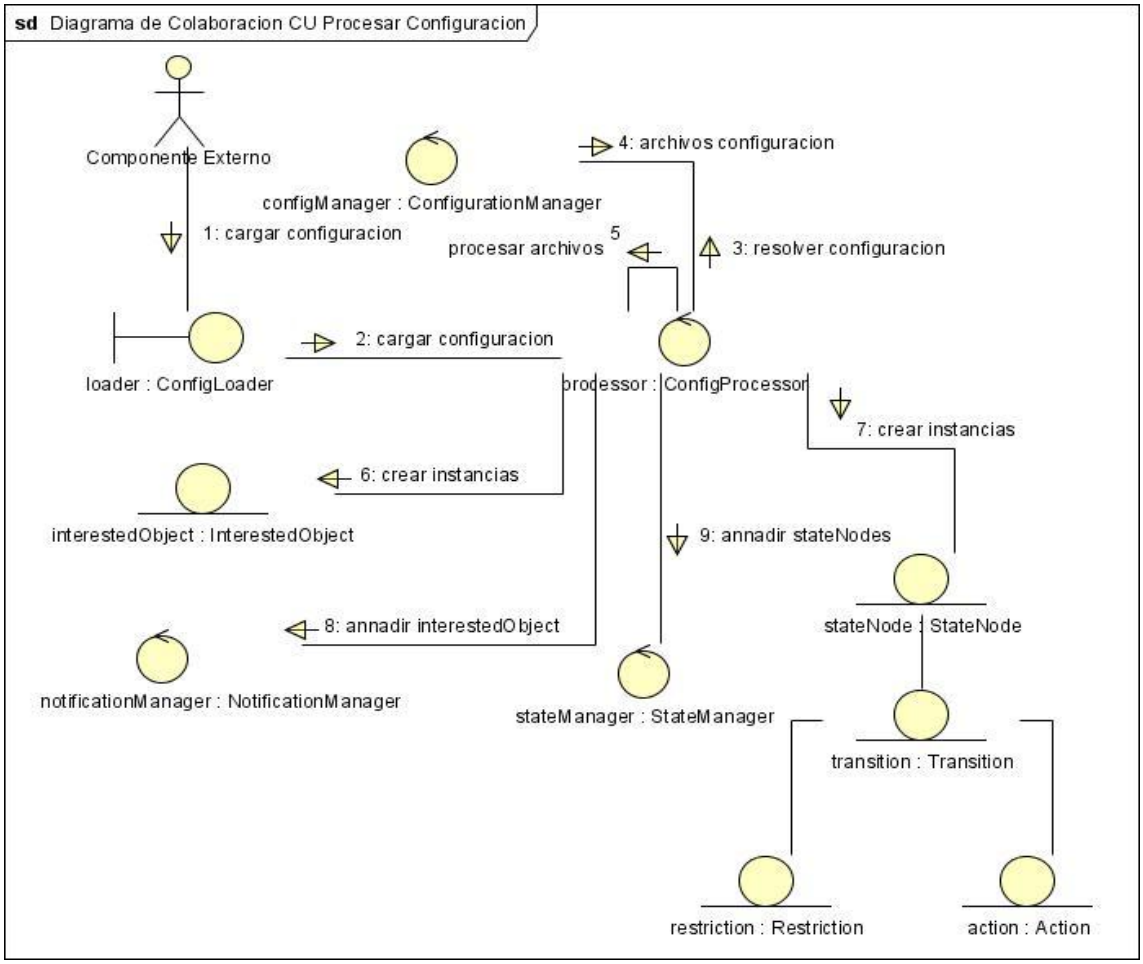


Figura6. Diagrama de Colaboración. CU Procesar Configuración.

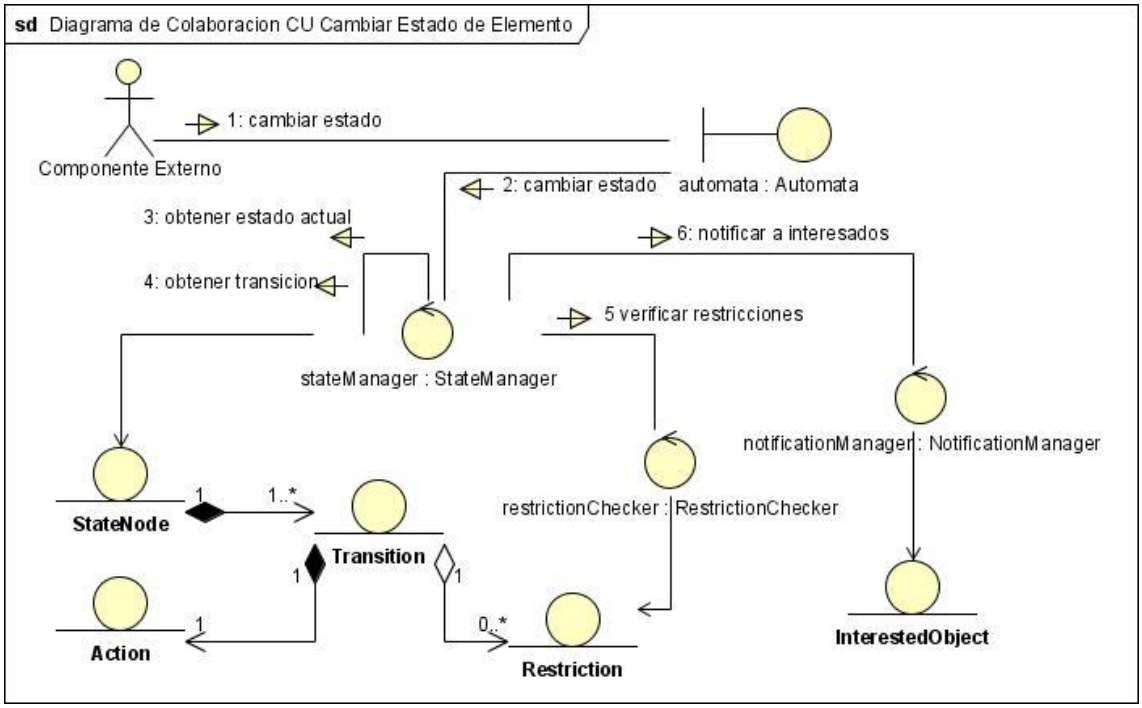


Figura7. Diagrama de Colaboración. CU Cambiar Estado de Elemento, CU Verificar Restricciones y CU Notificar Ocurrencia de Eventos a Elementos Interesados.

3.2.2 Manejo y Estandarización de Conceptos de Investigación.

Dentro de la clasificación de conceptos involucrados en la investigación se encuentran todos aquellos elementos que constituyen pilares fundamentales en este proceso, entiéndase expedientes, documentos y elementos investigativos. Estos conceptos se relacionan entre sí para cumplir su objetivo primordial de contribuir al desarrollo de la investigación. Cada concepto está envuelto en determinados procesos de negocio propios, o correspondientes a sus contenedores. Su creación y actualización constante es tarea primordial para lograr poseer referencias actualizadas y confiables, lo que contribuye a disminuir los períodos de investigación de un hecho, debido a que se pueden obtener determinadas conclusiones a partir del análisis estadístico.

El componente para manejar los conceptos de investigación engloba los casos de uso **Registrar Relaciones de Expedientes**, **Registrar Relaciones de Elementos Investigativos**, **Registrar Relaciones de Documentos**. Garantizando de esta forma el cumplimiento de los requisitos funcionales *R2*, *R3*, y *R4*. A continuación se detallan los procesos objeto de investigación, que conciernen a estos elementos.

3.2.2.1 Elementos Investigativos.

Constituyen una unidad atómica dentro de los procesos investigativos desarrollados en las organizaciones policiales. Esto se debe a que representan la manera común de recabar toda la información de objetos inanimados o personas asociadas a un hecho delictivo y/o punible. Son objeto de múltiples pruebas con el fin de arribar a conclusiones científicas que esclarezcan la falta cometida. Pueden estar relacionados entre sí, información que es registrada también, ya que proporciona una fuente valiosa de datos para referencias futuras. Poseen "vida" dentro de la organización ya que su estado depende de determinadas acciones que son realizadas sobre ellos.

3.2.2.2 Documentos.

Durante las actividades cotidianas en las instituciones policiales, son creados, modificados y archivados documentos que divergen tanto en su tipo y finalidad, como en la información que contienen. Constituyen la documentación de todas las acciones realizadas en el transcurso de una investigación y la fuente legal principal para mantener la comunicación dentro de la organización, y con entidades externas a ella. Estos documentos pueden estar relacionados con otros documentos, bien porque constituyan la respuesta a una petición realizada previamente, o porque simplemente son tomados como referencia en otros. Además, puede contener elementos investigativos con el objetivo de solicitar la realización de una determinada experticia sobre ellos, o de indicar la ejecución de una acción que varíe su estado dentro del negocio, por solo citar algunos ejemplos.

3.2.2.3 Expedientes.

Generalmente en las instituciones policiales, para cada investigación es abierto un expediente. Estos constituyen los "archivos" o "carpetas" donde serán ubicados todos los complementos que tributen de alguna manera al desarrollo de la investigación: pueden ser documentos, evidencias, estudios practicados por expertos sobre determinadas sustancias, o sencillamente otros archivos o carpetas. Por tanto, de forma general, los expedientes, constituyen una manera simple de mantener relacionados y agrupados elementos que tributan a un mismo objetivo, y su enriquecimiento o sustanciación garantiza el cumplimiento de dicho objetivo.

Producto de que el componente de Manejo y Estandarización de Conceptos de Investigación, más que un proveedor de funcionalidades, constituye un conjunto coherente y configurable de elementos de diseño que pueden ser usados por los desarrolladores de sistemas policiales, se consideró pertinente no realizar para este

caso un modelo de análisis, y centrarse fundamentalmente en la vista estática desde el punto de vista de diseño.

3.3 Elementos Investigativos Básicos.

Son considerados elementos investigativos todas aquellas entidades que modelan conceptos simples de la vida real y están relacionados a un hecho delictivo en específico. Su función es modelar y representar esta información con el objetivo de recabar toda la información posible para el correcto desarrollo de la investigación.

Este componente proporciona soporte para el caso de uso **Manejar Extensiones Comunes de Elementos Investigativos**, con lo que se garantiza el cumplimiento del requisito funcional *5*.

3.3.1 Extensiones Comunes

Atendiendo a los elementos manejados en los sistemas policiales estudiados, se propone la creación de los elementos descritos a continuación como extensiones comunes del concepto Elemento Investigativo.

- Arma: Representa las posibles armas existentes, tanto blancas como de fuego.
- Vehículo: Constituye una generalización de todos los vehículos, ya sean terrestres, marítimos o aéreos.
- Objeto: Agrupa todos los elementos que por sus características no puedan ser incluidos en las clasificaciones anteriores, tales como plumas, computadores, entre otros.
- Persona: Contiene las características básicas de una persona, representando ese importante concepto sobre el que está basado toda organización policial.

Al igual que el componente de Manejo y Estandarización de Conceptos de Investigación, este componente no será objeto de realización de actividades de análisis.

3.4 Diseño de la Solución Propuesta.

En el presente epígrafe se describirá el diseño de la propuesta de solución, brindando varias vistas con el objetivo de facilitar su comprensión.

3.4.1 Core.

Toda aplicación, sistema o componente posee un conjunto de funcionalidades comunes y de bajo nivel que generalmente brindan soporte para desarrollar las

funcionalidades del negocio, y para toda la arquitectura. Estas funcionalidades son encapsuladas en un paquete que es nombrado ***core*** o ***núcleo***.

Para el desarrollo de los componentes propuestos en esta investigación, es necesario contar con un paquete de este tipo que contenga todas las funcionalidades necesarias para poder dar cumplimiento a los requisitos funcionales y no funcionales, garantizando soporte para actividades que no tributan directamente al negocio, pero que forman parte invariable del flujo de actividades que deben ser realizadas. En la *figura8* se muestra la organización de paquetes del núcleo.

Estos componentes serán utilizados en la confección de sistemas policiales que generalmente serán aplicaciones web, por lo que poseen un contenedor web, donde son cargadas todas las configuraciones. Por esta razón, es necesario contar con un mecanismo que se encargue de procesar todas las configuraciones, e incluirlas en el contexto web de la aplicación en cuestión. Adicionalmente, y previendo posibles limitaciones, se permitirá su uso en aplicaciones que no posean contenedores web, ya que contará con un mecanismo propio para procesar las configuraciones en este caso, ampliando de esta forma, sus posibles utilizaciones y extensiones.

Para lograr un alto grado de reutilización, todos los datos y configuraciones serán encapsulados en ficheros XML y Properties, por lo que se hace necesario contar con un soporte para dar tratamiento a ficheros de este tipo, que sea independiente de otros frameworks y que permita la integración con los más importantes y utilizados de ellos entre los que se puede mencionar *Spring*.

Con el fin de lograr una correcta personalización y funcionamiento de los componentes, se hace necesario imponer determinados esquemas para la confección de estos ficheros xml, por lo que se contará con esquemas de validaciones *xsd* para cada uno de los posibles ficheros existentes, garantizando de esta manera su correcta utilización y configuración. Estos ficheros de validación serán utilizados para certificar la correcta estructura y solidez de cada fichero en el momento de su procesamiento, lo que adiciona un alto grado de robustez y flexibilidad.

Para el procesamiento de las configuraciones y datos contenidos en xml, se utilizará una implementación personalizada para lograr una independencia total del uso de frameworks, y alcanzar de esta manera una alta portabilidad.

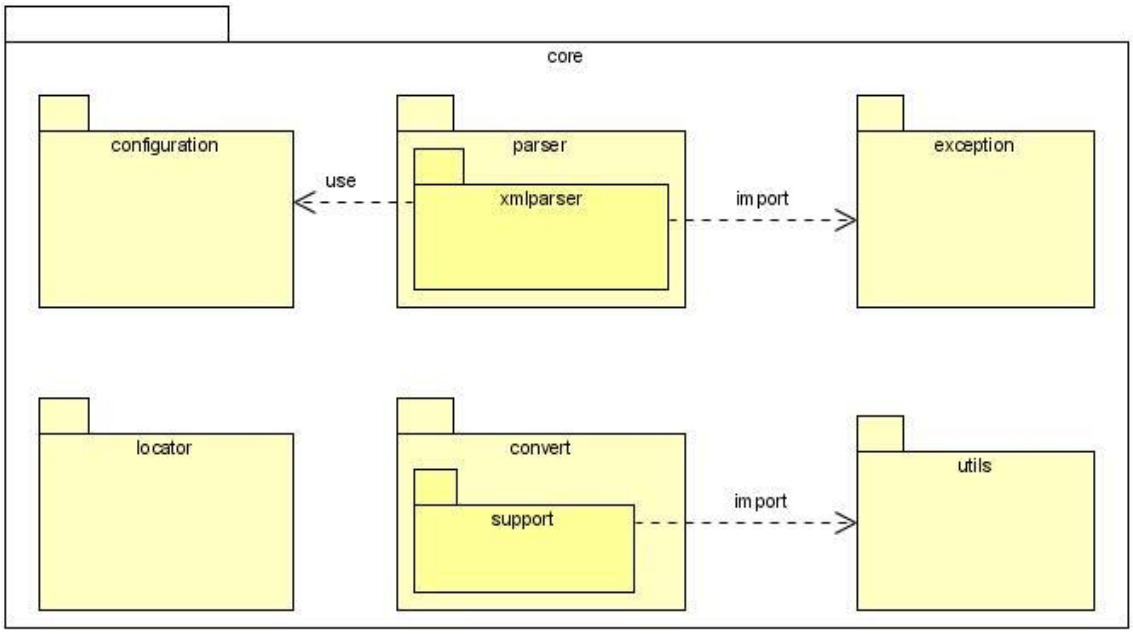


Figura8. Vista de Paquetes del Core.

Tabla 4. Paquetes del Core.

Nombre	Función
parser	Paquete que agrupa toda la lógica relacionada con la conversión de información contenida en un fichero determinado, a un lenguaje intermedio y orientado a objetos, siguiendo determinadas reglas.
xmlparser	Posee todas las funcionalidades requeridas para procesar un archivo xml.
utils	Constituye un paquete de servicios ya que contiene un conjunto de clases y coherentes acciones relacionadas funcionalmente, que se utilizan en varios escenarios de los casos de uso.
convert	Agrupa funcionalidades necesarias para realizar la conversión de elementos que se encuentren en un lenguaje intermedio y orientado a objetos, creados por el parser, hacia objetos de un dominio en específico. No realiza la conversión propiamente dicha, sino que constituye la base para confeccionar los convertidores según las necesidades de cada negocio.
locator	Encapsula la lógica de creación e instanciación de objetos, a partir de configuraciones realizadas en un fichero "*properties*", librando a todas las demás entidades y paquetes de posibles cambios realizados.
exception	Contiene la base de la jerarquía de excepciones que serán utilizadas el desarrollo de los componentes.
configuration	Contiene las configuraciones generales necesarias para el correcto funcionamiento y mantenimiento de los componentes.

A continuación se muestran diferentes vistas de los paquetes más importantes, con el fin de exponer en detalle las funcionalidades contenidas en el núcleo de los componentes, para lograr una mayor comprensión de su principio de funcionamiento y de proporcionar una visión detallada de la estructura del core, así como de la distribución de los componentes y clases según sus funcionalidades, teniendo siempre como premisas mantener un acoplamiento bajo y una cohesión alta a fin de lograr un modelo que permita ampliamente su reutilización y extensibilidad.

❖ El paquete ***parser***.

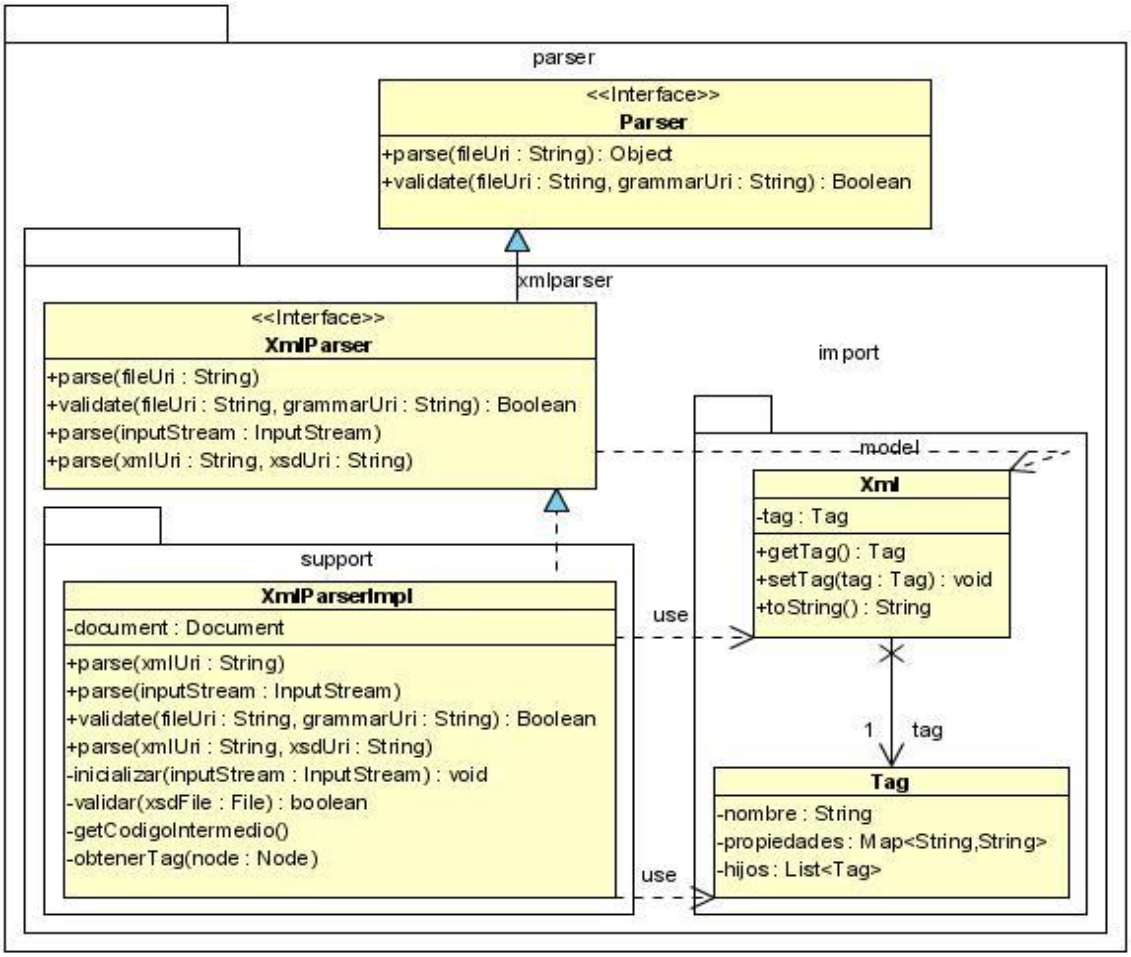


Figura9. Diagrama de Clases de Core. Paquete parser.

Como su nombre lo indica, este paquete está relacionado a todo el procesamiento de documentos o ficheros de un determinado tipo. Para procesar un archivo, se debe contar con uno o varios *analizadores*[3] y la información almacenada en él debe regirse

[3] Un analizador convierte el texto de entrada en otras estructuras (comúnmente árboles), que son más útiles para el posterior análisis y capturan la jerarquía implícita de la entrada.

por una *gramática*[4] determinada. En el caso concreto de esta investigación, el analizador está dedicado a procesar ficheros xml, y la gramática correspondiente está descrita en ficheros *xsd*[5]. El fin perseguido con este paquete de funcionalidades es separar la lógica de procesamiento y validación de los archivos, de su lógica de utilización; es decir, se centra en procesar los ficheros de entrada convirtiendo la información contenida en ellos, en un lenguaje o código intermedio que será procesado posteriormente de acuerdo a la lógica y restricciones del negocio en cuestión. Los componentes que están contenidos en este paquete son detallados a continuación.

Tabla 5. Clases del paquete parser.

Nombre	Función
Parser	Interfaz que provee un contrato para realizar posibles extensiones de analizadores de archivos.
XmlParser	Especifica las funciones básicas necesarias que debe tener un analizador de archivo xml. Es una extensión de la interfaz *Parser*. Todo analizador que sea desarrollado con el objetivo de ser utilizado en este componente debe implementar esta interfaz.
XmlParserImpl	Constituye la implementación concreta del analizador de archivos xml, brindando diferentes facilidades para su uso. Es capaz de procesar cualquier archivo xml, solamente cumpliendo con las normas elementales de formación para documentos de este tipo.
Tag	Constituye la base del lenguaje al que será llevado el archivo xml procesado por el analizador. Para ello contiene un nombre, una lista de propiedades, y una lista de Tag hijos, con lo cual se logra la definición recursiva requerida.
Xml	Es la representación de un archivo xml en leguaje orientado a objetos. Se corresponde a un lenguaje intermedio entre la información contenida en el fichero, y el uso que le será dado.

[4] Conjunto finito de reglas que describen toda la secuencia de símbolos pertenecientes a un lenguaje específico.

[5] Documentos esquema (usualmente con extensión .xsd de *XML Schema Definition,* XSD). Son utilizados para describir la estructura y las restricciones del contenido de los documentos XML de una forma muy precisa.

❖ El paquete **convert**.

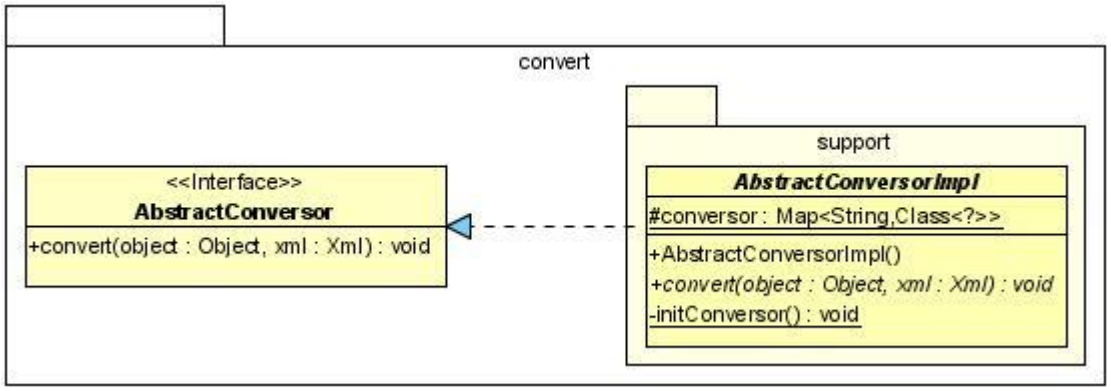


Figura10. Diagrama de Clases del Core. Paquete convert.

Provee los elementos básicos necesarios para realizar las conversiones del código intermedio entregado por el parser de xml, hacia el modelo de objetos más conveniente para el negocio. Su función es encapsular las funcionalidades requeridas por todos los convertidores personalizados, posibilitando de esta forma una mayor limpieza, centralización, y mantenibilidad de los mismos. Los componentes que están contenidos en este paquete son detallados a continuación.

Tabla 6. Clases del paquete convert.

Nombre	Función
AbstractConversor	Constituye el contrato de implementación para todas las clases que realicen la conversión de código intermedio que cumpla con las condiciones impuestas en el paquete *parser*, y el contrato de uso para todos los componentes o clases donde sea requerido.
AbstractConversorImpl	Implementación de la interface *AbstractConversor*, que constituye obligatoriamente la clase padre de todos los convertidores que sean desarrollados. Provee determinados tipos de datos que son necesarios para realizar la conversión.

❖ El paquete **locator**.

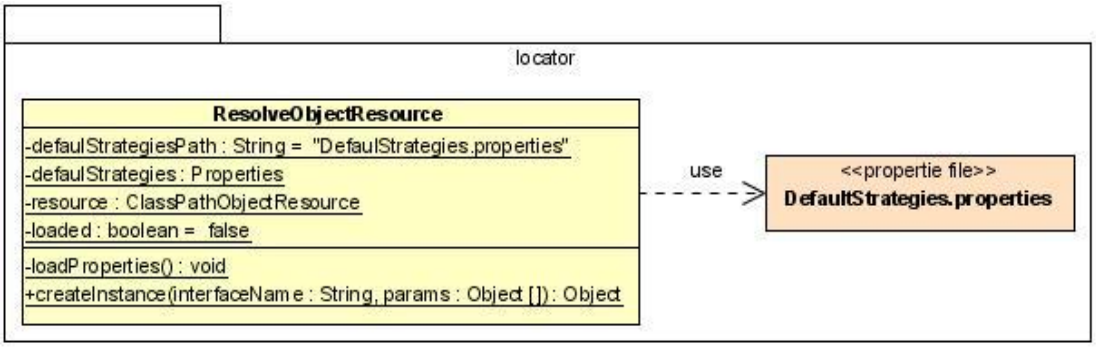


Figura11. Diagrama de Clases de Core. Paquete Locator.

Está diseñado para obtener los recursos indicados en un archivo de configuración, y hacer transparente para el usuario el proceso de instanciación concreta de cada interfaz, lo cual permite cambiar la realización de la interfaz en cuestión solo modificando la configuración, y que el resto del código no se vea afectado. Los componentes que están contenidos en este paquete son detallados a continuación.

Tabla 7. Componentes paquete locator.

Nombre	Función
ResolveObjectResource	Clase estática encargada de cargar la configuración contenida en el fichero properties, y de instanciar cada clase devolviendo una nueva instancia *enmascarada* por la interfaz correspondiente.
DefaultStrategies.properties	Fichero de configuración en el que están contenidas todas las implementaciones por defecto de las interfaces utilizadas.

- El paquete **utils**.

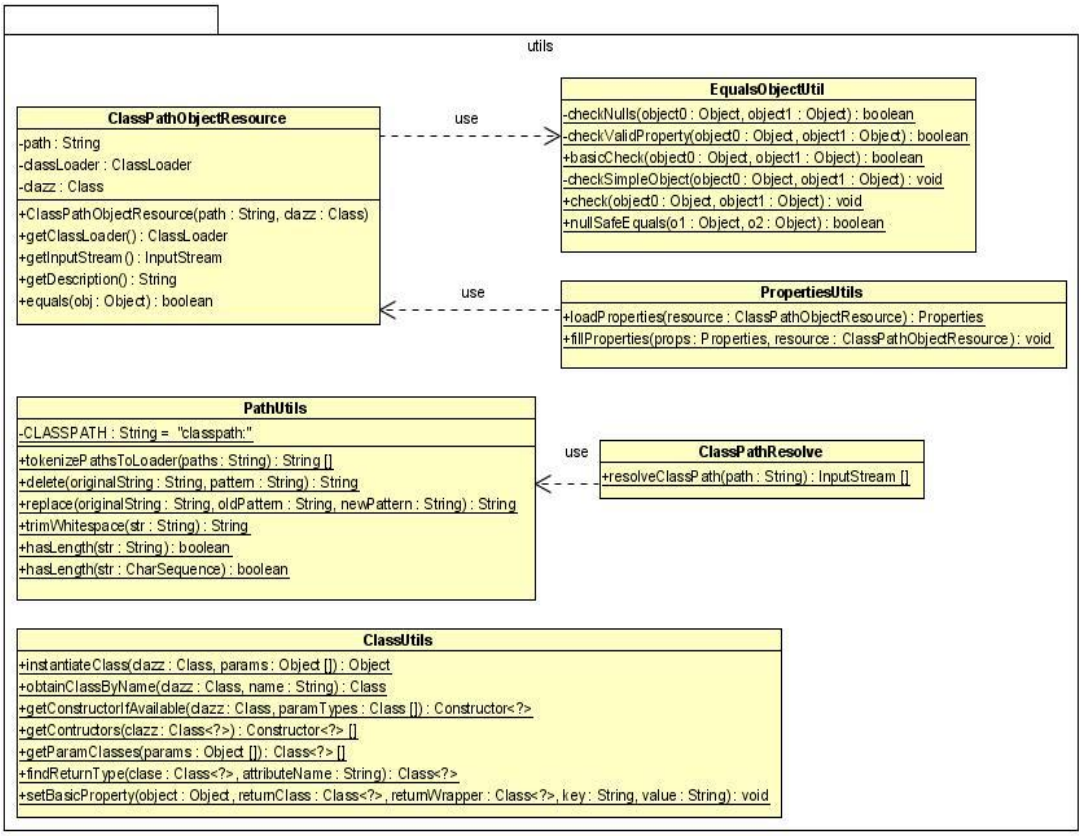


Figura12. Diagrama de Clases de Core. Paquete utils.

Este paquete contiene un conjunto de clases estáticas que proveen varias funciones utilitarias. Tiene como objetivo encapsular por funcionalidades operaciones que no se corresponden con procesos de negocio, pero que son necesarias para dar

cumplimiento a las funcionalidades requeridas. Adicionalmente son altamente utilizadas por varias clases, por lo que son contenidas en un solo lugar, minimizando el acoplamiento de paquetes y favoreciendo la reutilización de código.

Tabla 8. Clases paquete utils.

Nombre	Función
ClassPathObjectResource	Clase estática encargada de cargar los recursos indicados en una dirección determinada del *classpath*.
ClassPathResolve	Se encarga de resolver[6] los recursos que se desean cargar.
ClassUtils	Encapsula funciones utilitarias para el trabajo reflectivo con las clases.
EqualsObjectUtil	Provee un conjunto de funciones encargadas de verificar determinadas restricciones que indican si dos objetos son iguales.
PathUtils	Brinda funciones para trabajar con las direcciones de los recursos que se deseen cargar.
PropertiesUtils	Proporciona funciones utilitarias para cargar propiedades a partir de determinados objetos que encapsulan el trabajo con ficheros.

3.4.2 Componente Gestión de Estados.

Normalmente, toda aplicación que maneje distintos estados para elementos o entidades incorpora entre sus componentes determinados recursos en los que se encapsulan las actividades inherentes a los cambios de estado dentro de la lógica del negocio. Estos recursos pueden corresponderse, entre otras opciones, a la aplicación combinada de varios patrones de diseño como el *State*, *Strategy*, *Flyweight*, entre otros; o con una *máquina de estados*[7].

Las prestaciones que debe brindar el componente de gestión de estados no pueden ser cubiertas mediante la aplicación exclusiva de patrones de diseño, ya que estos solo se encargan de definir una posible solución correcta para un problema de diseño dentro de un contexto dado. Por estas razones se decidió incorporar al modelo una máquina de estados finito para controlar el flujo de estados de los elementos incluidos en este componente.

[6] Se refiere al proceso de localización y procesamiento de un archivo que se encuentra en una dirección determinada.

[7] Modelo de comportamiento de un sistema con entradas y salidas, en donde las salidas dependen no sólo de las señales de entradas actuales sino también de las anteriores.

Una máquina de estados, se encarga de “mover” un elemento (a menudo un objeto) entre diferentes estados, clasificando su comportamiento de acuerdo con los disparadores de transiciones y las guardas de restricciones. (16). Existen determinados *frameworks* para Java que proveen soporte para máquinas de estado. Entre ellos se puede mencionar *Java Finite State Machine* (*FSM)*. Estos *frameworks* poseen gran complejidad de uso, ya que emplean algoritmos y expresiones matemáticas para su configuración y no poseen abundante documentación al respecto. Además, poseen gran dependencia con otros *frameworks*.

Toda la configuración de los estados de cada elemento manejado, serán realizados en archivos XML, posibilitando de esta manera realizar cambios en el flujo de estados sin necesidad de recompilar la aplicación.

La intención perseguida es crear un componente, basado en una máquina de estados, mediante la utilización de varios patrones de diseño para facilitar su extensión, configuración, usabilidad, y que pueda ser utilizada en diferentes sistemas según los requerimientos de cada negocio, por lo que no es posible realizar un diagrama de estados, ya que no está relacionado a un negocio en especifíco.

A continuación se muestran diferentes vistas de los paquetes más importantes, con el fin de exponer en detalle las funcionalidades contenidas, para lograr una mayor comprensión de su principio de funcionamiento, y en el *Anexo1*, se puede observar en toda su expansión el diagrama de clases del componente.

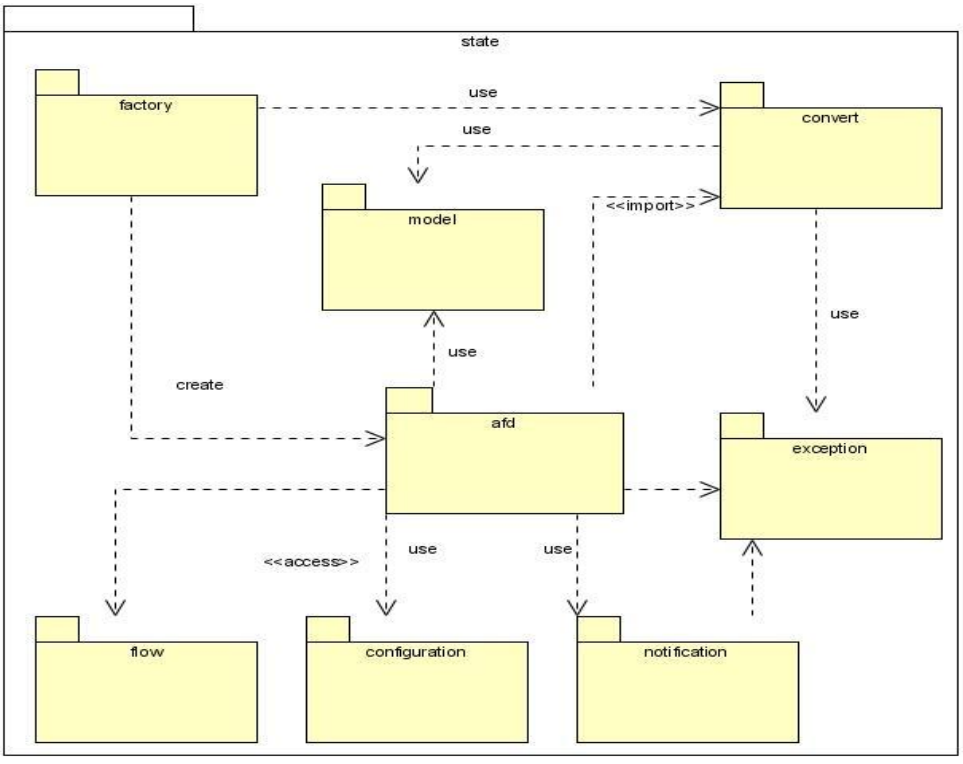


Figura13. Vista de Paquetes. Componente Gestión de Estados.

En el epígrafe correspondiente al análisis de este componente, se expuso una vista inicial de paquetes, la cual se mantuvo hasta este momento, solo adicionándole nuevos paquetes según las funcionalidades lo requirieron.

Tabla 9. Paquetes del Componente de Gestión de Estados.

Nombre	Función
factory	Encapsula la lógica necesaria para aplicar el patrón *Proxy* en la creación de los objetos implicados en la gestión del cambio de estado.
notification	Agrupa todas las funcionalidades necesarias para enviar notificaciones a elementos interesados, una vez que sea ejecutado un cambio de estado.
flow	Contiene las clases e interfaces necesarias para incluir un elemento determinado en un flujo de estados.
convert	Agrupa funcionalidades necesarias para realizar la conversión de elementos que se encuentren en un lenguaje intermedio y orientado a objetos, creados por el *parser*, hacia los objetos del modelo necesarios para realizar los cambios de estado. Se basa en el conversor definido previamente en paquete *convert* del core.
model	Contiene todas las clases necesarias para representar la información definida por el desarrollador en un archivo *xml*, en un modelo orientado a objetos.
afd	Constituye el núcleo del componente de gestión de estados, ya que posee toda la lógica necesaria para realizar las transiciones de estados.
configuration	Agrupa todas las configuraciones necesarias para el correcto funcionamiento, y los archivos *xsd*, para validar los *xml* de configuración.
exception	Contiene una jerarquía de excepciones con el fin de proporcionar información sobre cada posible error, y permitir su corrección.

❖ El paquete **notification**.

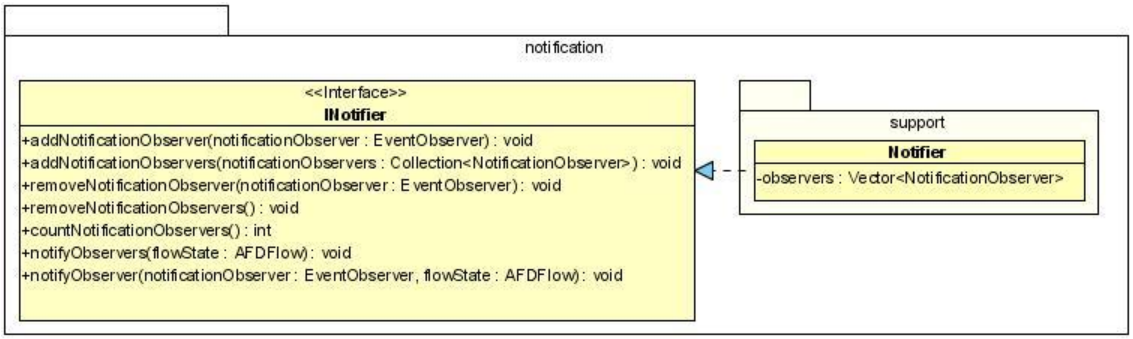


Figura14. Diagrama de Clases. Componente Gestión de Estados. Paquete notification.

Tabla 10. Clases y Recursos del Paquete notification.

Nombre	Función
INotifier	Constituye un contrato de uso e implementación, ya que encapsula la interacción con componentes externos, y define las funcionalidades que deben ser implementadas, en caso que sea requerido su personalización. Ofrece determinados métodos que brindan funcionalidades necesarias para soportar las funcionalidades requeridas para la notificación de un cambio de estado a elementos interesados.
Notifier	Implementación de la interface *INotifier*. Contiene la lógica necesaria para dar soporte al proceso de notificación a elementos interesados una vez ocurrido un cambio de estado. Clase concreta, que debe ser la clase padre de todo elemento que se desee esté implicado en el proceso de notificación, pero no directamente, es decir, no del elemento que esté cambiando de estado, sino del elemento que esté realizando la transición, en este caso, del autómata representado por la clase AFDImpl. De esta manera se logra no imponer restricciones técnicas en el diseño de las entidades del dominio del negocio en cuestión.

- El paquete **afd.**

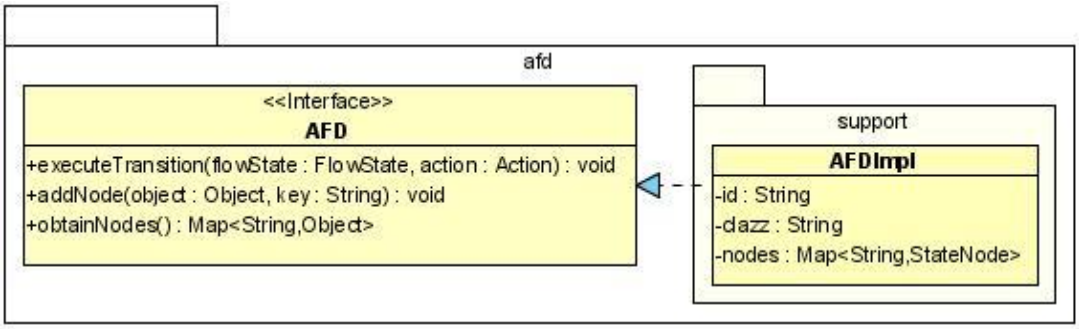


Figura15. Diagrama de Clases. Componente Gestión de Estados. Paquete afd.

Este paquete contiene la clase controladora del componente de gestión de estados, brindando facilidades para su extensión y personalización. Con este propósito, esta clase está *envuelta* por una interfaz que define los contratos de uso e implementación, para aquellos componentes o sistemas que interactúen con el componente, y para las extensiones o personalizaciones que sean requeridas.

Tabla 11. Clases y Recursos del Paquete afd.

Nombre	Función
AFD	Constituye un contrato de uso e implementación, ya que encapsula la interacción con componentes externos y define las funcionalidades que deben ser implementadas, en caso que sea requerida su personalización. Ofrece determinados métodos que brindan funcionalidades necesarias para soportar las funcionalidades requeridas.
AFDImpl	Implementación de la interfaz AFD. Contiene la lógica necesaria para cargar la configuración de los estados, efectuar las transiciones de estado y verificar las restricciones impuestas.

❖ El paquete **factory.**

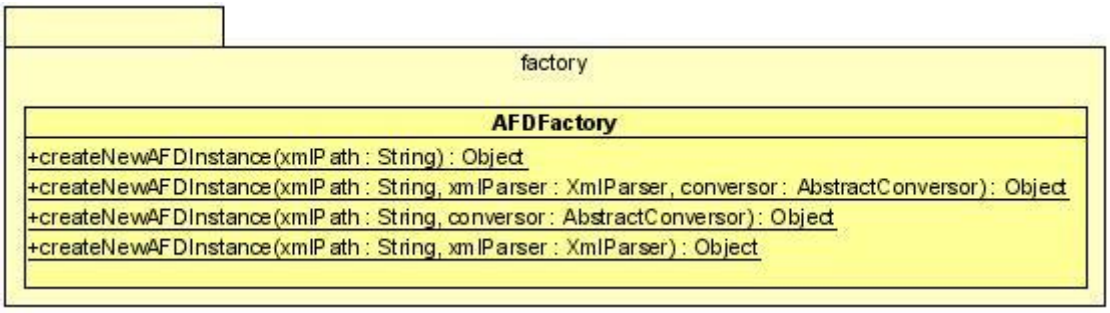


Figura16. Diagrama de Clases. Componente Gestión de Estados. Paquete factory.

Constituye la parte central de la implementación del conocido patrón *factory,* utilizado para cargar la configuracion necesaria de un archivo *properties*, separando de esta manera la lógica concreta de la creacion del AFD, de su uso o instanciación; y adicionando un nivel mas de configuración y flexibilidad al permitir que toda la configuración correspondiente sea definida en un fichero que es posible editar en tiempo real, sin necesidad de compilar nuevamente toda la aplicación. Este paquete contiene solo una clase, que brinda diferentes vías de utilización e instanciación del componente.

❖ El paquete **convert**.

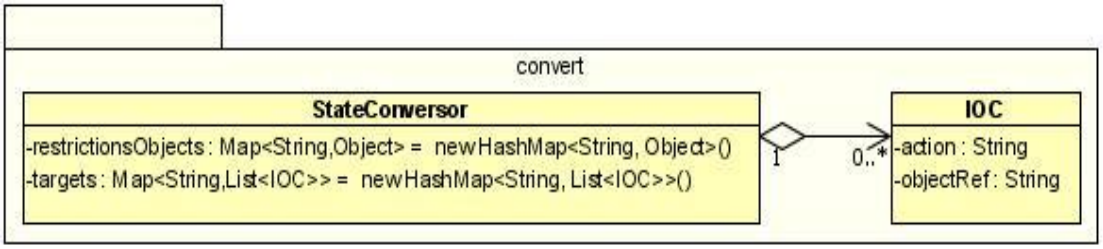


Figura17. Diagrama de Clases. Componente Gestión de Estados. Paquete convert.

Este paquete agrupa las funcionalidades necesarias para realizar la conversión del código intermedio entregado por el parser, al modelo objetual que representa la configuración definida en los arhicvos xml.

Tabla 12. Clases y Recursos del Paquete convert.

Nombre	Función
StateConvert	Clase encargada de realizar la conversión al modelo objetual del código intermedio.
IOC	Clase que tiene como funcionalidad almacenar temporalmente la información necesaria para realizar la inyección de dependencias en los archivos de configuración del componente.

- El paquete **model.**

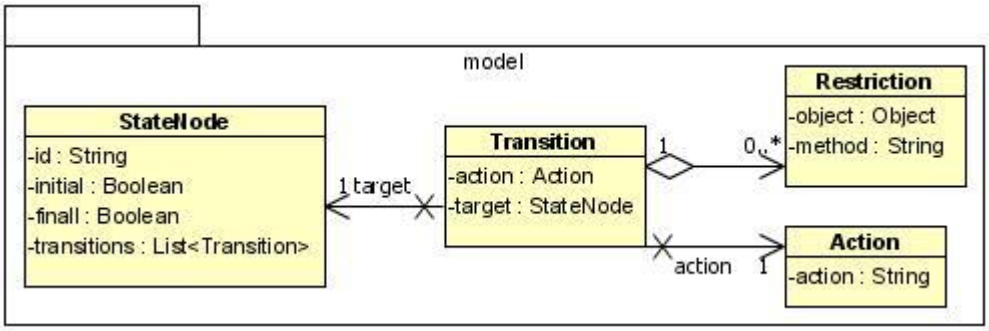


Figura18. Diagrama de Clases. Componente Gestión de Estados. Paquete model.

Agrupa las clases necesarias para representar la configuración definida en los archivos xml en el mundo objetual. Esta estructura de clases de dominio, es utilizada por el conversor, y posteriormente, por el autómata para efectuar todo el proceso que engloba una o varias transiciones de estado.

Tabla 13. Clases y Recursos del Paquete model.

Nombre	Función
StateNode	Elemento que representa en el modelo objetual un nodo del autómata de estados, elemento principal contenido en los archivos de configuración. Se corresponde con la ubicación de un estado en un momento determinado.
Transition	Es la representación objetual de una transición específica para realizar un cambio de estado.
Restriction	Posee la información necesaria para verificar una restricción determinada para efectuar la transición de estado.
Action	Constituye la representación de una acción que desencadenara un cambio de estado.

Teniendo en cuenta las tendencias de tecnologías utilizadas para el desarrollo de sistemas policiales en la UCI, este componente soporta su utilización en modo *stand-alone*[8] (sin *frameworks* y/o contenedores), y la integración con los *frameworks* de peso ligero más importantes y utilizados en la actualidad: *Spring* y *Seam*. Para lograr esto se realizaron dos distribuciones: la primera de ellas contiene la implementación estándar que brinda soporte para cargar restricciones que no necesiten pertenecer a un contexto, ya sea el contexto web de la aplicación o el perteneciente a algún *framework*. La principal diferencia entre estas dos distribuciones es la manera de realizar el proceso de conversión, y más específicamente, el proceso de inyección de dependencias.

Para el uso de este componente, de manera general para los dos modos de utilización, es necesario realizar los siguientes pasos:

1. La clase que se desee incluir en un flujo de estados que sea manejado por el autómata debe implementar la interface *AFDFlow*, la cual posee dos métodos necesarios para obtener el *StateNode* en el que se encuentra, y fijar el nuevo en caso de efectuarse la transición de estado.
2. Los métodos encargados de verificar las restricciones deben ser de tipo *Boolean*, y deben recibir un parámetro de tipo *AFDFlow*.
3. Definir las configuraciones necesarias en el archivo xml como se muestra en la *figura19*.

Para incluir la notificación a elementos interesados.

1. Los métodos que se desea sean ejecutado una vez que se produzca el cambio de estado deben recibir un parámetro de tipo *AFDFlow.*
2. Definir los *observer* en el archivo de configuración como se muestra en la *figura19*.

3.4.2.1 Modo Stand-Alone

Para utilizar este componente en este modo, sin interactúar con otros *frameworks* o componentes, es necesario realizar manualmente todo el proceso de instanciación del autómata, y ordenarle que ejecute la transición de estado, dada una acción determinada. En esta distribución, las clases encargadas de verificar el cumplimiento de las restricciones y las interesadas en ser notificadas una vez ocurra la transición de estado, se instancian la primera vez que sean procesadas, y posteriormente, en caso

[8] Modo de funcionamiento que define el comportamiento de un recurso, aplicación o componente sin la interacción o apoyo de componentes o recursos externos.

necesario, se utiliza esa misma instancia. Una cuestión muy importante a tener en cuenta es que aunque alguna de las clases interesadas y/o de las que verifican las restricciones pertenezca a un contexto, estas serán instanciadas normalmente, perdiendo todo privilegio o beneficios obtenidos por pertenecer al contexto. En la figura*19* se muestra un ejemplo del archivo xml de configuración para utilizar este componente en modo stand-alone, y se detalla cada *tag*[9], describiendo su utilización.

Un ejemplo de utilización de este componente en este modo se muestra a continuación. El proceso de instanciación del autómata se realiza a través de la *factory* y la transición de estado invocando directamente al método perteneciente al autómata devuelto por la *factory*.

```
package test;

import cmp.state.afd.AFD;

public class Test {

    public static void main(String[] args) {
        AFD afd = (AFD) AFDFactory.createNewAFDInstance("cmp/state/configuration/states.xml");
        Documento documento = new Documento();
        afd.executeTransition(documento, new Action("incluir"));
    }
}
```

Figura19. Ejemplo del uso del componente en modo Stand-Alone.

```
<?xml version="1.0" encoding="UTF-8"?>
<states>
    <automata id="estadoDocumento" clazz="test.EstadoDocumento">
        <observers>
            <observer class="test.MyObserver" method="updateByNotification" />
            <observer class="test.MyObserver1" method="updateByNotification1" />
        </observers>
        <state-node id="initialNode" initial="true">
            <transitions>
                <transition action="incluir" target="enCurso">
                    <restrictions>
                        <restriction class="test.MisRestricciones" method="checkEmitido" />
                        <restriction class="test.MisRestricciones" method="check" />
                    </restrictions>
                </transition>
                <transition action="archivar" target="archivado" />
            </transitions>
        </state-node>
        <state-node id="enCurso" />
        <state-node id="archivado" finall="true" />
    </automata>
</states>
```

Figur20. Ejemplo de xml de configuración para el uso Stand-Alone.

[9] En sentido informático un *tag* es un conjunto de caracteres que se añade a un elemento de los datos para identificarlo.

Tabla 14. Tags del archivo de configuración.

Nombre	Función
states	Etiqueta global de todo el archivo de configuración. Indica que el archivo en cuestión contiene las definiciones correspondientes a la máquina de estados definida en este componente.
automata	Indica la definición de un autómata sobre una clase determinada.
observers	Contiene la lista de elementos interesados en ser notificados cuando ocurra una transición de estado cualquiera contenida en el autómata donde está definido.
observer	Constituye un observador concreto, es decir, una clase concreta a la que se debe notificar. Más específicamente, se refiere a un método contenido en una clase determinada, permitiendo de esta manera que una misma clase pueda recibir múltiples notificaciones de cambios de estado provenientes de diferentes autómatas. El método y la clase son especificados en los atributos *method* y *class* respectivamente.
state-node	Se corresponde técnicamente con un nodo del autómata, y según la lógica del negocio a una posición determinada de un elemento en un momento específico. Su procesamiento devendrá en la clase *StateNode* vista anteriormente. El atributo *id*, identifica un nodo de estado dentro del autómata; las propiedades *initial* y *finall*, indican si el nodo es el inicial o el final de toda la configuración, y ambas son mutuamente excluyentes.
transitions	Contiene la lista de transiciones que serán ejecutadas a partir de un nodo determinado.
transition	Constituye una transición concreta, es decir, un cambio de estado a efectuar al ocurrir una acción determinada. Las propiedades definidas en este tag: *action*, se refiere a la acción que desencadenará la transición de estado (en el modelo creado para representar la configuración en el mundo objetual, se corresponden con la clase *Action)*, *target*, el nodo destino que tomará el elemento una vez ejecutada, se refiere a un *state-node*.
restrictions	Contiene la lista de restricciones que deben cumplirse para efectuar una transición de estado determinada.
restriction	Constituye una condición del negocio impuesta para realizar un cambio de estado, se corresponde con la clase *Restriction*. Las propiedades definidas *class* y *method*, son exactamente el método que se encargará de realizar la verificación y la clase a la que pertenece.

3.4.2.2 Modo de Integración.

Esta forma de utilización constituye la más importante y debe constituir la más utilizada por los desarrolladores en sentido general, ya que en la mayoría de las aplicaciones empresariales desarrolladas actualmente con J2EE, son utilizados *frameworks* de peso ligero que permiten un desarrollo mas rápido y eficiente.

La arquitectura del componente permite ser extendido para integrarse con estos *frameworks*. En esta distribución se brinda el soporte necesario para la integración con *Spring.* El mismo mecanismo utilizado para realizar esta integración puede ser extendido para integrarse con cualquier contenedor de POJOS.

Para lograr esto se hizo necesario redefinir algunas funcionalidades del conversor de estados, a fin de obtener los recursos encargados de verificar las restricciones y los observadores del contexto de la aplicación o *framework*, en lugar de crear nuevas instancias de estos. Adicionalmente y como una solución más completa, esta distribución incluirá lo necesario para que funcione perfectamente en modo Stand-Alone, e incluyendo combinaciones de ambos. Básicamente, los pasos para realizar la integración con cualquier *framework*, son, además de los generales vistos anteriormente, los que siguen:

1. Crear una clase conversor, que *extienda* [10]del *StateConversor*, e implemente la interfaz proporcionada por el *framework* para obtener su contexto. El objetivo de la extensión es obtener todos los métodos invariables, teniendo que redefinir solamente aquellos que están directamente vinculados a la inyección de dependencias de las restricciones y los observadores, para resolverlas en el contexto del contenedor de POJOS y registrarla en el archivo *DefaultStrategies.properties,* para indicar al componente que la clase que utilizará para realizar la conversión será la nueva que ha sido creada.
2. Definir esta clase en un archivo de configuración que será procesado por el *framework*, para tener acceso al contexto desde ella.
3. Declarar los objetos del autómata que sean necesarios en un archivo de configuración utilizando la *fábrica abstracta*, desarrollada para crear instancias del autómata proporcionando los parámetros deseados. Una vez hecho esto, los objetos del autómata declarados pueden ser utilizados obteniéndolos directamente del contexto o mediante inyección de dependencias, lo cual depende completamente del *framework* escogido y de la intención con que sea utilizado.

[10] Término utilizado en Java para referirse a la herencia de clases.

A continuación se muestra como sería el archivo xml de configuración para utilizar este componente en modo de integración.

```
<?xml version="1.0" encoding="UTF-8"?>
<states>
    <automata id="estadoDocumento" clazz="test.EstadoDocumento">
        <observers>
            <observer ref="agendaService2" method="updateByNotification" />
            <observer class="vnz.cicpc.test.MyObserver" method="updateByNotification" />
        </observers>
        <state-node id="initialNode" initial="true">
            <transitions>
                <transition action="incluir" target="enCurso">
                    <restrictions>
                        <restriction ref="agendaService1" method="checkEmitido" />
                        <restriction class="vnz.cicpc.test.RestrictionCheck" method="check" />
                    </restrictions>
                </transition>
                <transition action="archivar" target="archivado" />
            </transitions>
        </state-node>
        <state-node id="enCurso" />
        <state-node id="archivado" finall="true" />
    </automata>
</states>
```

Figura21. Xml de configuración para el uso integrado.

Los tags en este modo son exactamente equivalentes a los utilizados en el modo Stand-Alone. La diferencia que existe entre ambos está dada por la propiedad *ref*, que indica que el objeto referenciado se encuentra en otro contexto con el identificador indicado, por lo que se debe obtener a través de él, en lugar de instanciarlo directamente, para mantener los beneficios proporcionados por el *framework*.

A continuación se muestra cómo fue realizada la integración con Spring, uno de los *frameworks* de peso ligero más importantes y de uso más generalizado.

1. Crear la clase indicada en el primer paso para obtener los objetos del contexto, como se muestra en la siguiente figura. Para la integración con Spring la clase creada debe implementar la interfaz ApplicationContextAware.

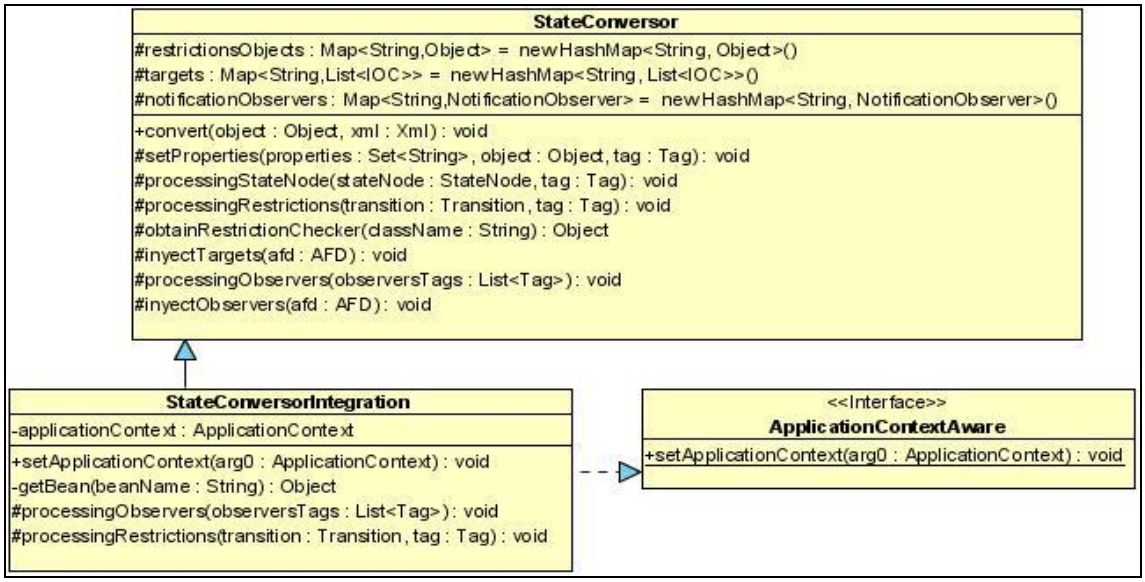


Figura22. Implementación del conversor para la integración con Spring.

2. Registrar la clase creada en el archivo *properties* indicado.

```
# Implementacion por defecto de todas las interfaces
cmp.state.afd.AFD =cmp.state.afd.support.AFDImpl
cmp.core.parser.xmlparser.XmlParser =cmp.core.parser.xmlparser.support.XmlParserImpl
cmp.core.convert.AbstractConversor =cmp.state.convert.StateConversorIntegration
```

Figura23. Archivo properties para definir las implementaciones de las interfaces.

3. Los pasos 2 y 3 establecidos anteriormente se refieren a la definición de objetos, en este caso *beans,* en archivos de configuración. Por tanto se muestra en la siguiente figura la configuración de un archivo xml que contiene todos los elementos necesarios para el correcto funcionamiento de la integración. En la figura, el bean *agendaService* contiene una propiedad que se refiere a un autómata con identificador *afdState*, el cual es creado a través de alguno de los métodos de la fábrica abstracta, mediante la propiedad *factory-method*, pasando los parámetros como argumentos del constructor, ya que *Spring* asume este método como si fuera el constructor de la clase. La clase creada para realizar la integración(ver *figura22*) *StateConversorIntegration* es utilizada para crear un *bean* nombrado *stateConvertIntegration* el cual es pasado como parámetro a la factoría. Los demás *beans* registrados en este archivo son utilizados para configurar el autómata como se muestra en la *figura21*.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:
          http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
          http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.0.xsd
          http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.0.xsd">

    <bean id="agendaService" class="vnz.cicpc.bussineslogic.agendatrabajo.impl.AgendaServiceImpl">
        <property name="afd" ref="afdState" />
    </bean>

    <bean id="afdState" factory-method="createNewAFDInstance" class="cmp.state.factory.AFDFactory">
        <constructor-arg value="vnz/cicpc/bussineslogic/agendatrabajo/config/spring-states.xml" />
        <constructor-arg ref="stateConvertIntegration" />
    </bean>

    <bean id="stateConvertIntegration" class="cmp.state.convert.StateConversorIntegration" />

    <bean id="agendaService1" class="vnz.cicpc.bussineslogic.agendatrabajo.impl.AgendaServiceImpl1" />

    <bean id="agendaService2" class="vnz.cicpc.bussineslogic.agendatrabajo.impl.AgendaServiceImpl2" />

</beans>
```

Figura24. Configuraciones necesarias para realizar la integración.

A continuación se muestra un ejemplo sencillo de utilización de esta integración.

```
public class Principal extends AbstractSingleSpringContextTests {

    private AgendaService agendaService;

    @Override
    protected String[] getConfigLocations() {
        return CargadorContextos.cargarContextosDeWebXml();
    }

    @Override
    protected void onSetUp() throws Exception {
        agendaService = (AgendaService) this.applicationContext.getBean("agendaService");
        super.onSetUp();
    }

    public void testIntegration() {
        agendaService.probar();
    }
}
```

Figura25. Ejemplo de utilización del componente integrado.

3.4.2.3 Realización de los Casos de Uso.

En el presente epígrafe se muestra cómo con el diseño propuesto son realizados un conjunto de casos de uso que están soportados por este componente.

- Registrar Configuración de Estado, Registrar Elementos Interesados.

  Estos casos de uso se refieren a la definición en archivos xml de toda la configuración de uno o varios autómatas. Para garantizar el correcto funcionamiento del componente se definirá un archivo xsd que establecerá una forma correcta y única para la confección de los archivos xml. Incluyendo este xsd en cada xml que se cree, se validará la estructura definida por el desarrollador, proporcionando de esta manera un alto grado de confiabilidad y seguridad en el trabajo que está siendo realizado. Es válido indicar además que el proceso de validación no termina aquí, sino que en el momento de procesar el archivo, este será validado nuevamente en concordancia con las definiciones establecidas. Como restricción de uso, con el objetivo de lograr una mayor limpieza y organización en todo el código y la configuración referente y usada por este componente, se creará un archivo de configuración por cada automáta. Este archivo de configuración se muestra en el *Anexo2*. Dado que estos casos de uso consisten meramente en el registro de configuraciones en un archivo de datos y su validación para su posterior utilización, se estima que no es necesario realizar un diagrama de interacción para mostrar como interactúan los objetos presentes.

- Procesar Configuración de Estado.

  Se encarga de obtener la información contenida en un archivo xml, convertirla a código intermedio y posteriormente representar esta información mediante varias clases diseñadas especialmente para este fin y que serán contenidas por un autómata. A continuación se muestra el diagrama de secuencia correspondiente a este flujo de eventos.

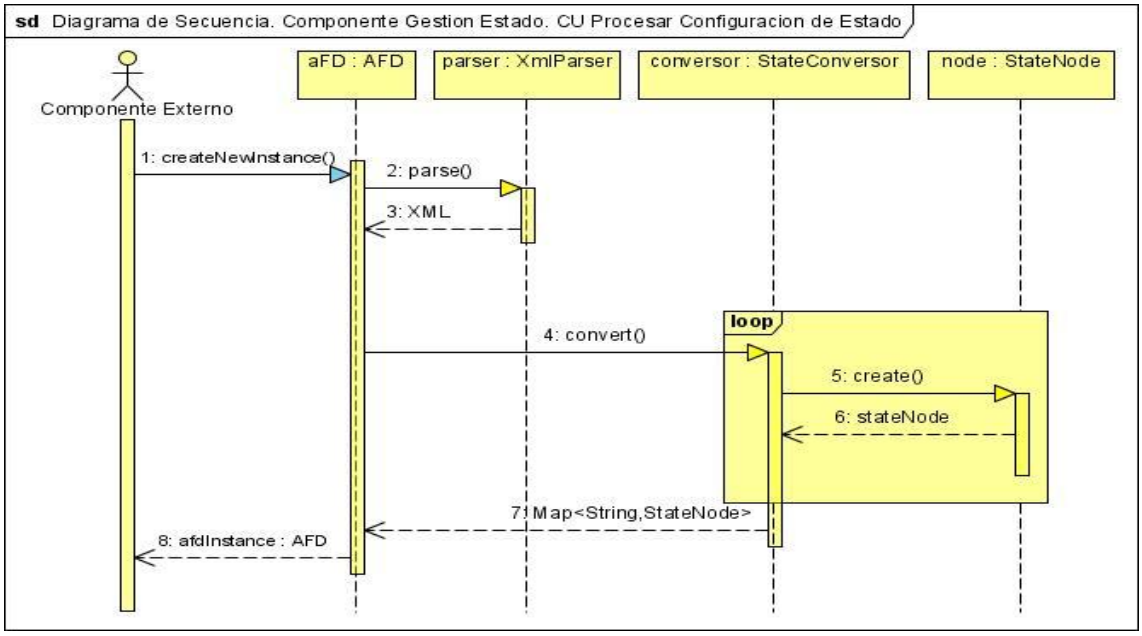


Figura26. Diagrama de Secuencia. CU Procesar Configuración.

- Cambiar Estado de Elemento.

  Engloba los procesos necesarios para realizar una transición de estado, habiendo previamente procesado los datos contenidos en el archivo xml, y teniendo toda la configuración en el autómata correspondiente. A continuación se muestra el diagrama de secuencia correspondiente a este flujo de eventos.

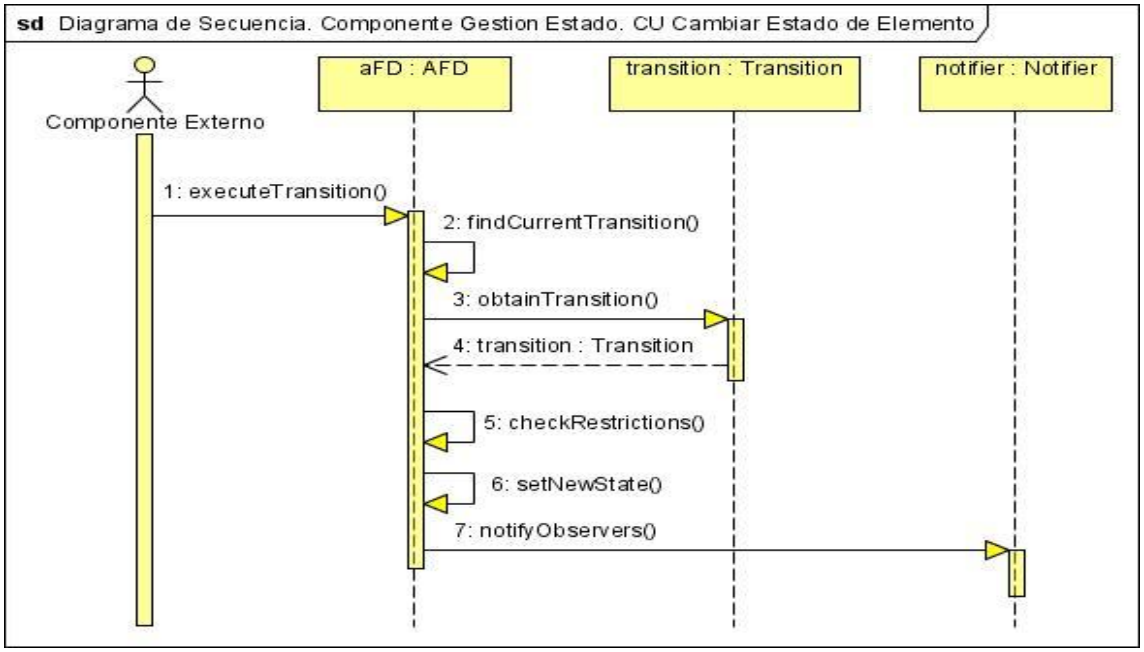


Figura27. Diagrama de Secuencia CU Cambiar Estado de Elemento.

- Verificar Restricciones.

  Este caso de uso es nivel subfunción, es decir, nunca será instanciado directamente por un actor, sino que dará soporte a determinados casos de uso, que serían los casos de uso bases.

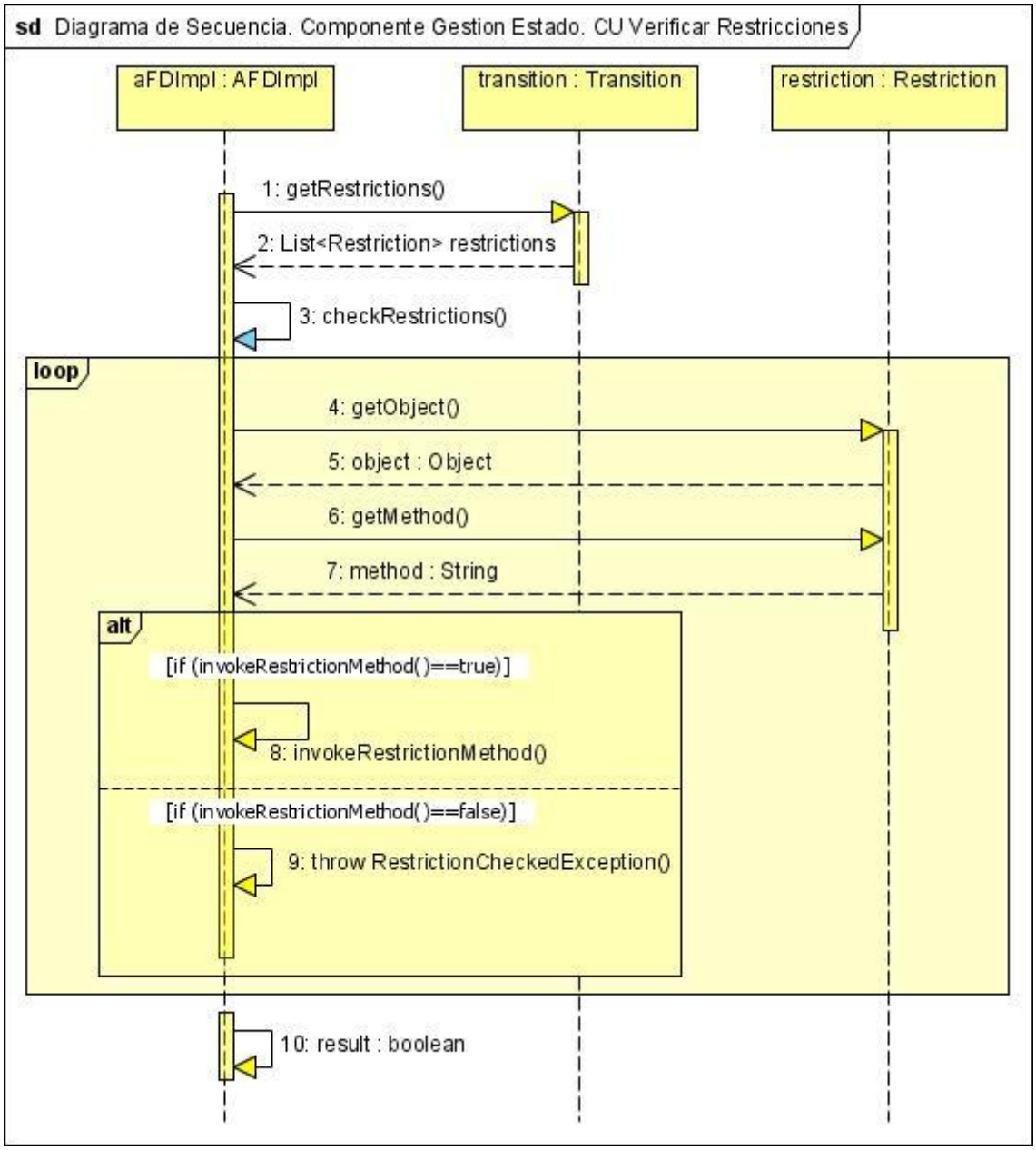


Figura28. Diagrama de Secuencia. CU Verificar Restricciones.

- Notificar Ocurrencia de Eventos a Elementos Interesados.

  Se encarga de notificar la ocurrencia de un cambio de estado a todos aquellos interesados en conocerlo. Es de nivel subfunción, es decir, no interactúa directamente con un actor, sino que es llamado desde un caso de uso base.

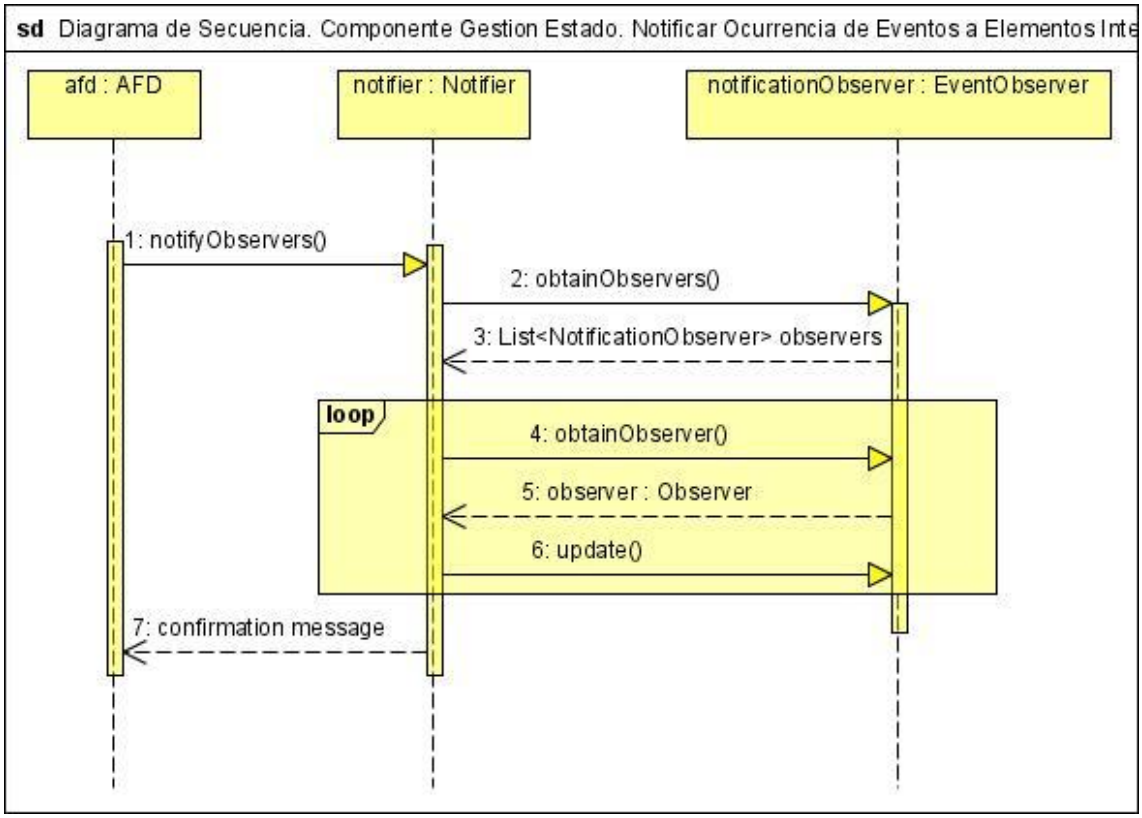


Figura29. Diagrama de Secuencia. CU Notificar Ocurrencia de Eventos a Elementos Interesados.

3.4.3 Componente Manejo de Conceptos de Investigación.

El desarrollo de aplicaciones web en Java se caracteriza por el uso de *frameworks* que facilitan y agilizan la construcción del software, especialmente si son de mediana o gran envergadura. Estos *frameworks* se caracterizan por proporcionar formas genéricas de realizar actividades comunes, lo que supone un aumento de la productividad.

Un proyecto de mediana o gran envergadura posee generalmente un equipo de desarrollo de considerable tamaño, lo que supone un aumento de la velocidad de desarrollo, pero trae consigo falta de comunicación lo que supone problemas de diseño que a la larga conducen a una reducida mantenibilidad de los sistemas.

Este componente propone un modelo general para representar conceptos investigativos. Brindando diferentes opciones para manejar las actividades comunes y generales en los que están implicados, permitiendo su extensibilidad y reutilización para adaptar el modelo brindado a las necesidades de cada negocio específico.

Engloba funcionalidades referentes a expedientes, documentos y elementos investigativos, proporcionando soporte para la gestión de estados de cada uno. A continuación se muestran diferentes vistas de los paquetes más importantes, con el fin

de exponer en detalle las funcionalidades contenidas, para lograr una mayor comprensión de su principio de funcionamiento y el objetivo perseguido.

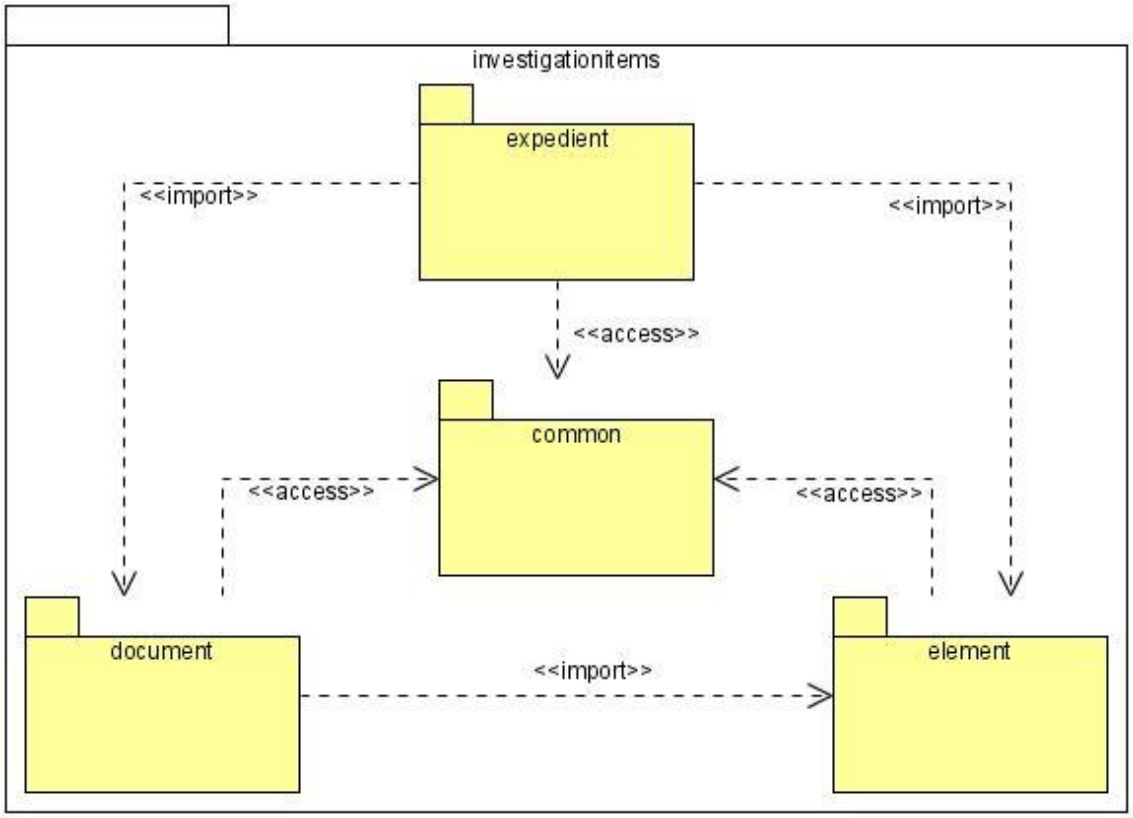


Figura30. Vista de Paquetes Componente Manejo y Estandarización de Conceptos de Investigación.

Tabla 15. Paquetes del Componente Manejo y Estandarización de Conceptos de Investigación.

Nombre	Función
element	Encapsula la lógica necesaria para soportar las funcionalidades referentes a los elementos investigativos.
document	Agrupa las clases relativas al tratamiento y distribución básica de los documentos. Brindando posibilidades de extenderlos y personalizarlos.
expedient	Contiene los componentes necesarios para proporcionar una distribución básica de expedientes para sistemas policiales. Incluye la composición de expedientes a partir de otros expedientes.
common	Brinda funcionalidades comunes a todos los conceptos de investigación.
relations	Agrupa toda la lógica requerida para la integración de elementos investigativos, documentos y expedientes.

A continuación se muestran diferentes vistas de los paquetes más importantes, con el fin de exponer en detalle las funcionalidades contenidas, para lograr una mayor comprensión de su principio de funcionamiento, así como de la distribución de los componentes y clases según sus funcionalidades, teniendo siempre como premisas mantener un acoplamiento bajo y una cohesión alta a fin de lograr un modelo que permita ampliamente su reutilización y extensibilidad.

❖ El paquete **common**.

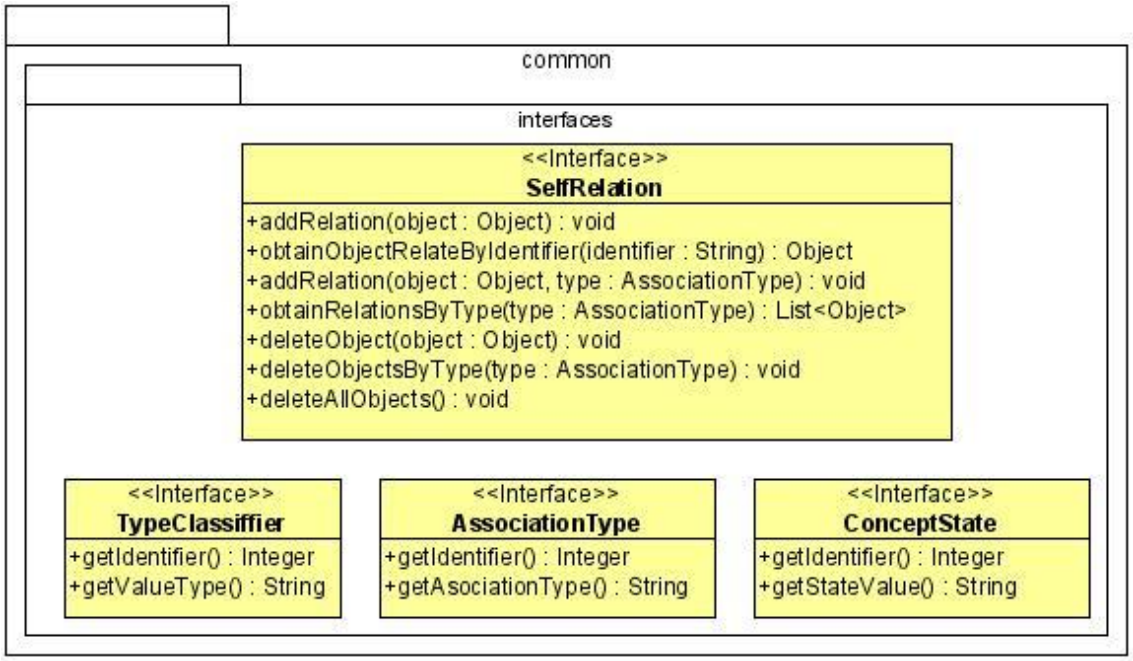


Figura31. Diagrama de Clases. Componente Conceptos de Investigación. Paquete common.

Este paquete contiene las interfaces requeridas para estandarizar las relaciones de composición entre clases, la tipificación de elementos, y la clasificación de las asociaciones. Es un paquete de uso común por lo que estará incluido en todos los diagramas donde sea requerido para proporcionar un mayor entendimiento del diseño propuesto.

Tabla 16. Recursos del paquete common.

Nombre	Función
SelftRelation	Interface creada con el objetivo de establecer un contrato de implementación y uso para los elementos, documentos, expedientes y todas aquellas extensiones que sean requeridas del modelo que contengan relaciones con ellos mismos.

TypeClassifier	Interface utilizada como wrapper para los tipificadores de elementos. Su objetivo principal es aumentar la reusabilidad manteniendo un acoplamiento bajo.
AssociationType	Su objetivo es actuar como wrapper para los clasificadores de relaciones, desacoplando el modelo.
ConceptState	Su objetivo es actuar como wrapper para los diferentes estados de los conceptos investigativos, desacoplando el modelo.

❖ El paquete **element**.

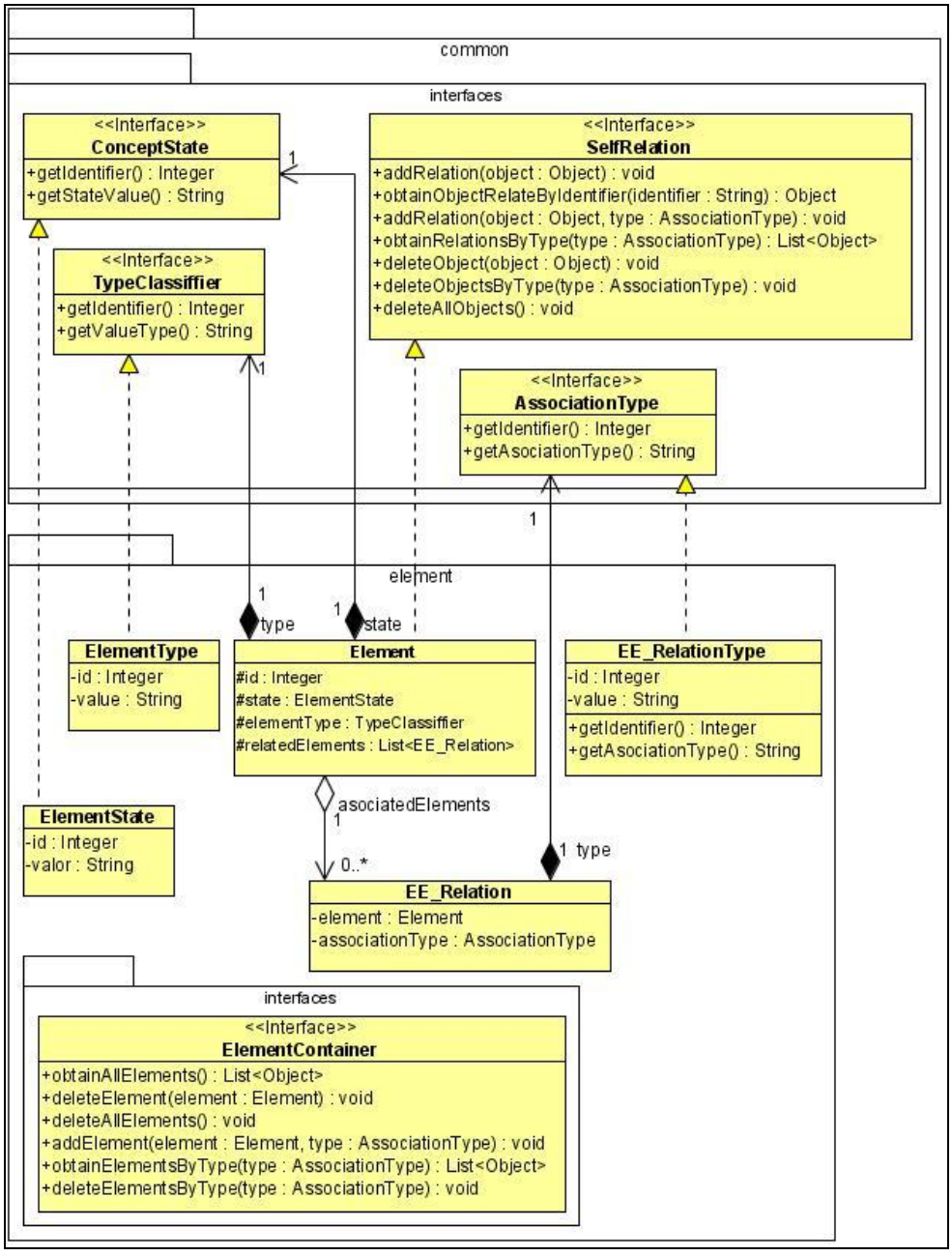
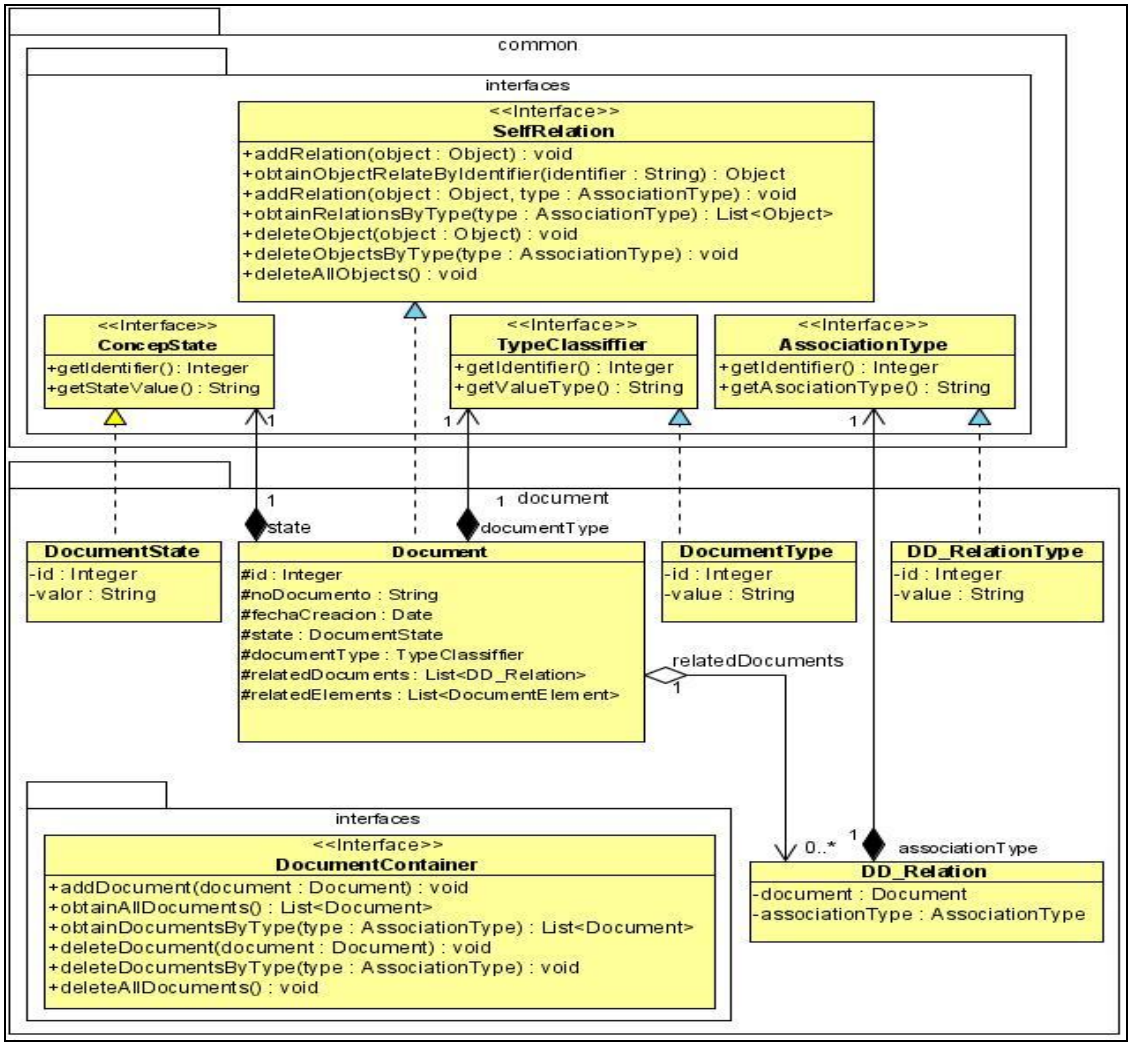


Figura32. Diagrama de Clases. Componente Conceptos de Investigación. Paquete element.

Tabla 17. Recursos del paquete element.

Nombre	Función
ElementType	Implementación de la interface *TypeClassifier* que tipifica los posibles tipos de elementos que puedan existir en el modelo.
ElementState	Representa el estado de un elemento.
EE_RelationType	Implementación de la interface AssociationType que clasifica las posibles relaciones entre elementos.
EE_Relation	Da soporte a las relaciones entre elementos a cualquier nivel de anidación.
Element	Constituye la base de todos los elementos investigativos. Posee atributos básicos, y funciones generales que pueden ser personalizadas. Adicionalmente incorpora el modelo de composición entre elementos investigativos.
ElementContainer	Constituye un contrato de implementación y uso para todas las entidades que deseen contener elementos investigativos.

❖ El paquete **document**.

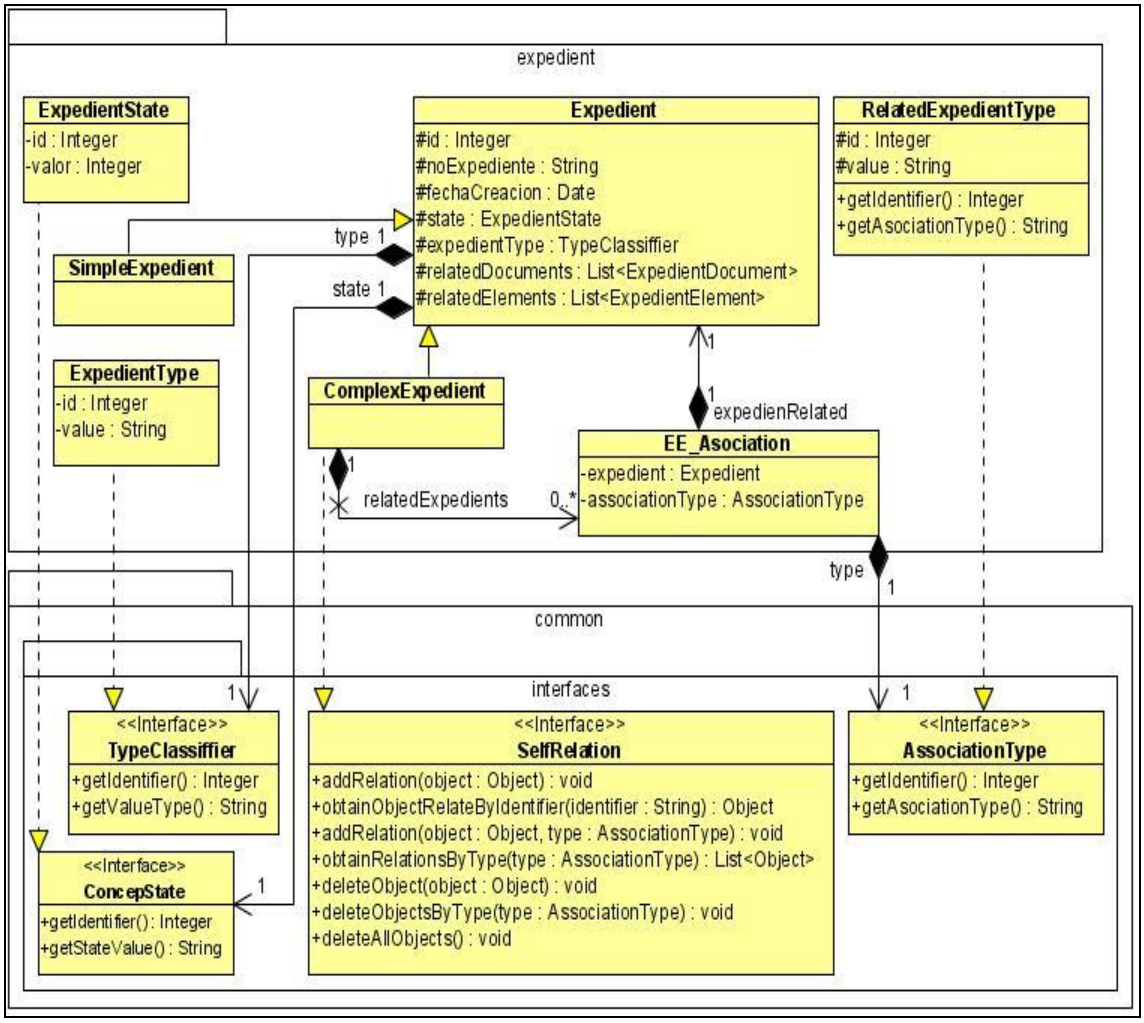


Figur33. Diagrama de Clases. Componente Conceptos de Investigación. Paquete document.

Tabla 18. Recursos del paquete document.

Nombre	Función
DocumentType	Implementación de la interface *TypeClassifier* que tipifica los posibles tipos de documentos que puedan existir en el modelo.
DocumentState	Representa el estado de un documento.
DD_RelationType	Implementación de la interface *AssociationType* que clasifica las posibles relaciones existentes documento-documento.
DD_Relation	Da soporte a las relaciones entre documentos a cualquier nivel de anidación.
Document	Constituye la base de todos los documentos. Posee atributos básicos, y funciones generales que pueden ser personalizadas. Adicionalmente incorpora el modelo de composición entre documentos.
DocumentContainer	Constituye un contrato de implementación y uso para todas las entidades que deseen contener documentos.

❖ El paquete **expedient**.



Figur34. Diagrama de Clases. Componente Conceptos de Investigación. Paquete expedient.

Tabla 19. Recursos del paquete expedient.

Nombre	Función
ExpedientType	Implementación de la interface *TypeClassifier* que tipifica los posibles tipos de expedientes que puedan existir en el modelo.
ExpedientState	Representa el estado de un expediente.
RelatedExpedientType	Implementación de la interface *AssociationType* que clasifica las posibles relaciones existentes expediente-expediente.
EE_Asociation	Da soporte a las relaciones entre expedientes a cualquier nivel de anidación. Registrando para cada una el expediente asociado y el tipo de asociación dado por la clase *RelatedExpedientType.*
Expedient	Constituye la base de todos los expedientes. Posee atributos básicos, y funciones generales que pueden ser personalizadas. Adicionalmente incorpora el modelo de relación con elementos investigativos y documentos.
SimpleExpedient	Constituye la personalización más simple de los expedientes conocidos. No contiene el modelo que soporta las relaciones de composición entre expedientes.
ComplexExpedient	Representa a una familia de expedientes compuestos por los atributos que hereda de *Expedient*, e incorpora el modelo de composición entre expedientes, permitiendo tipificar la relación y proporcionando funcionalidades básicas necesarias para el trabajo.

❖ El paquete **relations**.

Contiene las clases e interfaces necesarias para relacionar todo el modelo, así como las implementaciones de las interfaces establecidas por cada concepto para ser contenido. Se proporciona soporte para las relaciones expediente-documento, expediente-elemento y documento-elemento, proporcionando de esta manera una solución para tratar los contenedores de información. En la vista proporcionada se muestran varios elementos correspondientes a otros paquetes con el objetivo de clarificar las funcionalidades brindadas y mostrar la interacción entre los principales conceptos del componente.

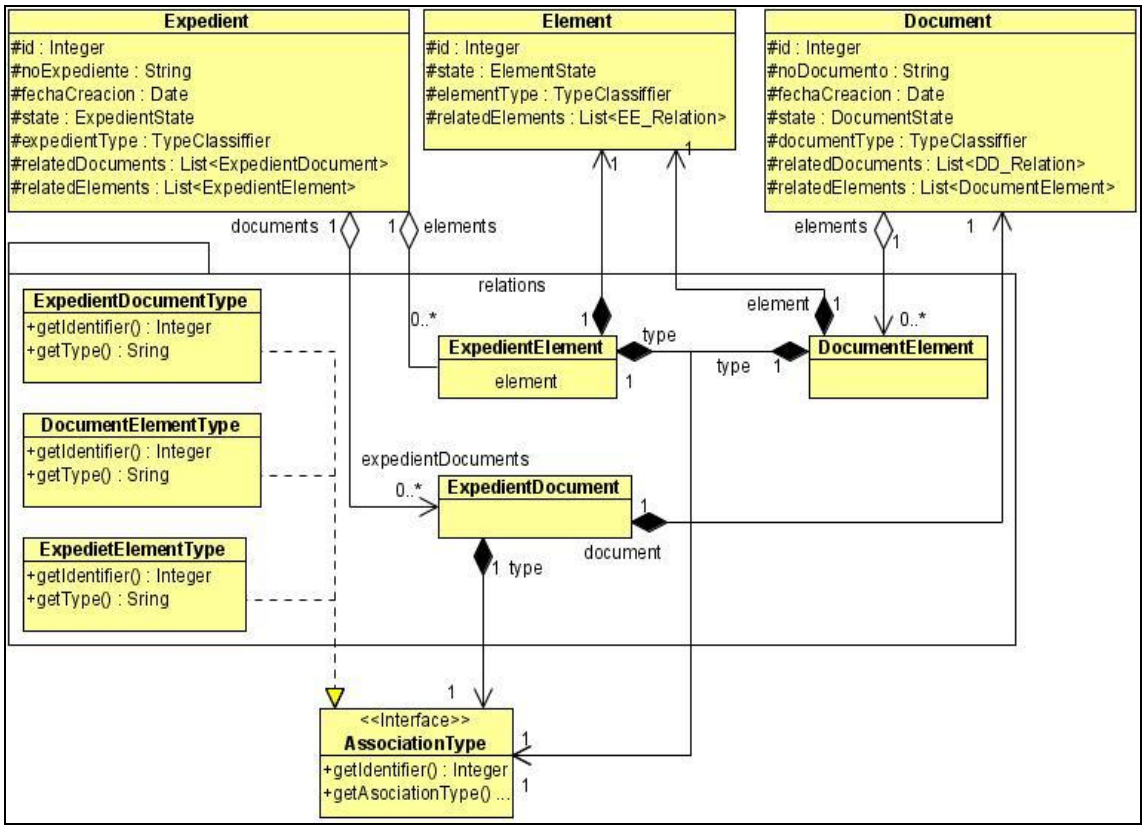


Figura35. Diagrama de Clases. Componente Conceptos de Investigación. Paquete relations.

Tabla 20. Recursos del paquete relations.

Nombre	Función
ExpedientDocumentType	Implementación de la interface *AssociationType* que tipifica las posibles relaciones existentes entre expedientes y documentos.
DocumentElementType	Implementación de la interface *AssociationType* que tipifica las posibles relaciones existentes entre documentos y elementos.
ExpedientElementType	Implementación de la interface *AssociationType* que tipifica las posibles relaciones existentes entre expedientes y elementos.
ExpedientElement	Da soporte a las relaciones entre expedientes y elementos a cualquier nivel de anidación. Registrando para cada una el expediente y el elemento asociado además del tipo de relación.
ExpedientDocument	Da soporte a las relaciones entre expedientes y documentos a cualquier nivel de anidación. Registrando para cada una el elemento y el documento asociado además del tipo de relación.
DocumentElement	Da soporte a las relaciones entre elementos y documentos a cualquier nivel de anidación. Registrando para cada una el elemento y el documento asociado además del tipo de relación.

3.4.4 Componente de Elementos Investigativos Básicos.

Todo sistema policial maneja diferentes tipos de elementos investigativos. En este componente son agrupados los elementos comunes de los sistemas policiales estudiados, con los atributos generales y comunes de cada uno de ellos, manteniendo siempre como principio base la necesidad de permitir su futura extensión para lograr una adaptacion sencilla y rápida a las especifidades del negocio en cuestión. Las extensiones comunes proporcionadas se nutren de las funcionalidades soportadas en los componentes anteriores, por lo que soportan el manejo del ciclo de vida mediante el componente de gestión de estados, y los modelos de relaciones a través del componente para el manejo de conceptos de investigación, con el cual está estrechamente relacionado.

A continuación se muestra la vista de paquetes para mostrar la estructura y organización del presente componente.

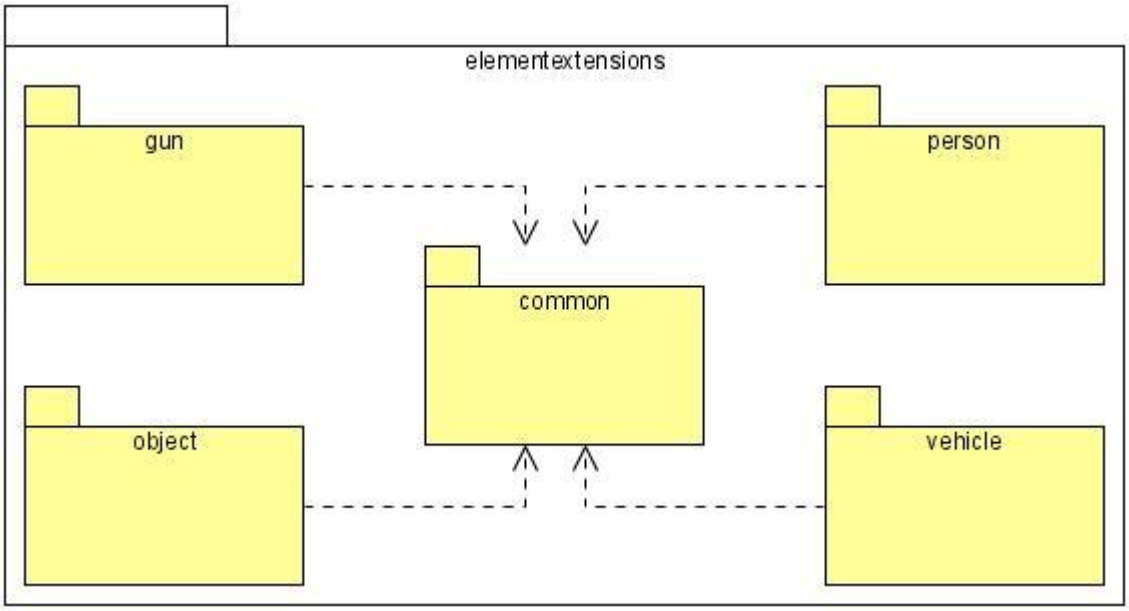


Figura36. Vista de Paquetes Componente de Elementos Investigativos Básicos.

Tabla 21. Paquetes del Componente *de Elementos Investigativos Básicos*.

Nombre	Función
common	Brinda funcionalidades comunes a todos los complementos de investigación.
gun	Agrupa las clases requeridas para dar soporte a las armas.
object	Agrupa las clases requeridas para dar soporte a los objetos.
person	Agrupa las clases requeridas para dar soporte a las personas.
vehicle	Agrupa las clases requeridas para dar soporte a los vehículos.

A continuación se muestran los diagramas de clases correspondientes a cada uno de los paquetes del diagrama anterior, como la vista estática desde el punto de vista de diseño. En estos diagramas son representados elementos del componente para el manejo de conceptos de investigación, ya que constituyen conceptos importantes a tener en cuenta para la comprensión general del modelo.

❖ El paquete **common**.

Este paquete constituye un paquete de propósito general que agrupa funcionalidades y clases comunes a todas las extensiones. Su objetivo es contribuir a desacoplar el modelo.

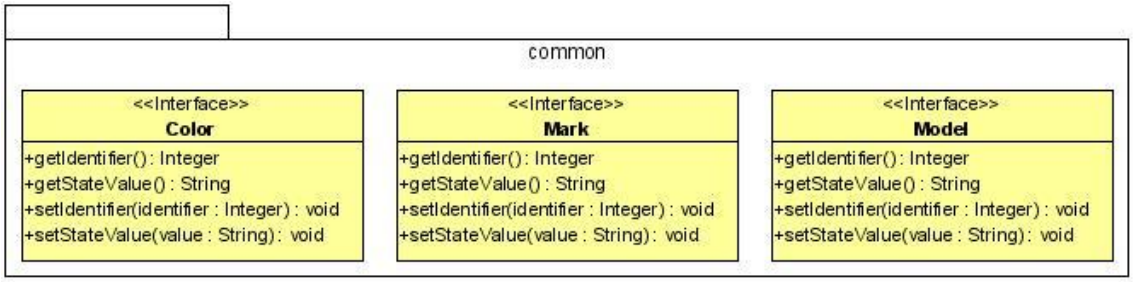


Figura37. Diagrama de Clases. Componente de Elementos Investigativos Básicos. Paquete common.

❖ El paquete **gun**.

Contiene las clases necesarias para proporcionar una distribución predeterminada para el elemento Arma. Permite ampliamente la extensibilidad e inclusión en el flujo de control de estados, propiedad que hereda de la clase *Element*.

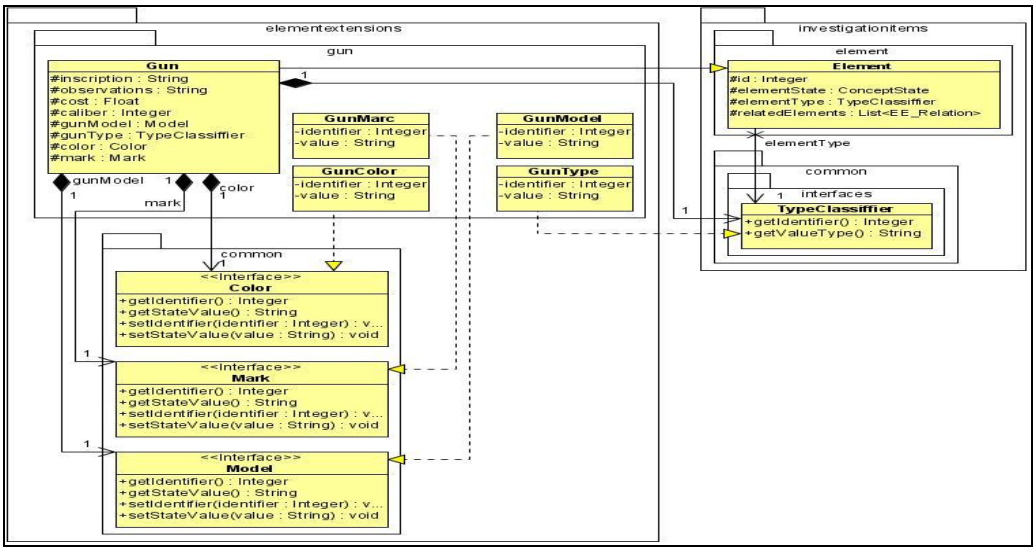


Figura38. Diagrama de Clases. Componente de Elementos Investigativos Básicos. Paquete gun.

❖ El paquete **object**.

Agrupa los elementos necesarios para soportar el elemento Objeto, portando una serie de atributos obtenidos como resultado del proceso de generalización de los sistemas estudiados con anterioridad.

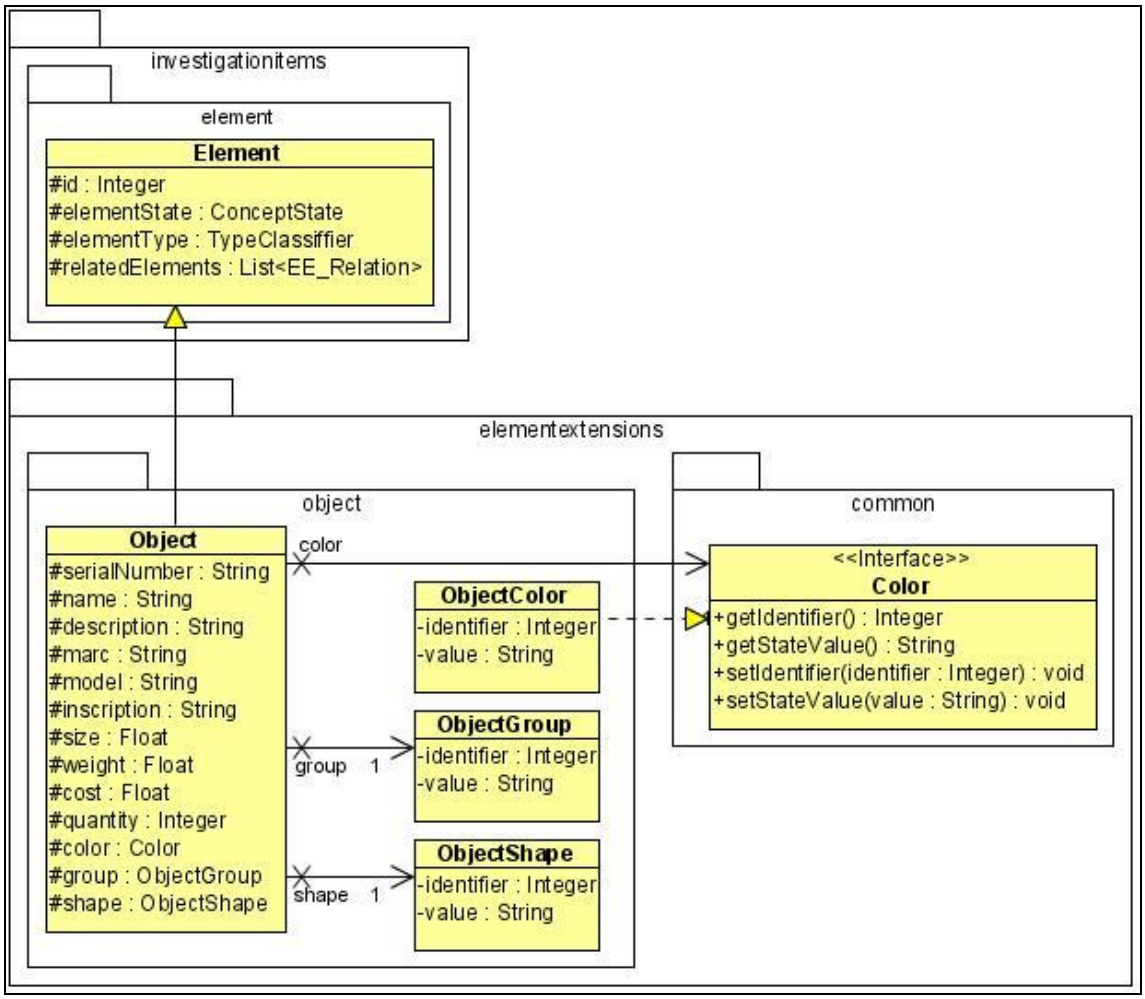


Figura39. Diagrama de Clases. Componente de Elementos Investigativos Básicos. Paquete object.

❖ El paquete **person**.

Constituye el elemento investigativo más importante y utilizado en los sistemas de este tipo.

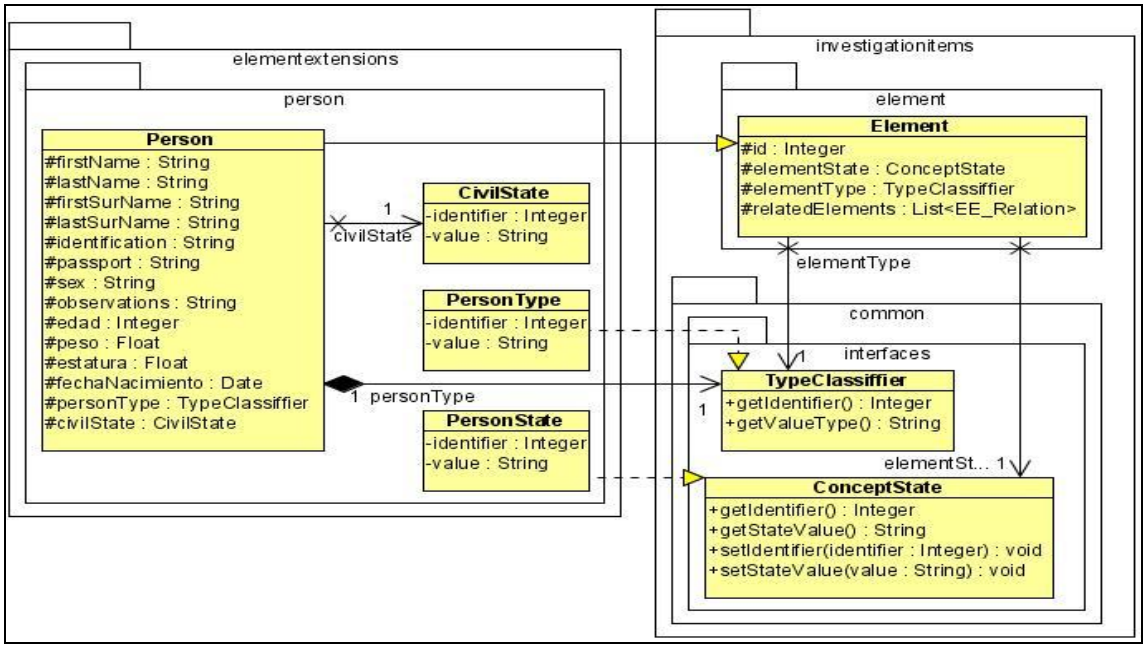


Figura40. Diagrama de Clases. Componente de Elementos Investigativos Básicos. Paquete person.

❖ El paquete **vehicle**.

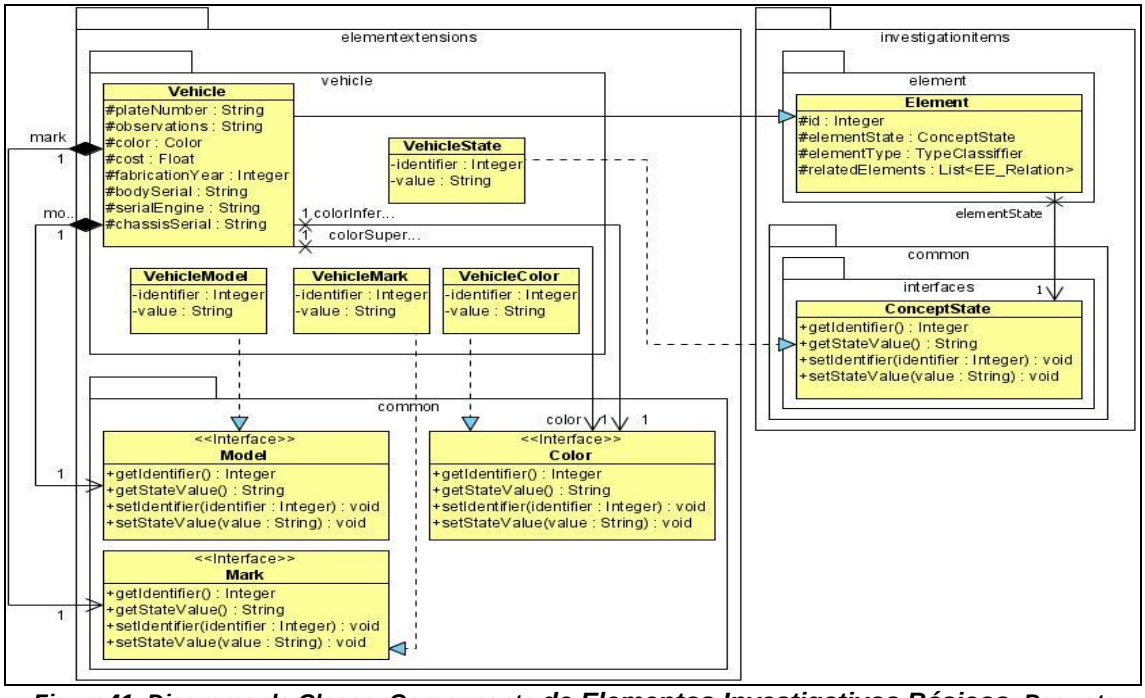


Figura41. Diagrama de Clases. Componente de Elementos Investigativos Básicos. Paquete vehicle.

3.5 Patrones de Diseño Utilizados.

En la tecnología de objetos, un patrón es una descripción de un problema y su solución, a la que se da un nombre y que se puede aplicar a nuevos contextos. Lo importante de los patrones no es expresar nuevas ideas de diseño. Es justamente lo contrario: los patrones pretenden codificar conocimiento, estilos y principios existentes y que se han probado que son válidos; cuanto más trillados y extendidos, mejor. De ahí que una definición de patrón en términos coloquiales podría ser: “Un patrón es una solución común a un problema recurrente”.

A continuación se muestran algunos de los patrones más relevantes utilizados en el diseño de la propuesta de solución.

Bajo Acoplamiento: Se refiere a cómo soportar bajas dependencias, bajo impacto del cambio e incremento de la reutilización. El acoplamiento es una medida de la fuerza con que un elemento está conectado a, tiene conocimiento de, o confía en otros elementos. Un elemento con bajo (o débil) acoplamiento no depende demasiado de otros elementos. Para garantizar un bajo acoplamiento, se utilizan interfaces, que permiten realizar todas las operaciones necesarias sin estar vinculado a una implementación concreta, lo que permitiría cambiar la implementación de dicha interfaz y todo el código que utilice, o dependa de ella, no sufra ningún cambio. Adicionalmente, y en aras de lograr una mayor flexibilidad, haciendo que el impacto sufrido al realizar un cambio sea mínimo, se registra en ficheros de configuración las interfaces con la implementación que está usando en ese momento, bastaría con cambiar la dirección donde se encuentra la implementación de una interface, sin riesgos adicionales.

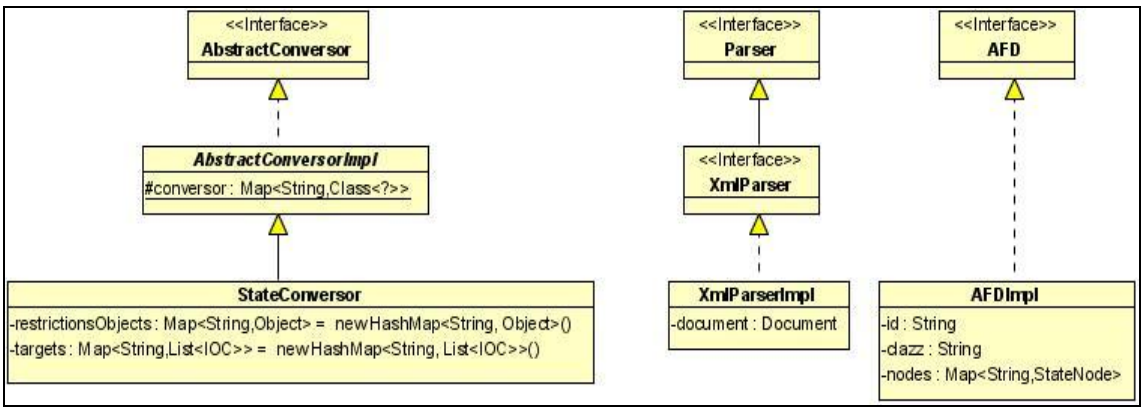


Figura42. Ejemplo del uso de interfaces para lograr un Bajo Acoplamiento.

Alta Cohesión: Consiste en la correcta asignación de responsabilidades, de forma tal que la complejidad permanezca manejable. Es un principio a tener presente durante todas las decisiones de diseño.

Abstract Factory: La fábrica abstracta se utiliza cuando el sistema debe ser independiente de cómo sus productos se crean, componen y representan; cuando debe configurarse con una familia de productos de entre múltiples posibles; cuando se quiere dar énfasis a la restricción de que una familia de productos ha sido diseñada para que actúen todos juntos y cuando se quiere proporcionar una librería de clases de productos, de los que sólo se desea revelar su interfaz, no su implementación. Para independizar la lógica de creación del AFD, se encapsuló en una factory denominada *AFDFactory*, la cual provee determinados métodos sobrecargados para brindar variedad en el momento de instanciar una clase de este tipo.

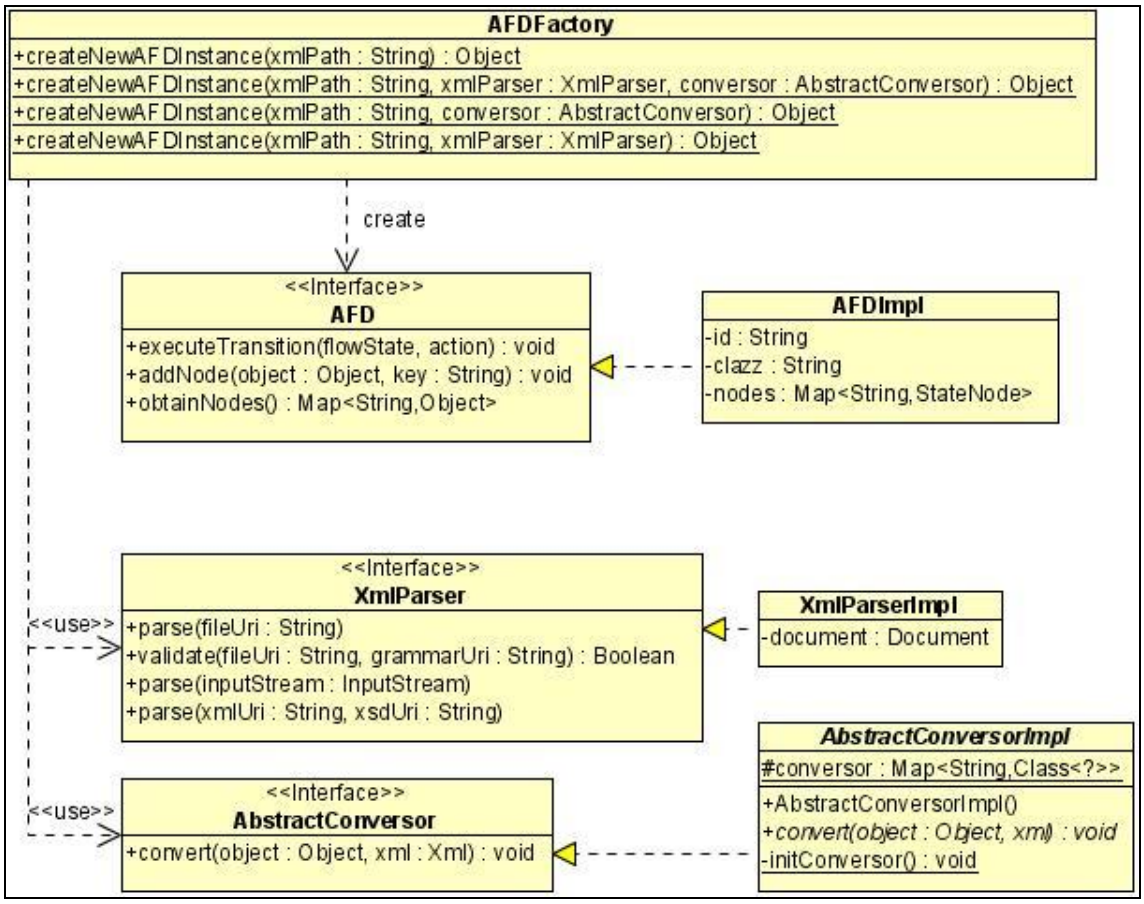


Figura43. Implementación del patrón Abstract Factory.

State: Este patrón permite a un objeto modificar su comportamiento a medida que su estado interno va cambiando, dando así la impresión de que el objeto "cambia de clase". En el desarrollo del componente para la gestión de estado, no se utilizó

fielmente el diseño establecido para este patrón, pero sí se tuvo en cuenta su principio de funcionamiento y la intención que persigue.

Creador: Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos, una tarea muy común. Su intención es encontrar un creador que necesite conectarse al objeto creado en alguna situación. Eligiéndolo como el creador se favorece el bajo acoplamiento. En este caso, la clase *ResolveObjectResource* es la encargada de crear objetos a partir de las especificaciones contenidas en el fichero de configuración *DefaultStrategies*. La intención de su aplicación en este contexto es permitir la adición y modificación de implementaciones concretas de interfaces a los componentes sin necesidad de recompilarlos.

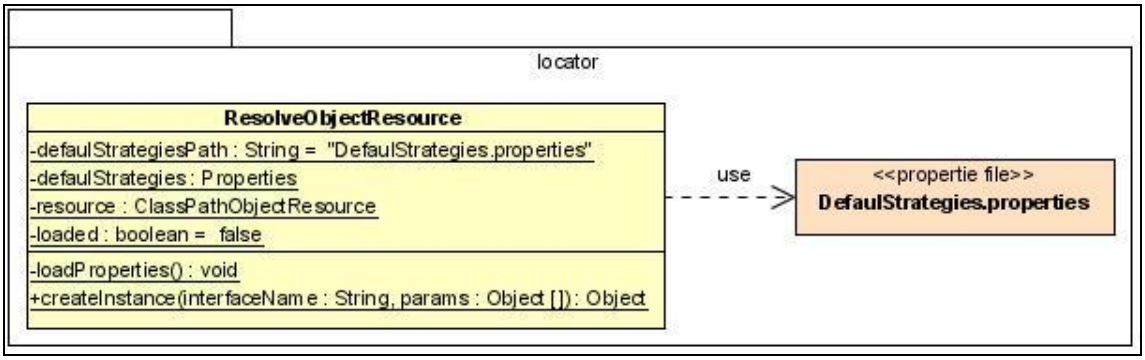


Figura44. Implementación del patrón Builder.

Observer: También conocido como *Listener* o *Publish-Subscribe*, es utilizado para disminuir el acoplamiento en sistemas donde varias clases necesitan conocer el estado de otras clases con las cuales se interrelacionan. La idea general es que uno o varios objetos (subscriptores) registren su interés en ser notificados cuando ocurra un cambio sobre otros objetos (publishers). Este patrón fue utilizado para dar soporte a la notificación a elementos interesados en recibir notificación, aunque con la variante de que los objetos que son modificados no son quienes se encargan de notificar el cambio, dejando esta responsabilidad para aquellos objetos que efectúan o ejecutan la transición. Los elementos interesados (subscriptores) son registrados en un archivo de configuración, del cual son obtenidos y procesados, y una vez que se realice un cambio de estado, son notificados informando de la acción realizada.

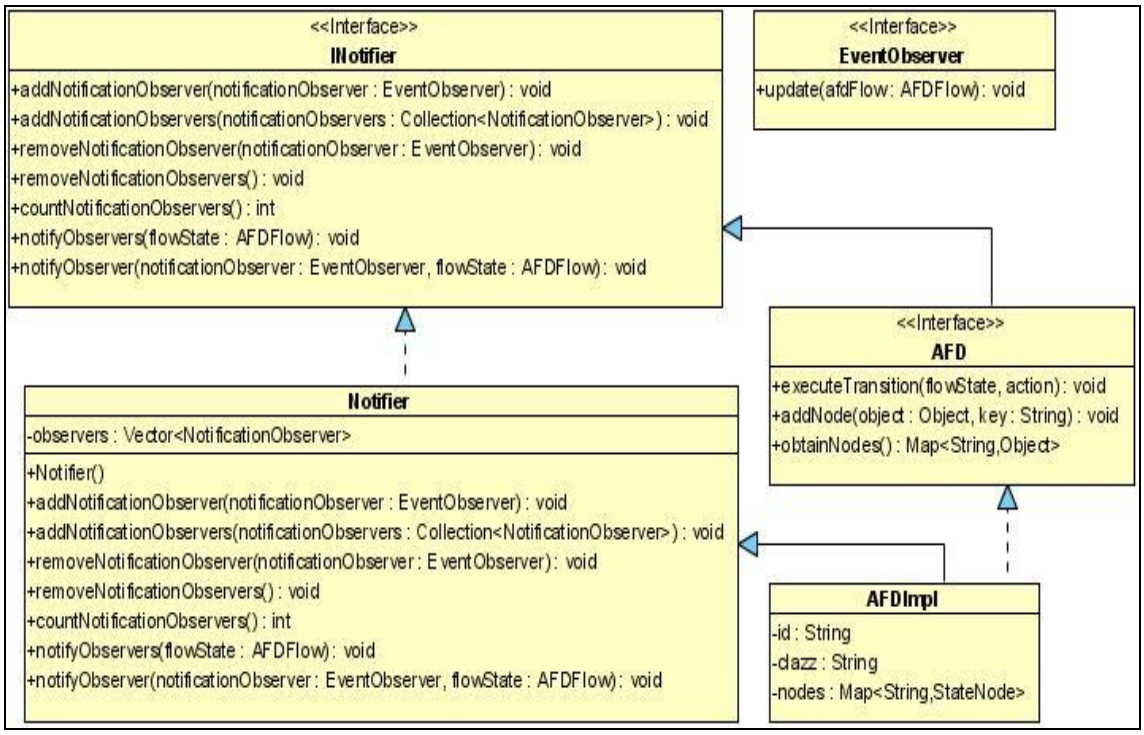


Figura45. Implementación del patrón Observer.

Composición: También conocido como *Composite*, es utilizado cuando se necesita crear una jerarquía de objetos y tratarlos uniformemente. La idea general es definir objetos primitivos y compuestos, en una definición recursiva de sí mismos. Su aplicación permite tratar uniformemente a los objetos simples y compuestos de una estructura jerárquica recursiva. Este patrón no se utilizo fielmente, sino que se realizó una modificación, que en realidad no afecta para nada su estructura. Se introdujo un mediador entre la composición del hijo con el padre para adicionarle algunos atributos necesarios.

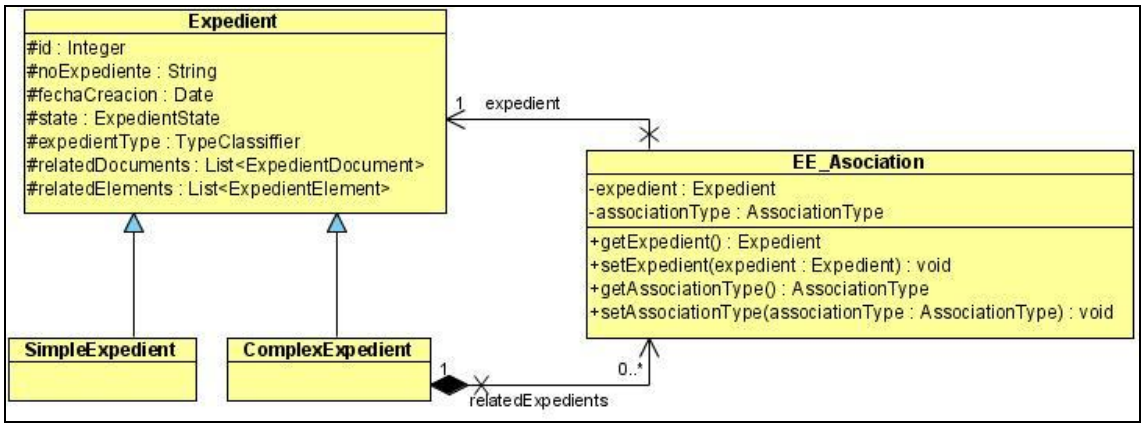


Figura46. Implementación del patrón Composición.

3.6 Conclusiones.

En este capítulo se realizó el análisis y diseño de la propuesta de solución. Se brindaron diferentes vistas UML con el objetivo de proporcionar una mayor comprensión de la solución propuesta. Se describió la lógica para la utilización de cada uno de los componentes en sus diferentes modos, así como las posibles vías para su extensión. Adicionalmente se describieron los patrones de diseño utilizados, proporcionando ejemplos concretos de su uso y las modificaciones de que fueron objeto.

CAPÍTULO 4

VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN

4.1 Introducción.

En el presente capítulo se valida la propuesta de solución con el desarrollo de una sencilla aplicación web donde se utilicen los componentes desarrollados. Los detalles de las herramientas y tecnologías seleccionadas para su confección fueron descritas en el *epígrafe 1.3.7* correspondiente al capítulo 1. Para ello se describen los requisitos funcionales que esta debe cumplir.

4.2 Requisitos de la Aplicación de Muestra.

Los requisitos que se describen a continuación están contenidos en los sistemas policiales estudiados con anterioridad, por tanto, responden a negocios reales, aunque no alcanzan a modelar procesos completos ya que el objetivo perseguido es realizar la integración de los componentes desarrollados en la implementación de un sistema policial. La aplicación de muestra poseerá una Agenda de Trabajo y un caso de uso de nivel usuario, accesible desde el menú, llamado Sustanciar Expediente. A continuación se especifican con mayor nivel de detalle estos requerimientos.

4.2.1 Agenda de Trabajo.

Se corresponde con un flujo de despacho por el que pasan los documentos. Permite modelar los procesos normales por los que pasa un documento durante su vida útil. Está formada por varias carpetas llamadas “bandejas” dado que su comportamiento es similar a las bandejas de correo electrónico.

A continuación se detallan los requisitos funcionales de la agenda de trabajo, especificando las bandejas seleccionadas y la información que se mostrará en cada una de ellas.

- Bandeja Aprobaciones: Los documentos que están emitidos y listos para revisar y aprobar. Brindando la opción de aprobarlo para validar su contenido.
- Bandeja Remisiones: Los documentos que están aprobados y listos para remitir. Brindando la opción de remitirlo hacia el despacho destino.

- Bandeja de Archivo: Los documentos que ya fueron remitidos y se encuentran archivados.

4.2.2 Sustanciar Expediente.

Con este caso de uso es posible validar la sustanciación de expedientes, el manejo de elementos investigativos, las asociaciones de elementos, documentos y expedientes, así como las notificaciones de cambio de estado.

4.2.2.1 Especificación general del Caso de Uso.

1. El sistema muestra los datos del expediente:
    a. Pestaña I: Estructura y Datos Generales.
        i. Datos planos.
        ii. Componentes (documentos y expedientes hijos – usar de ejemplo expedientes para cada Experticia realizada)
    b. Pestaña II: Elementos Investigativos Asociados.
        i. Elementos investigativos asociados, especificando el tipo de la asociación
    c. Pestaña III: Notificaciones Recibidas.
        i. Notificaciones de cambio de estado de elementos componentes y/o asociados al expediente, que se han recibido hasta el momento.
2. El usuario puede:
    a. Adicionar un documento nuevo.
        i. Asociar/Disociar elementos investigativos.
    b. Visualizar el contenido de un documento existente y a partir de este punto:
        i. Adicionar un documento en respuesta al presente documento.
        ii. Modificar el contenido del documento si no hay restricciones de estado.
            1. Asociar/Disociar elementos investigativos.
    c. Asociar/Disociar elementos investigativos del expediente.
    d. Modificar el tipo de la asociación del expediente con los elementos investigativos con los que se encuentra asociado.
    e. Cambiar el estado del expediente (de Iniciado a Remitido).
    f. Adicionar un nuevo Expediente de Experticia (Expediente hijo).
    g. Explorar el contenido de un Expediente hijo ya existente (Experticia en este caso).

4.3 Evaluación de la factibilidad del uso de los componentes desarrollados.

Con el uso del resultado de esta investigación en el desarrollo se logrará disminuir el tiempo de creación de los sistemas de gestión policial, una alta reusabilidad de sus componentes, gran flexibilidad lo que permite la realización de cambios en el código fuente de la aplicación sin impactos excesivos, y una alta mantenibilidad con un costo mínimo.

Los componentes están orientados a los desarrolladores de sistemas de gestión policial, aumentando la velocidad de desarrollo de los mismos. Están desarrollados sobre una plataforma libre, de código abierto, y la más utilizada en la universidad en la creación de este tipo de sistemas. Constituye un aporte práctico a la comunidad de desarrollo en sentido general. Es un primer paso para convertir los sistemas de gestión policial en productos que puedan ser comercializados en el mercado internacional con costo mínimo de desarrollo, en lugar de soluciones a la medida para un cliente específico.

Por todo lo planteado anteriormente se considera que es factible la utilización de los componentes genéricos desarrollados.

4.4 Conclusiones.

Con el desarrollo de la aplicación de muestra se realizó la integración de los componentes desarrollados en la implementación de varios requisitos funcionales comunes en un sistema policial, marcándose un aumento en la rapidez de desarrollo, la extensibilidad y posibilidades de reutilización. Por tanto, queda cumplido el objetivo principal de esta investigación, y se sientan las bases para la confección de un *framework* de negocio enmarcado en la creación de sistemas policiales como núcleo central del futuro polo productivo entorno a la creación de sistemas de este tipo en la universidad.

CONCLUSIONES GENERALES

Este trabajo demostró la necesidad de la existencia de componentes genéricos que automaticen el control de flujos de estados, brinde soporte para los conceptos y funcionalidades básicas relativas a la investigación y sus componentes, y maneje extensiones comunes de elementos investigativos, funcionalidades que constituyen la base para la automatización de procesos de investigación y la construcción de sistemas policiales. Adicionalmente, y como aporte teórico, develó procesos que constituyen el núcleo de un sistema policial, aspecto que debe ser tenido en cuenta en los futuros sistemas de este tipo que sean desarrollados.

Luego de un análisis de las tecnologías más usadas en la actualidad en la universidad para la construcción de sistemas informáticos similares, se elaboró la propuesta de utilizar el lenguaje Java, sobre el entorno de desarrollo integrado (IDE) *Red Hat Developer Studio*, como herramienta de modelado el *Visual Paradigm for UML 6.0*, y la utilización del lenguaje de marcado extensible XML para aumentar las posibilidades de reutilización y configuración.

Se llegó a la conclusión de que la metodología idónea para llevar a cabo el proceso de desarrollo es RUP (Proceso Unificado de Rational).

Se describieron los procesos identificados, los actores presentes, así como las actividades que son objeto de automatización.

Se definieron los requerimientos del sistema, tanto funcionales como no funcionales, y posteriormente se estructuró el modelo de casos de uso del sistema, proporcionándose una breve descripción de cada caso de uso.

Se realizó el diseño de los componentes propuestos, a través de vistas de paquetes, diagrama de clases, entre otros artefactos propuestos por la metodología.

Se plantearon los principios a seguir en el diseño de los componentes y algunas convenciones a respetar durante la escritura del código fuente.

Finalmente, se realizó una valoración sobre la factibilidad del uso de los componentes creados y se concluyó que su utilización genera una gran cantidad de ventajas para los desarrolladores de sistemas policiales.

Luego, se puede concluir que los componentes desarrollados dan solución a la situación problémica que los originó y que su explotación significará una mejora considerable en la calidad y eficiencia del desarrollo de sistemas policiales, cumpliéndose de esta forma los objetivos de la investigación.

RECOMENDACIONES

- Se recomienda el desarrollo de las funcionalidades que no fueron incluidas en la presente versión de la solución, como por ejemplo el tratamiento de la información histórica, de manera que se brinde un soporte estándar para la mayor parte de los procesos que automatizan los sistemas policiales, y de esa forma ir construyendo paulatinamente un *framework* que permita el rápido desarrollo de sistemas policiales.
- Se recomienda la profundización en el estudio de los procesos de negocio de otras instituciones policiales, a fin de perfeccionar el modelo genérico de procesos de negocio de este tipo de instituciones y permitir una refinación de los requisitos de los componentes desarrollados.
- Además, se recomienda la aplicación de los componentes desarrollados en un proyecto real (a modo de prueba piloto) de modo que puedan detectarse defectos e identificar nuevos requisitos y así permitir su evolución.

BIBLIOGRAFÍA

1. **Ortiz Anderson, César.** gacetaucayalina.com. *gacetaucayalina.com.* [En línea] 31 de 08 de 2008. http://www.gacetaucayalina.com/2008/08/tengamos-claro-que-la-seguridad-ciudadana-es-un-bien-publico.html.

2. Baufest. *Baufest.* [En línea] 2006. http://www.baufest.com/spanish/scrum/scrumconference2006/Que_es_scrum.pdf.

3. **H. Canós, José, Letelier, Patricio y Penad, María Carmen.** *Métodologías Ágiles en el Desarrollo de Software.* Valencia : Universidad Politécnica de Valencia, 2008.

4. **Rational.** Rational. *Rational.* [En línea] http://www.rational.com.ar/herramientas/roseenterprise.html.

5. **Hamar, Vanessa.** *Aspectos metodológicos del desarrollo y reutilización de componentes de software.* 2004.

6. **García Vicente, Diana y Ruiz Durán, Susel.** *Ágora: Centro Comercial Virtual para la UCI.* Habana : s.n., 2005.

7. **Ballester Marsal, Andrés y Notario Laborí, Alejandro.** *Sistema de Reservaciones.* Habana : s.n., 2005.

8. **Soria Ramírez, Jorge Amado y Altuna Castillo, Enrique José.** *Análisis, Diseño e Implementación de un Jurado Online.* Habana : s.n., 2007.

9. **ALBET.** *Procesos Elementales del Negocio CICPC.* Caracas : s.n., 2006.

10. **Asamblea Nacional Constituyente.** Artículo 55. [aut. libro] Asamblea Nacional Constituyente. *Constitución de la República Bolivariana de Venezuela.* Caracas : s.n., 1999.

11. **ALBET.** *Procesos Elementales de Negocio del Sistema de Gestión Policial.* Caracas : s.n., 2007.

12. —. *Modelo de Casos de Uso del Módulo de Reseña. SIGEPOL.* Habana : s.n., 2007.

13. —. *Modelo de Casos de Uso del Módulo de Operativos Policiales. SIGEPOL.* Habana : s.n., 2007.

14. —. *Modelo de Casos de Uso del Módulo de Denuncias. SIGEPOL.* Habana : s.n., 2007.

15. **Wikepedia.** wikipedia. *wikipedia.* [En línea] http://es.wikipedia.org/wiki/Ministerio_del_Interior_(Cuba).

16. Guía de Usuario de Enterprise Architect 7.1. *Guía de Usuario de Enterprise Architect 7.1.* [En línea] http://www.sparxsystems.com.ar/download/Ayuda%20HTML/index.html?statediagram.htm.

17. **Ince, Darrel, [trad.].** *Ingeniería del Software. Un enfoque práctico.* 5ta. Roger.S.Pressman.

GLOSARIO DE TÉRMINOS

A continuación se presenta el significado de algunos términos usados en este documento que no son de uso común y que pueden dificultar la compresión del mismo. El objetivo principal es que la presente investigación sea comprendida por la mayor cantidad de personas, posean o no conocimientos de computación y/o del funcionamiento de las organizaciones policiales. Por esta razón, son incorporados en esta sección términos, conceptos y expresiones comunes de informática y del proceso de desarrollo de software.

1. Aplicaciones web: Son aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores web (HTML, JavaScript, Java, etc.) en la que se confía la ejecución al navegador.

2. Árboles: Es un grafo en el que dos vértices están conectados por *exactamente un* camino. Un **bosque** es un grafo en el que dos vértices cualquiera están conectados por *como máximo un* camino. Una definición equivalente es que un bosque es una unión disjunta de árboles.

3. Arquitectura distribuida cliente-servidor: Es el procesamiento cooperativo de la información por medio de un conjunto de procesadores, en el cual múltiples clientes, distribuidos geográficamente, solicitan requerimientos a uno o más servidores centrales. Desde el punto de vista funcional, se puede definir como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información de forma transparente aún en entornos multiplataforma. Se trata pues, de la arquitectura más extendida en la realización de Sistemas Distribuidos.

4. Arquitectura: Según "El proceso unificado de desarrollo de software" de Jacobson-Grady-Rumbaugh", es el conjunto de planos de un desarrollo de software. Planos con las características más importantes resaltadas dejando de lado los detalles. características como requisitos de los usuarios e inversores, plataforma (sistema operativo, hardware, base de datos, protocolos de red), bloques de construcción reutilizables, consideraciones de implantación, sistemas heredados y requisitos no funcionales.

5. Autómata: Dispositivo o conjunto de reglas que realizan un encadenamiento automático y continuo de operaciones capaces de procesar una información de entrada para producir otra de salida.

6. Automatizar: En informática, se refiere al conjunto de métodos que sirven para realizar tareas repetitivas en un computador.

7. Bean: Un bean es el elemento básico con el que se desarrolla una aplicación Java, típicamente un objeto simple. En dependencia de la tecnología elegida al declarar un bean también suele ser necesario especificarle algunas propiedades, específicamente la clase y el identificador. Un bean puede ser identificado por id o por nombre, o sea, por los atributos *id* o *name*.

8. Bucle: Secuencia de instrucciones que se repite mientras se cumpla una condición prescrita. Debe contener condiciones que establezcan cuándo empieza y cuándo acaba, de manera que, mientras las condiciones se cumplan, ejecute una secuencia de código de manera repetitiva.

9. Clase: Es un contenedor de uno o más datos (variables o propiedades miembro) junto a las operaciones de manipulación de dichos datos (funciones/métodos). Las clases pueden definirse como estructuras, uniones o clases, pudiendo existir diferencias entre cada una de las definiciones según el lenguaje. Además son agrupaciones de objetos que describen su comportamiento

10. Classpath: En el lenguaje de programación Java se entiende por Classpath una opción admitida en la línea de órdenes o mediante variable de entorno que indica a la Máquina Virtual de Java dónde buscar paquetes y clases o recursos definidos por el usuario a la hora de ejecutar programas.

11. Cliente Enriquecido: Es un término medio entre el cliente liviano y el cliente pesado. El objetivo del cliente enriquecido consiste en proporcionar una interfaz gráfica, escrita con una sintaxis basada en XML, que proporciona funcionalidades similares a las del cliente pesado (arrastrar y soltar, pestañas, ventanas múltiples, menús desplegables). Los clientes enriquecidos también pueden realizar un procesamiento fundamental en el lado del servidor. Seguidamente, los datos se envían con un formato de intercambio estándar, al utilizar la sintaxis de XML, que después el cliente enriquecido interpreta.

12. Cliente liviano: Es una computadora cliente o un software de cliente en una arquitectura de red cliente-servidor que depende primariamente del servidor central para las tareas de procesamiento, y principalmente se enfoca en transportar la entrada y la salida entre el usuario y el servidor remoto.

13. Codificación: Operación consistente en representar una información mediante un código, por ejemplo, representar cada carácter alfanumérico mediante de un conjunto de bits valor 0 a 1. Expresión de un conjunto de datos o comandos en un lenguaje simbólico de programación que puede ser procesado por el ordenador.

14. Componente: Unidad de composición con interfaces contractuales especificadas y dependencias de contexto explícitas. Un componente de software puede ser desarrollado independientemente y utilizado por terceras partes para integrarlo mediante composición a sus sistemas. Para que un componente pueda ser integrado por terceras partes en sus sistemas, éste deber ser suficientemente autocontenido y debe proveer una especificación de lo que requiere y provee. En otras palabras, los componentes deben encapsular su implementación e interactuar con otros componentes a través de interfaces bien definidas.

15. Contenedor web: Un contenedor que implementa el contrato de componente web de la arquitectura J2EE. Este contrato especifica un entorno de tiempo de ejecución para componentes web que incluye servicios de seguridad, concurrencia, administración del ciclo de vida, transacción, implementación y otros servicios. Un contenedor web proporciona los mismos servicios como contenedor JSP así como vista federada de las API de la plataforma J2EE. Un contenedor web lo proporciona una web o un servidor J2EE.

16. CVS: Concurrent Versions System (CVS), también conocido como Concurrent Versioning System, es una aplicación informática que implementa un sistema de control de versiones: mantiene el registro de todo el trabajo y los cambios en los ficheros (código fuente principalmente) que forman un proyecto (de programa) y permite que distintos desarrolladores (potencialmente situados a gran distancia) colaboren. CVS se ha hecho popular en el mundo del software libre.

17. Desarrollo web: Conjunto de tecnologías de software del lado del servidor y del cliente que involucran una combinación de procesos de base de datos con el uso de un navegador en internet a fin de realizar determinadas tareas o mostrar información.

18. Disparador de transición: Es la ocurrencia del evento que desencadena una transición de estado.

19. Encapsular: Mecanismo que consiste en organizar datos y métodos de una estructura, evitando el acceso a estos por cualquier otro medio distinto a los especificados, garantizando la integridad de los datos.

20. Entorno monolítico: Marco de trabajo que no puede ser extendido mediante plugins.

21. Estandarización: Proceso de adaptación o adecuación a un modelo, o de normalización de los componentes comunes que integran determinados procesos, flujos, o negocios, con el propósito de incrementar la productividad en su desarrollo.

22. Excepción: Es una estructura de control de los lenguajes de programación diseñada para manejar condiciones anormales que pueden ser tratadas por el mismo programa que se desarrolla.

23. Expediente Tanatológico: Expediente utilizado para archivar los resultados de la tanatología, disciplina que estudia el fenómeno de la muerte en los seres humanos. En él son almacenados todas las experticias realizadas y documentación relacionada con un cadáver. Tiene un formato básico, que incluye la información del levantamiento y de la autopsia, pero además se le adjunta toda la documentación generada sobre el cadáver.

24. Experticias: Prueba pericial donde peritos examinan una o más pruebas. Medio de prueba, o procedimiento a través del cual se prueba algo ante un tribunal u otro ente parecido.

25. Flagrancia: Acción que ocurre o se realiza en el momento presente.

26. Framework: es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

27. Generalizar: Es un elemento fundacional de la lógica y el razonamiento humano. Extensión o propagación de algo. Conclusión general que se saca de algo particular.

28. Guarda de restricción: Es una expresión lógica escrita en términos de los parámetros del evento disparado, y de los atributos y enlaces del objeto al que pertenece la máquina de estados.

29. Instanciar: Acción y efecto de crear una instancia. El crear en memoria un ejemplar de un conjunto de datos y código definido por una clase o estructura

30. Interfaz: Es uno de los componentes más importantes de cualquier sistema computacional, pues funciona como el vínculo entre el humano y la máquina. Es un conjunto de protocolos y técnicas para el intercambio de información entre una aplicación computacional y el usuario. La IU es responsable de solicitar comandos al usuario, y de desplegar los resultados de la aplicación de una manera comprensible.

31. J2EE: Es una plataforma de programación, parte de la Plataforma Java, para desarrollar y ejecutar software de aplicaciones en lenguaje de programación Java con arquitectura de N niveles distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

32. Log: Archivo que registra toda la actividad de un servidor, aplicación o software. El mismo es presentado cronológicamente con datos adicionales muy detallados que se utilizan generalmente para llevar estadísticas de uso de un determinado sitio, aplicación o software.

33. Navegadores: Es una aplicación software que permite al usuario recuperar y visualizar documentos de hipertexto, comúnmente descritos en HTML, desde servidores web de todo el mundo a través de Internet.

34. Negocio: Término utilizado en ingeniería de software para referirse al conjunto de tareas relacionadas lógicamente que son llevadas a cabo para lograr un resultado de definido.

35. POJOS: Acrónimo de *Plain Old Java Object*, es una sigla creada por Martin Fowler, Rebecca Parsons y Josh MacKenzie en septiembre de 2000 y utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un *framework* en especial. Este acrónimo surge como una reacción en el mundo

Java a los *frameworks* cada vez más complejos, y que requieren un complicado andamiaje que esconde el problema que realmente se está modelando.

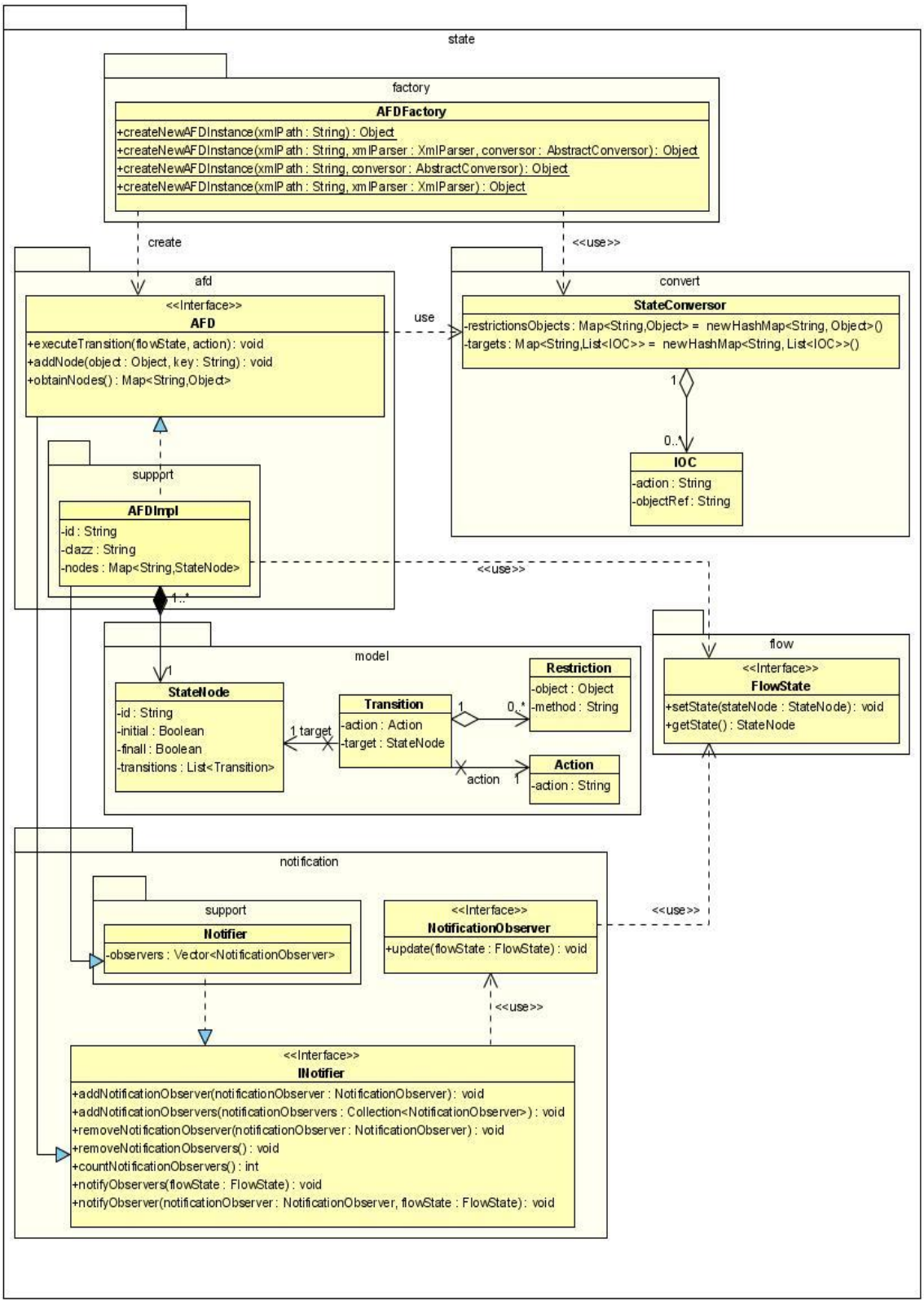
36. Portabilidad: Característica de ciertos programas que les permite ser utilizados en distintos ordenadores sin que precisen modificaciones de importancia.

37. Spring: Es un *framework* de aplicaciones de código abierto para la plataforma Java. introdujo técnicas y funcionalidades inexistentes con anterioridad, lo que resulta en un *framework* que ofrece un modelo consistente y aplicable a la mayor parte de las aplicaciones que se pueden crear en esta plataforma. Puede considerarse una colección de varios *frameworks* más pequeños.

38. Sustanciación: Proceso de recopilación o compendio de datos o elementos con un fin determinado.

ANEXOS

Anexo1. Diagrama de Clases. Componente Gestión de Estados.



Anexo2. Vista gráfica del fichero de configuración del flujo de estados (XSD).