Universidad de las Ciencias Informáticas

Facultad 3

Diseño e implementación de un componente para el Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos que informatice los procesos Prueba Testifical y de Reconocimiento.

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas

Autores: Aymeé Reina Rodríguez
Eddy Santana Navarro

Tutor: Ing. Maikel Navarro Díaz
Co-tutor: Ing. Adilaraima Martínez Barrio

La Habana, Cuba
Curso: 2013-2014

El futuro tiene muchos nombres. Para los débiles es lo inalcanzable.

Para los temerosos, lo desconocido. Para los valientes es la oportunidad.

VICTOR HUGO

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____	_____
Aymeé Reina Rodríguez	Eddy Santana Navarro
Autor	Autor
_____	_____
Ing. Maikel Navarro Díaz	Ing. Adilaraima Martínez Barrio
Tutor	Co-tutor

A mi abuela Luisa por haber cuidado siempre de mí, por tantos sacrificios para hacerme feliz y porque gracias a ella me he convertido en la persona que soy.

A mi mamá por quererme, apoyarme y por ser un ejemplo de lucha diaria.

A mi papá por quererme y preocuparse siempre por mí.

A mi abuela Nery por estar siempre pendiente de todo.

A mi hermana Jenni porque a pesar de todas las peleas me ha apoyado siempre que la he necesitado.

A mis hermanos Abraham y Elizabeth por su cariño incondicional.

A mi hermano Jeremy por ser capaz de sacarme una sonrisa sin importar el momento.

A mi novio por quererme a pesar de mi carácter.

A mis compañeros del 3501 y a todos aquellos que alguna vez fueron el 3101 por los buenos y malos momentos que nos ayudaron a llegar hasta aquí.

A todas las amistades que he hecho en estos cinco años y que han colaborado para hacer de esta una experiencia inolvidable.

A mi compañero de tesis por compartir conmigo esta tarea con paciencia y dedicación.

A nuestros tutores por habernos guiado durante todo este proceso y por apoyarnos.

A la Revolución Cubana por haberme dado la oportunidad de estudiar en una universidad como esta.

Aymeē Reina Rodríguez

Ante todo agradecer a mis padres por el apoyo que me han brindado durante toda mi vida, por haber estado allí siempre que los he necesitado.

A Mima por haber sido como otra madre para mí, por haberme consentido en todo y siempre darme ánimos cuando más falta me hacía.

A Pipo quien siempre ha sido mi más grande ejemplo a seguir en la vida y a quien siempre llevo conmigo adonde quiera que vaya.

A toda mi familia que siempre ha estado pendiente de mí y me ha ayudado a llegar hasta donde me encuentro hoy día.

A los compañeros que he conocido durante todos mis años de estudio, que han aportado, un poco cada uno, a convertirme en la persona que hoy ven ante ustedes.

A todos esos profesores que han contribuido, de una forma u otra, a mi formación como profesional, y particularmente a aquellos que han sabido con sus enseñanzas consolidarme como un hombre de bien.

Finalmente, a mi compañera de tesis por haber confiado en mí para juntos llevar adelante esta enorme tarea y por su esfuerzo y dedicación con la misma; y a nuestros tutores por todo el tiempo que nos dedicaron y todo el apoyo que nos han dado.

Eddy Santana Navarro

A mi abuela Luisa y a mis padres por haber hecho de mí la persona que soy con todo su cariño y dedicación.

A mis hermanos por quererme siempre y por ayudarme a levantar mi ánimo con una sonrisa.

Aymeé Reina Rodríguez

A mis padres, mi abuela y el resto de mi familia por todos los sacrificios que han hecho por mí y por estar siempre ahí.

A Juanillo y a Pipo, mis abuelos, que ya no se encuentran entre nosotros y con quienes hubiera querido poder disfrutar este logro.

Eddy Santana Navarro

Resumen

En la actualidad los procesos Prueba Testifical y de Reconocimiento de los Tribunales Populares Cubanos (TPC) se realizan de forma manual. Este hecho da lugar a una serie de errores como la duplicación de resoluciones y asientos en los libros y equivocaciones al registrar la información proveniente de las partes. Todo esto provoca lentitud en la tramitación de dichos procesos. Los jueces y abogados que hacen uso de la ley pueden realizar diferentes interpretaciones de la misma, lo que evidencia poca uniformidad en la realización de estos procesos. El presente trabajo tiene como objetivo desarrollar un componente que informatice los requisitos existentes de los procesos Prueba Testifical y de Reconocimiento para mejorar la uniformidad en la tramitación y la gestión de la información en los TPC. Para la creación de este componente se realizó un estudio de la metodología de desarrollo a utilizar, así como de las diferentes tecnologías y herramientas necesarias para realizar el diseño y la implementación del mismo. Fueron generados los artefactos correspondientes al diseño y la implementación de la solución. Estos artefactos fueron validados mediante métricas de diseño, pruebas de caja blanca y pruebas de caja negra. Como resultado se obtuvo un componente que puede contribuir a mejorar la uniformidad en la tramitación y la gestión de la información de los procesos Prueba Testifical y de Reconocimiento en los TPC. Este componente actualmente se encuentra en pruebas de liberación en el centro CALISOFT de la Universidad de las Ciencias Informáticas.

Palabras claves: Prueba de Reconocimiento, Prueba Testifical, Tribunales Populares.

Índice de contenido

Índice de imágenes

Índice de tablas

Índice de gráficos

Introducción

Las Tecnologías de la Información y las Comunicaciones (TIC) se han desarrollado en gran medida en los últimos años. Tanto es así que en la actualidad resulta casi indispensable su uso en muchas de las esferas de la sociedad. El término TIC está relacionado con todos los aspectos del manejo, procesamiento y comunicación de información. La automatización oportuna de procesos mediante el uso de la informática favorece la disminución de los costos, el incremento de la productividad del trabajo, agiliza la toma de decisiones y contribuye a una cultura organizacional más eficiente en la dirección de cualquier proceso social.

Cuba no está ajeno a este desarrollo y el estado invierte numerosos recursos para lograr insertar la informatización en todas las esferas sociales. Ejemplo de ello lo constituye el sector jurídico y como parte de este los Tribunales Populares Cubanos (TPC). Estos constituyen un sistema de órganos estatales, estructurados con independencia funcional de cualquier otro, y sólo subordinados, jerárquicamente, a la Asamblea Nacional del Poder Popular y al Consejo de Estado (1).

Con el objetivo de informatizar los diferentes procesos judiciales que se llevan a cabo en los TPC, en la Universidad de las Ciencias Informáticas (UCI) se desarrolla el Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos (SITPC). El desarrollo de este sistema cuenta como otro importante paso en la lucha de Cuba para lograr la informatización como un medio más para alcanzar una sociedad cada vez más justa, equitativa y solidaria, permitiendo su evolución y la disminución de la brecha digital que existe en la actualidad. El SITPC permitirá informatizar los procesos que maneje, trayendo consigo impactos positivos en la sociedad, entre los cuales se encuentra una mayor eficacia y eficiencia en todos los procesos que opere, y por consiguiente, un aumento en la calidad del trabajo que se desarrolla en los TPC, contribuyendo a disminuir el tiempo que demoran las gestiones en estas entidades.

Para la correcta administración de justicia los TPC se encuentran estructurados en tres instancias: Tribunal Supremo Popular, Tribunales Provinciales Populares y Tribunales Municipales Populares. En estas instancias se llevan a cabo diferentes procesos distribuidos en cinco materias: Penal, Administrativo, Civil, Laboral y Económico.

De acuerdo a la Ley de Procedimiento Civil Administrativo Laboral y Económico (LPCALE), algunos procesos de diferentes materias se tramitan de igual forma. Entre estos procesos están las pruebas judiciales, que pueden ser de varios tipos, entre ellos se encuentran las pruebas testificales y de reconocimiento.

En la tramitación de estas pruebas se gestiona gran cantidad de información proveniente tanto de las partes como de las resoluciones y actas creadas. La gestión de la información es un proceso que incluye operaciones como almacenamiento, manipulación y control de la información adquirida y que gestiona el acceso y los derechos de los usuarios sobre la misma.

Actualmente la tramitación de las pruebas judiciales en los TPC se atiende de forma manual, confeccionándose cuadernos independientes al expediente por cada tipo de prueba, este hecho da lugar a una serie de problemas tales como:

- Errores a la hora de registrar una información proveniente de las partes.
- El uso del papel para las notificaciones ocupa el 60% del tiempo hábil de un proceso.
- No hay un óptimo control de la información pues pueden existir duplicaciones de resoluciones y de asientos en los libros de la sala.
- El almacenamiento se realiza en estantes donde los documentos se encuentran expuestos al deterioro por la humedad del local, fumigación, polvo, entre otros y en ocasiones a la pérdida.
- La manipulación de los documentos se vuelve complicada ya que al encontrarse en estantes la búsqueda se torna engorrosa.
- Los jueces y abogados a la hora de realizar cada trámite deben tener en cuenta la forma de tramitar el procedimiento según la LPCALE y al existir una forma de trabajo poco centralizada, se infieren diferentes interpretaciones al procedimiento, incluso en ocasiones puede llegarse a violar algún término importante; evidenciándose poca uniformidad en la tramitación.

Hasta el momento se cuenta con la especificación de los requisitos funcionales y no funcionales de los procesos Prueba Testifical y de Reconocimiento firmados por el cliente.

Con todo lo expuesto hasta el momento se plantea el **problema a resolver**:

¿Cómo contribuir a mejorar la uniformidad en la tramitación y la gestión de la información en los procesos Prueba Testifical y de Reconocimiento en los Tribunales Populares Cubanos?

Se define como **objeto de estudio**:

Procesos Prueba Testifical y de Reconocimiento en la informática jurídica de gestión.

Teniendo en cuenta el problema planteado, el presente trabajo tiene como **objetivo general**:

Desarrollar un componente que informatice los requisitos existentes de los procesos Prueba Testifical y de Reconocimiento para mejorar la uniformidad en la tramitación y la gestión de la información en los Tribunales Populares Cubanos.

Teniendo como **campo de acción**:

Desarrollo de sistemas informáticos para la informática jurídica.

Como **Idea a defender** se plantea:

Con el diseño e implementación de los requisitos de los procesos Prueba Testifical y de Reconocimiento se contribuirá a la mejora de la uniformidad en la tramitación y la gestión de la información en el Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos.

Para darle cumplimiento al objetivo general se plantean los **objetivos específicos**:

- Definir el marco teórico de la investigación para conceptualizar los elementos fundamentales que serán utilizados.
- Diseñar una propuesta de solución para tener una representación técnica de la solución a implementar.
- Implementar la solución para obtener un componente de software que contribuya a la mejora de la uniformidad en la tramitación y la gestión de la información en el Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos.
- Validar la solución propuesta mediante pruebas de software y el cumplimiento de los objetivos de la investigación para comprobar la correcta funcionalidad del componente obtenido.

Para dar cumplimiento a los objetivos se emplearon métodos científicos de investigación. Para ello se utilizaron los métodos teóricos y empíricos.

Métodos teóricos:

Analítico – Sintético

Permitió realizar un estudio teórico sobre la forma en la que se realizan las pruebas testificales y de reconocimiento, analizando cada una de las partes que componen estos procesos. La síntesis permitió sacar los elementos comunes entre cada una de estas partes para extraer los elementos más importantes relacionados con el objeto de estudio.

Histórico-Lógico

Este método permitió la realización de un estudio de la evolución de las pruebas judiciales y los cambios que han sufrido estos procesos con el desarrollo de las TIC; así como de los antecedentes y componentes de los sistemas informáticos existentes que realizan pruebas judiciales para analizar si existe algún sistema informático que satisfaga los requisitos acordados con el cliente.

Modelación

La modelación es uno de los métodos más importantes dentro de la construcción de software. La utilización de este método permitió crear los modelos y diagramas para explicar el flujo de acciones a seguir y las relaciones existentes entre los distintos componentes de los procesos Prueba Testifical y de Reconocimiento.

Métodos empíricos:

Medición

Este método se utilizó en la métricas de validación del diseño para obtener información numérica de los distintos atributos de calidad que estas miden y de esta forma poder determinar si fue o no correcto el diseño realizado.

Simulación

La simulación es un método experimental controlado que permitió la creación de los diferentes casos de prueba con datos artificiales. Estos casos de prueba fueron utilizados para comprobar la correcta ejecución de los flujos básicos y alternos correspondientes a cada caso de uso.

Para dar solución a esta problemática el trabajo constará de 3 capítulos, organizados de la siguiente manera:

Capítulo 1- Fundamentación teórica. En este capítulo se recoge la fundamentación teórica que sustenta el desarrollo del componente propuesto. Se relacionan aspectos como el estado del arte, la metodología, arquitectura, lenguajes de programación y herramientas de desarrollo definidos para la realización del componente y la contribución de los mismos al cumplimiento de los requisitos previamente identificados.

Capítulo 2- Propuesta de solución. En este capítulo se propone la solución de la investigación. Se presenta la arquitectura definida para el componente, así como los modelos de diseño, de implementación y de despliegue; cada uno de ellos con los artefactos generados, como diagrama de clases, de secuencia, de componentes y de despliegue. Se incluyen además otros elementos de la implementación del componente, así como el resumen de las especificaciones de los casos de uso y los patrones de diseño utilizados en la solución.

Capítulo 3- Validación de la solución. En este capítulo se analizan los resultados obtenidos, basándose en métricas de diseño y pruebas de software que se realizan a la solución propuesta con el fin de determinar si cumple las expectativas del cliente.

Capítulo 1: Fundamentación teórica.

1.1. Introducción

El presente capítulo tiene como objetivo describir los elementos necesarios para entender los procesos Prueba Testifical y de Reconocimiento a fin de realizar su diseño e implementación. Se realiza un estudio de los sistemas existentes en Cuba y el mundo con características similares al que se debe desarrollar. Se analizan los componentes de software y los elementos esenciales relacionados con el componente a desarrollar. También se hace un estudio de los fundamentos teóricos del proceso de desarrollo de software incluyendo metodologías, marcos de trabajo, lenguajes y herramientas de desarrollo.

1.2. Procesos Prueba Testifical y de Reconocimiento

La prueba judicial es el intento de las partes de conseguir el convencimiento psicológico del juez con respecto de la existencia, la veracidad o la falsedad de determinados datos (2), y ha llegado a ostentar una gran importancia en los procesos jurídicos.

El proceso de las Pruebas Judiciales abarca 7 tipos de pruebas: Testificales, de Reconocimiento (para objeto, lugar y cosa), Pericial, Documentales y de Reproducciones, de Confesión judicial, de Libro y de Presunción. Las que serán abordadas en el presente trabajo son las pruebas testificales y de reconocimiento.

La prueba de reconocimiento judicial es aquella dirigida a lograr que el juez o tribunal realicen un examen directo de lugares, objetos o personas, cuando dicha percepción resulte necesaria o conveniente a los efectos de la apreciación o esclarecimiento de los hechos objeto del proceso (3). Es una diligencia procesal, practicada por un funcionario judicial, con el objeto de obtener argumentos de prueba para la formación de su convicción, mediante el examen y la observación con sus propios sentidos, de hechos ocurridos durante la diligencia o antes pero que aún subsisten o de rastros o huellas de hechos pasados, y en ocasiones de su reconstrucción (4). Este es un medio de prueba de carácter directo debido a que el órgano judicial tiene un contacto inmediato con el objeto de la prueba, sin delegar dicha función en otra parte (3).

En Cuba, el Artículo 316 de la Legislación Cubana, aclara que el reconocimiento judicial se acordará, de oficio[1] o a instancia de parte[2], cuando para el esclarecimiento y apreciación de los hechos sea necesario que el Tribunal examine por sí mismo cosas, lugares o personas.

La prueba testifical es un medio probatorio de naturaleza personal, en el que la fuente de prueba está constituida por el testigo y su conocimiento de los hechos (5). El testigo es una persona física ajena al proceso que aporta una declaración sobre hechos percibidos, vistos y oídos por ella o que ha sabido de referencia y sobre los cuales se dirige al Tribunal para ser interrogada (6).

En este tipo de prueba siempre existen determinadas personas que se consideran inhábiles para declarar como testigos, según algunas condiciones que varían en dependencia del país donde se desarrolla el proceso. En Cuba se consideran inhábiles los que están privados del uso de la razón; los ciegos y sordos, para declarar sobre hechos cuyo conocimiento dependa, respectivamente, de la vista y el oído; y los menores de doce años. También, según el Artículo 328 de la Legislación Cubana, están exentos de la obligación de declarar como testigos: los que tengan interés directo en el pleito; los ascendientes en los pleitos de los descendientes y estos en los de aquellos; el marido en los pleitos de la mujer y la mujer en los del marido; el suegro o la suegra en los pleitos del yerno o la nuera, y viceversa; y el hermano de cualquiera de los involucrados.

Es necesario aclarar que según el Artículo 348 de la Legislación Cubana, los Tribunales apreciarán el valor probatorio de las declaraciones de los testigos conforme a los principios y reglas de la lógica, teniendo en consideración la razón de conocimiento que hubieren dado y las circunstancias que en ellos concurran.

1.3. Sistemas informáticos que incluyen las pruebas judiciales.

En el momento actual, caracterizado por la aplicación de las TIC a todos los ámbitos de la sociedad, el sistema judicial debe afrontar la revolución tecnológica para aprovechar las ventajas que le ofrece. Es por ello que se viene promoviendo el uso de la informática en esta rama, por lo que existen numerosas aplicaciones en todo el mundo para facilitar la gestión de los procesos judiciales. Un elemento importante en estos procesos lo constituyen las pruebas. Las nuevas tecnologías han influenciado notablemente la prueba documental que en la actualidad puede estar contenida en soportes magnéticos e informáticos. Esta

[1] El juez inicia el procedimiento sin que exista denuncia o solicitud de un agente externo.
[2] La persona interesada realiza la solicitud para iniciar el procedimiento.

práctica de prueba a través de soportes informáticos exige, en no pocas ocasiones, la intervención en el proceso de técnicos en la materia para, entre otras cosas, autentificar la procedencia y el contenido de los soportes informáticos aportados, dando lugar una nueva prueba pericial. Otra de las pruebas que ha sido modernizada es la Testifical. En países como Italia, Francia, Singapur y Estados Unidos se permite la utilización de la videoconferencia para obtener el testimonio de personas, a fin de proteger a las víctimas, los testigos o el acusado[3].

A continuación se realizará un análisis de sistemas informáticos que incluyen pruebas judiciales con el fin de analizar si existe alguno que cumpla con las funcionalidades necesarias de los procesos Prueba Testifical y de Reconocimiento.

Sistema	País en el que fue implementado	Tipo de prueba que realiza	Materia en la que se utiliza	Multiplataforma	Herramienta privativa	Notificaciones automáticas
LexNet	España	Documental	-	Sí	Sí	No
LawNet	Singapur	Documental	Civil	Sí	Sí	No
Justice Online	Singapur	Testifical	Penal	Sí	Sí	No
H@bilus	Portugal	Documental	-	Sí	Sí	Sí

TABLA 1 ANÁLISIS DE SISTEMAS QUE IMPLEMENTAN PRUEBAS JUDICIALES

El mayor percance hasta el momento viene dado por la imposibilidad de generalizar en un sistema el desarrollo de un proceso judicial debido a las diferencias en las leyes de los distintos países. Incluso cuando estos sistemas permiten realizar algunos tipos de pruebas, no abarcan todos los tipos que existen en Cuba. Los sistemas Lexnet, Lawnet y H@bilus no permiten la realización de las pruebas testificales ni de reconocimiento. El sistema Justice Online solo se utiliza en conferencias de antelación a juicio en la materia penal, el componente que se necesita para dar solución al problema general es para las materias Civil, Administrativo, Laboral y Económico que son las que se rigen por la LPCALE, ya que la materia penal se rige por otra ley. Además este sistema no permite la realización de la prueba de reconocimiento. Todos estos sistemas son herramientas privativas por lo que no es posible realizar su distribución o modificación

[3] En mayo de 2006 se inició el juicio en el Tribunal de Apelaciones de Palermo (Sicilia) contra Provenzano, jefe de la Cosa Nostra que fue arrestado esa primavera tras estar huido de la justicia desde 1963. Al acusado se le tomó declaración a través de videoconferencia en la que aportó varios documentos y mantuvo una larga conversación telefónica con su abogado.

libremente. Otro aspecto importante es que a excepción de H@bilus, el resto de los sistemas no realizan las notificaciones automáticas al vencimiento de los términos. Estas son las razones principales de que no se pueda aplicar ninguno de estos sistemas a los Tribunales Populares Cubanos de forma que permita la administración correcta de la justicia.

1.4. Metodología de desarrollo.

Una metodología impone un proceso disciplinado sobre el desarrollo de software con el objetivo de hacerlo más predecible y eficiente, por tanto, define un camino reproducible para obtener resultados confiables (7).

Proceso Unificado de Desarrollo (RUP)

RUP es un proceso formal: provee un acercamiento disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad que satisfaga los requerimientos de los usuarios finales (respetando cronograma y presupuesto) (8).

RUP tiene 3 características esenciales:

- Dirigido por casos de uso
- Centrado en la arquitectura
- Iterativo e incremental.

Los casos de uso no son utilizados solamente para especificar los requisitos del sistema sino que son un hilo conductor para el diseño, implementación y prueba. Además de utilizar los casos de uso para guiar el proceso también es importante establecer una buena arquitectura, que debe evolucionar paralelamente con estos. RUP propone dividir el trabajo en mini proyectos que serán vistos como iteraciones. Una iteración puede realizarse por medio de una cascada y pasa por los flujos fundamentales (Requisitos, Análisis, Diseño, Implementación y Pruebas). El proceso se divide en 4 fases, dentro de las cuales se realizan varias iteraciones.

Las cuatro fases del ciclo de vida son:

- Concepción
- Elaboración
- Construcción
- Transición

En la fase de Concepción las iteraciones ponen mayor énfasis en el modelado del negocio y de requisitos. En la Elaboración se desarrolla la línea base de la arquitectura. En la Construcción se lleva a cabo la

construcción del producto mediante una serie de iteraciones. En la Transición se debe garantizar que el producto resultante está listo para ser entregado.

Entre las razones principales para utilizar RUP es válido destacar que el SITPC es un proyecto bastante complejo, que cuenta con un equipo de desarrollo grande dentro del cual hay inestabilidad de los desarrolladores, ya que estos cambian frecuentemente. Por las características del proyecto, el cliente no siempre se encuentra presente y es de gran importancia la extensa documentación que ofrece RUP para el entendimiento del “complejo negocio”. Esto garantiza poder brindarle al cliente una visión clara de lo que se está realizando en cada etapa del desarrollo. Además, RUP posee alto soporte y herramientas integrales, facilitando aplicar con mayor efectividad esta metodología y permitiendo aprovecharla al máximo.
Resulta válido destacar que el SITPC no utiliza todos los artefactos que genera RUP, pues se rige por las normas que se le exigen en el Expediente de Proyecto según el Programa de Mejoras establecido por el Centro Nacional de Calidad de Software (CALISOFT).

1.5. Componentes de software

Según la RAE, "Componente" es un adjetivo para aquello "que compone o entra en la composición de un todo."
El concepto de componentes para el desarrollo de software no es un concepto nuevo. Existen varias definiciones de componentes realizadas por diferentes autores:

- Un componente es una parte no trivial, casi independiente, y reemplazable de un sistema que llena claramente una funcionalidad dentro de un contexto en una arquitectura bien definida. Un componente se conforma y provee la realización física por medio de un conjunto de interfaces (9).
- Un componente de negocio representa la implementación de software del concepto de un negocio “autónomo” o un proceso de negocio. Que consiste de artefactos de software necesarios para expresar, implementar y poner en marcha el concepto de elemento reusable de un sistema más grande de negocios (10).
- Un componente de software es una unidad de composición con interfaces contractualmente especificadas y explícitas sólo con dependencias dentro de un contexto. Un componente de software puede ser desplegado independientemente y es sujeto a la composición de terceros (11).

Estas definiciones no son mutuamente excluyentes, por el contrario, se complementan y construyen el significado de componente. En esencia, un componente es una pieza de código preelaborado que

encapsula alguna funcionalidad expuesta a través de interfaces estándar. Los componentes son los "ingredientes de las aplicaciones", que se juntan y combinan para llevar a cabo una tarea.

Sin embargo más allá de su definición existen algunas características claves para que un elemento pueda ser catalogado como componente:

- **Identificable:** debe tener una identificación que permita acceder fácilmente a sus servicios y que permita su clasificación.
- **Reemplazable:** se puede reemplazar por nuevas versiones u otro componente que lo sustituya y mejore.
- **Bien Documentado**: un componente debe estar correctamente documentado para facilitar su búsqueda si se quiere actualizar, integrar con otros o adaptarlo.
- **Genérico**: sus servicios deben servir para varias aplicaciones.
- **Independiente de la plataforma:** no dependen de un hardware, software o sistema operativo específico.

Todas estas características se pueden apreciar en el componente que informatiza los procesos Prueba Testifical y de Reconocimiento. El mismo es identificable a través de su ruta de acceso y se puede acceder fácilmente a sus servicios. Puede ser reemplazado sin afectar al resto del sistema donde se ejecute e incluso puede ser eliminado. Se cuenta con toda la documentación necesaria en cada uno de los artefactos generados por la metodología RUP en cada una de sus fases. Puede ser utilizado desde cualquiera de los módulos del SITPC y no depende de ningún software o hardware específico pues es un componente web.

El paradigma de ensamblar componentes y escribir código para hacer que estos componentes funcionen se conoce como Desarrollo de Software Basado en Componentes (DSBC).

El DSBC permite reutilizar piezas de código preelaborado que permiten realizar diversas tareas, conllevando a diversos beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo y el mayor retorno sobre la inversión (12).

El uso de este paradigma posee algunas ventajas:

- **Reutilización del software:** permite alcanzar un mayor nivel de reutilización de software.
- **Simplifica las pruebas:** permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- **Simplifica el mantenimiento del sistema:** cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.

- **Mayor calidad:** dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

Generalmente los modelos de desarrollo de componentes existentes en la actualidad tienen sus características peculiares, ya que cada modelo surge debido a la necesidad de satisfacer una línea de producción de software determinada que responde a un grupo de necesidades específicas.

Para la creación del módulo Común del SITPC se desarrollan una serie de componentes que responden a los procesos de negocio que son comunes a todos los módulos del proyecto. Para la construcción del componente que informatice los procesos Prueba Testifical y de Reconocimiento se utilizará el paradigma de la Programación Orientada a Objetos. Dicho componente podrá ser integrado con el resto del SITPC para mejorar la uniformidad en la tramitación y la gestión de la información en los Tribunales Populares Cubanos.

1.6. Arquitectura de software.

Una vez definido que se utilizará el DSBC es necesario conocer la arquitectura. Según David Garlan la Arquitectura de Software establece un puente entre el requerimiento y el código. A su vez el documento de IEEE Standard 1471-2000 define:

“La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución” (13).

1.6.1. Marco de trabajo.

Un marco de trabajo o *framework* es un diseño reutilizable del sistema completo o de alguna de sus partes y se expresa mediante un conjunto de clases abstractas y la forma de interactuar de sus instancias (14).

A continuación se exponen las principales características de los *frameworks* que serán utilizados para el desarrollo de la solución.

1.6.1.1. JQuery 1.8.

JQuery es una biblioteca JavaScript cuyo creador, John Resig, ideó con el fin de simplificar la forma en que se interactuaba con los documentos HTML. Esta permite una serie de características que facilitan el trabajo con este tipo de documentos alguna de las cuales son: la manipulación del árbol del Modelo de Objetos del

Documento (DOM, por sus siglas en inglés), el desarrollo de animaciones, el manejo de eventos y lograr la interacción con las páginas web gracias a la técnica JavaScript Asíncrono y XML (AJAX, por sus siglas en inglés).

Es un software libre y de código abierto, publicado bajo la doble licencia MIT y Licencia Pública General de GNU en su segunda versión, lo que facilita su utilización en todo tipo de proyectos.

1.6.1.2. Symfony 2.1.

Symfony es un proyecto PHP de software libre que permite crear aplicaciones y sitios web rápidos y seguros, de forma profesional, basados en el patrón Modelo Vista Controlador, el cual fue construido con varios componentes independientes creados por el proyecto Symfony. Su código y el de todos los componentes y bibliotecas que incluye, se publican bajo la licencia MIT de software libre. La documentación del proyecto también es libre e incluye varios libros y decenas de tutoriales específicos (15).

Symfony2 es la versión más reciente de Symfony, la cual supone un cambio radical tanto en arquitectura interna como en filosofía de trabajo respecto a sus versiones anteriores. Ha sido ideado para aprovechar todas las nuevas características de PHP 5.3 y por eso es uno de los *frameworks* PHP con mejores rendimiento. Su arquitectura interna está completamente desacoplada, lo que permite reemplazar o eliminar fácilmente aquellas partes que no se acoplan correctamente en un determinado proyecto (16).

1.6.1.3. Twitter Bootstrap 2.1.

Twitter Bootstrap es una colección de herramientas de software libre para la creación de sitios y aplicaciones web el cual es compatible con gran parte de los navegadores web.

Bootstrap 2.1 es un *framework* diseñado para simplificar el proceso de creación de diseños web. Para ello ofrece una serie de plantillas CSS y de ficheros JavaScript, los cuales permiten obtener:

- Interfaces que funcionen de forma perfecta en los navegadores actuales y correctamente en los no tan actuales.
- Un diseño que pueda ser visualizado de forma correcta en distintos dispositivos y a distintas escalas y resoluciones.
- Una mejor integración con las bibliotecas que se suelen usar habitualmente, como por ejemplo jQuery.
- Un diseño sólido basado en herramientas actuales y potentes.

1.6.1.4. Doctrine 2.0.

Doctrine 2.0 es un Asignador Objeto Relacional (ORM, por sus siglas en inglés) para PHP 5.3.0 y versiones posteriores que proporciona persistencia transparente de objetos PHP. Se sitúa en la parte superior de una poderosa Capa de Abstracción de Base de Datos (DBAL, por sus siglas en inglés). La principal tarea de los asignadores objeto relacionales es la traducción transparente entre objetos (PHP) y las filas relacionales de la base de datos.

Una de las características clave de Doctrine es la opción de escribir las consultas de base de datos en una dialecto SQL propio orientado a objetos llamado Lenguaje de Consulta Doctrine (DQL por *Doctrine Query Language*), inspirado en Hibernate HQL. Además, DQL difiere ligeramente de SQL en que abstrae considerablemente la asignación entre las filas de la base de datos y objetos, permitiendo a los desarrolladores escribir poderosas consultas de una manera sencilla y flexible.

Otras características de Doctrine son las siguientes:

- Muy sencillo de configurar.
- Puede generar los modelos a partir de la base de datos.
- Del mismo modo, también puede generar la base de datos a partir de los modelos.

1.6.1.5. Twig 1.9.0

Twig es un flexible, rápido y seguro motor de plantillas para PHP. Cuenta con un ambiente amigable para los diseñadores y desarrolladores y añade funcionalidades útiles a los entornos de plantillas.

La parte más poderosa de Twig es la herencia entre plantillas, que permite crear un “esqueleto” de plantilla base que contenga todos los elementos comunes de un sitio y define los bloques que las plantillas descendientes pueden sustituir.

Sus características claves son:

Rápido: Twig compila las plantillas hasta código PHP regular optimizado. El costo general en comparación con código PHP regular se ha reducido al mínimo.

Seguro: Twig tiene un modo de recinto de seguridad para evaluar el código de plantilla que no es confiable. Esto permite utilizar Twig como un lenguaje de plantillas para aplicaciones donde los usuarios pueden modificar el diseño de la plantilla.

Flexible: Twig es alimentado por flexibles analizadores léxico y sintáctico. Esto permite al desarrollador definir sus propias etiquetas y filtros personalizados (17).

1.6.2. Lenguajes de programación.

La elección de los marcos de trabajo expuestos previamente deriva en la utilización de los diversos lenguajes de programación que proponen los mismos. Dentro de ellos los más relevantes son Pre-procesador de Hipertexto PHP (PHP), Hoja de estilo en Cascada (CSS), JavaScript y Lenguaje de Marcado de Hipertexto (HTML).

PHP 5.3

PHP es un acrónimo recursivo del inglés *PHP Hypertext Pre-processor*, que significa Pre-procesador de Hipertexto PHP. Es un lenguaje de programación interpretado. Su código se ejecuta del lado del servidor y se utiliza para la creación de páginas web dinámicas. PHP permite acceder a la información almacenada en las bases de datos y es compatible con la mayoría de los Sistemas Gestores de Bases de Datos utilizados en la actualidad tales como MySQL, PostgreSQL, Oracle, ODBC, DB2, Microsoft SQL Server, Firebird y SQLite. Además puede ser utilizado desde la mayor parte de los sistemas operativos, entre ellos Linux, Mac OS X y Microsoft Windows. La programación PHP es segura y confiable ya que, entre otras características, el código no se muestra en el navegador sino que el servidor lo ejecuta y manda solamente el resultado HTML al navegador web. PHP es libre lo que presenta una alternativa de fácil acceso para todos, además cuenta con gran documentación en su sitio web oficial. Permite al desarrollador aplicar técnicas de programación orientada a objetos, así como cualquier otra que este desee para estructurar y organizar su trabajo, ejemplo de ello son las aplicaciones que utilizan el patrón Modelo-Vista-Controlador que permite separar el acceso a datos, el control y la interfaz de usuario. De todas las novedades introducidas por PHP 5.3 las más relevantes para los programadores de Symfony2 son las funciones anónimas y los *namespaces*.

CSS 3

CSS viene del inglés *Cascading Style Sheets*, que significa hoja de estilo en cascada. Es un lenguaje usado para definir la presentación de un documento estructurado escrito en XML o HTML. Su objetivo principal es separar la presentación del código de la aplicación lo que permite que se pueda modificar la visualización del documento sin alterar el contenido del mismo. Al utilizar CSS se optimiza el ancho de banda de la aplicación ya que un mismo estilo puede ser aplicado varias veces sin necesidad de volver a definirlo.

HTML 5

HTML viene del inglés *HyperText Markup Language*, que significa Lenguaje de Marcado de Hipertexto, es el lenguaje predominante para la elaboración de páginas web. Sus componentes vitales son las etiquetas, los atributos y los tipos de datos que estos admiten. HTML5 es la quinta versión de este lenguaje. El mismo incorpora una serie de etiquetas y atributos que son más afines con los sitios web modernos, tales como funcionalidades para mejorar las etiquetas de audio y video, así como renderizar elementos 3D. Implementa mejoras en los formularios, añade nuevos tipos de datos y facilidades para validar el contenido sin utilizar *javascript*. Añade etiquetas para manejar la web semántica (web 3.0) e incluye nuevas Interfaces de Programación de Aplicaciones (API) para realizar tareas como arrastrar, soltar y trabajar sin conexión.

JavaScript 3

JavaScript es un lenguaje interpretado que permite incluir macros en páginas web. Estas macros se ejecutan del lado del cliente y no en el servidor. La característica de JavaScript que más simplifica la programación es que, aunque el lenguaje soporta varios tipos de datos, no es necesario declarar el tipo de las variables, argumentos de funciones ni valores de retorno de las funciones. El tipo de las variables cambia implícitamente cuando es necesario, lo que ayuda a programar con rapidez macros sencillas.

1.6.3. Herramientas de desarrollo.

Mozilla Firefox 24.0

Es un navegador web libre y de código abierto desarrollado para los sistemas operativos Windows, Mac y GNU/Linux, coordinado por la Corporación Mozilla y la Fundación Mozilla, siendo este su producto estrella. Entre sus principales características destacan la tradicional navegación por pestañas, corrector ortográfico, búsqueda progresiva, marcadores dinámicos, soporte de estándares como HTML5 y CSS3, un administrador de descargas, lector RSS, navegación privada e integración del motor de búsqueda que desee el usuario. Se pueden añadir funciones a través de complementos, también llamados extensiones, desarrollados por la propia Mozilla o por terceros aficionados y comerciales.

Como ventaja para los desarrolladores web, posee un repertorio de herramientas incorporadas, como la Consola de errores, Scratchpad (para probar código JavaScript), editor HTML, el Inspector DOM, o la inclusión de extensiones como Firebug.

NetBeans 7.4

Es un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés), disponible para los sistemas operativos Windows, Mac, Linux y Solaris. El proyecto NetBeans consta de un IDE de código abierto y una plataforma de aplicaciones que permiten a los desarrolladores crear rápidamente aplicaciones móviles utilizando la plataforma Java, así como JavaFX, PHP, JavaScript y Ajax, Ruby y Ruby on Rails, Groovy y Grails y C/C++, web, empresa y escritorio. Una de sus mejores ventajas es que es un producto libre y gratuito sin restricciones de uso. Netbeans permite la reutilización de módulos así como el control de versiones a través de la integración con Subversion. Permite también crear aplicaciones web con PHP 5 y además viene con soporte para el marco de trabajo Symfony2.

PostgreSQL 9.2

Es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo la licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones está muy igualado a otras bases de datos comerciales. Utiliza un modelo cliente/servidor y usa multiprocesos para garantizar la estabilidad del sistema ya que un fallo en uno de los procesos no afecta el resto y el sistema continúa con su funcionamiento. Funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema.

Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada apoyados por organizaciones comerciales. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

Visual Paradigm 8.0

Es una herramienta de Ingeniería de Software Asistida por Computación (CASE por sus siglas en inglés), la cual propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas. Permite también la generación de código inverso, código desde diagramas y documentación.

La importancia de esta herramienta radica en que permite aumentar la calidad del software a través de la mejora de la productividad en el desarrollo y mantenimiento del mismo. También permite la reutilización del software, portabilidad y estandarización de la documentación, además del uso de las distintas metodologías

propias de la Ingeniería de Software. Visual Paradigm será utilizado para la creación de los modelos y diagramas necesarios en el diseño de la solución.

Subversion 1.5

Es un sistema centralizado de control de versiones. En otras palabras, es un sistema que permite llevar el control de los cambios realizados por una o más personas a una serie de archivos a través del tiempo. Un desarrollador solamente tiene que encargarse de guardar sus cambios al proyecto y el sistema se ocupa de integrarlos con los cambios de los otros miembros del equipo y de asegurarse de que no existan conflictos. Es una herramienta de software libre bajo una licencia de tipo Apache/BSD.

Apache 2.2

Apache es un servidor web HTTP, es un proyecto de código abierto y uso gratuito, multiplataforma, muy robusto y que destaca por su seguridad y rendimiento. El servidor Apache se desarrolla dentro del proyecto HTTP Server (httpd) de la Apache Software Foundation. Desde 1996, Apache, es el servidor HTTP más usado. Un servidor web es el encargado de aceptar las peticiones de páginas (o recursos en general) que provienen de los visitantes que acceden al sitio web y gestionar su entrega o denegación, de acuerdo a las políticas de seguridad establecidas.

1.7. Conclusiones parciales.

- Se evidenció la necesidad de la creación de un componente que informatice los procesos Prueba Testifical y de Reconocimiento debido a la falta de sistemas que gestionen los mismos en Cuba y el mundo.
- Se concretó como metodología de desarrollo RUP para asegurar la producción de software de alta calidad que satisfaga los requerimientos de los usuarios finales.
- Se determinó el uso de las herramientas Mozilla Firefox 24.0, NetBeans 7.4, PostgreSQL 9.2, Subversion 1.5, y el servidor Apache 2.2 lo que permitirá la creación de un componente de software libre, multiplataforma y de fácil actualización.
- Se preparó el entorno de desarrollo para realizar el diseño e implementación de los procesos Prueba Testifical y de Reconocimiento que contribuya a mejorar la uniformidad en la tramitación y la gestión de la información en el Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos.

Capítulo 2: Propuesta de solución

2.1 Introducción

El presente capítulo tiene como objetivo describir la solución técnica de la investigación. Se analiza la arquitectura del sistema y cada uno de sus componentes. Se describen las pruebas testificales y de reconocimiento, así como los requisitos no funcionales y se analizan los patrones de diseño utilizados en la solución propuesta. Se presentan además los artefactos generados en los modelos de diseño, implementación y despliegue.

2.2 Arquitectura del sistema.

El desarrollo del SITPC responde a una arquitectura en capas basada en el patrón arquitectónico Modelo-Vista-Controlador. Entre las principales ventajas de esta arquitectura se encuentra el aislamiento de la lógica de aplicaciones en componentes independientes que son susceptibles de reutilizarse después en otros sistemas (18).

Cada capa contiene solamente las funcionalidades relacionadas con sus tareas, esto permite que en caso de ser necesario realizar algún cambio, el mismo se realice solo en la capa correspondiente sin afectar al resto del sistema.

Dentro de la arquitectura en capas se utilizó el patrón arquitectónico Modelo-Vista-Controlador (MVC) con los componentes que propone el marco de trabajo Symfony2. Este patrón separa conceptualmente la representación visual de la aplicación, las acciones que intercambian datos y el modelo de negocio y su dominio por lo que se consigue un mantenimiento más sencillo de las aplicaciones.

El marco de trabajo Symfony2 propone una estructura de paquetes que deja bien definido en qué lugar se encuentran las clases correspondientes a la vista, al modelo y al controlador. Esta estructura se muestra en la imagen a continuación.

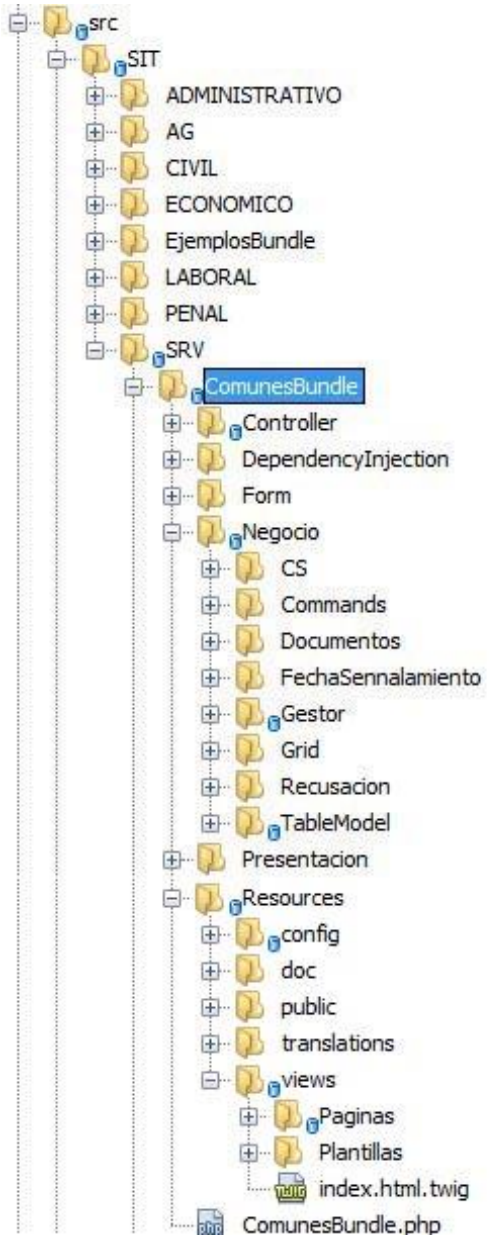


IMAGEN 1 ESTRUCTURA PROPUESTA POR SYMFONY2 EN EL SITPC

En la imagen 2 se muestra la arquitectura del SITPC y la ubicación de los componentes en cada capa. Todos los módulos se estructuran arquitectónicamente igual. La arquitectura cuenta además con una serie de complementos transversales a las capas que son los encargados de garantizar seguridad, mensajería, configuración y tratamiento de excepciones, entre otros aspectos. Las entidades del dominio son accesibles desde el modelo y el controlador.

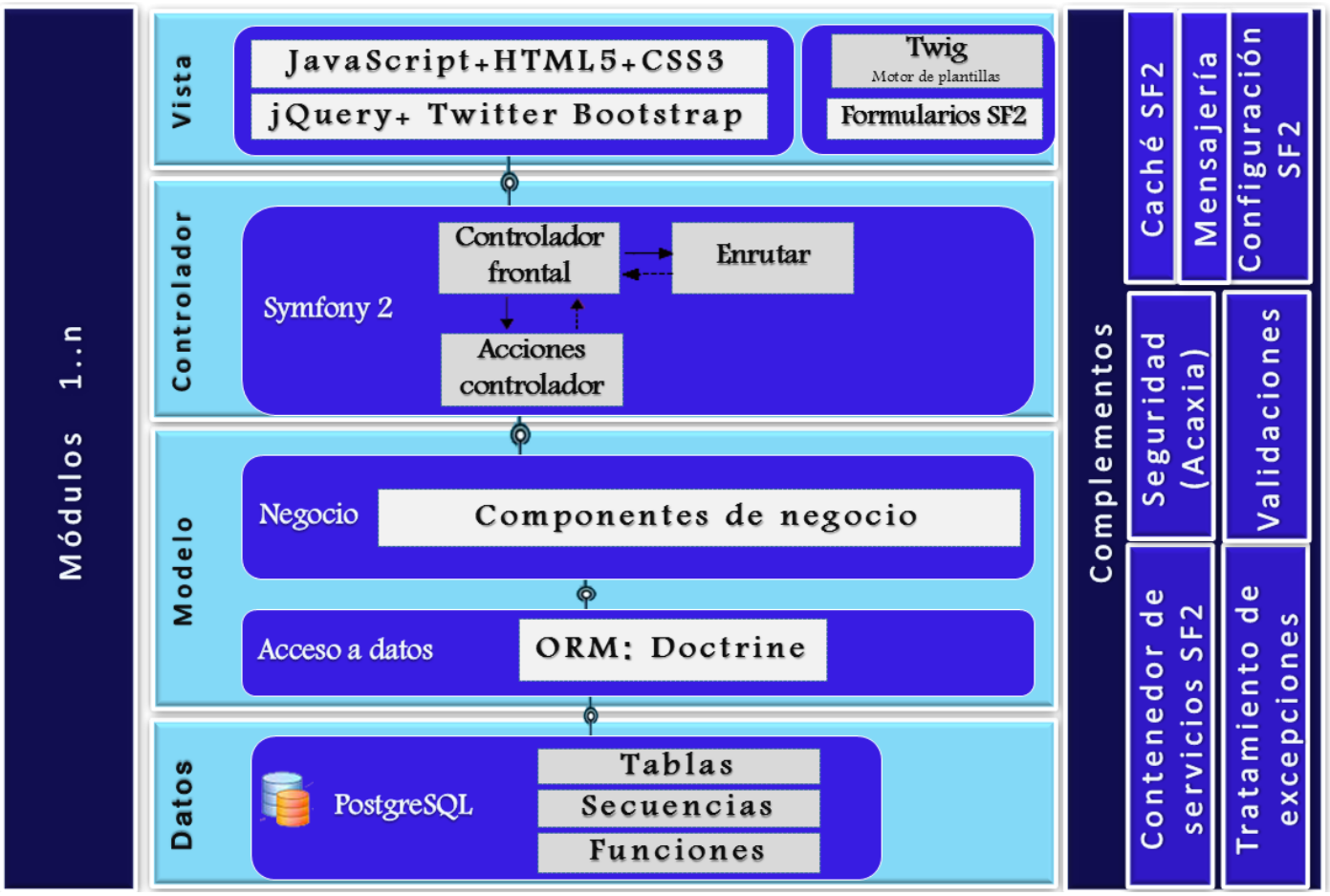


IMAGEN 2 ARQUITECTURA DEL SISTEMA

Vista

Se encarga de gestionar los datos de entrada y salida mediante las interfaces de usuario, esta capa contiene los componentes o páginas con los que el usuario interactúa. En ella se visualizan los datos procesados mediante el navegador web utilizando tecnologías como JavaScript, HTML5, CSS3, JQuery y Twitter Bootstrap, así como las plantillas generadas por el motor de plantillas Twig. En la vista se encuentran también los formularios de Symfony2 que se utilizan para recoger o mostrar datos al usuario.

En la estructura propuesta por Symfony2 en el SITPC (Imagen 1) las vistas se encuentran en el directorio src\SIT\SRV\ComunesBundle\Resources\views y los formularios se encuentran en el directorio src\SIT\SRV\ComunesBundle\Form.

Controlador

En esta capa se encuentra el controlador frontal, el mismo se comunica con el enrutador para obtener la ruta de acceso necesaria y comprueba que se tienen permisos de acceso a esta ruta para luego devolver el controlador correspondiente a la tarea que desea ejecutar, este último se comunica con el modelo para devolver la información en las páginas de la vista. En esta capa se encuentran además todos los controladores por caso de uso. En la estructura propuesta por Symfony2 en el SITPC (Imagen 1) las clases controladoras se encuentran en el directorio src\SIT\SRV\ComunesBundle\Controller.

Modelo

Esta capa está compuesta por las subcapas Negocio y Acceso a datos. La subcapa de Negocio contiene las clases gestoras encargadas del manejo de la lógica del negocio. Estas clases desacoplan los componentes del negocio de las clases controladoras. En la estructura propuesta por Symfony2 en el SITPC (Imagen 1) las clases gestoras se encuentran en el directorio src\SIT\SRV\ComunesBundle\Negocio\Gestor. La subcapa de Acceso a datos gestiona las peticiones de la capa de negocio consultando la base de datos y retornando los datos que se recuperan al gestor correspondiente. Esta capa contiene los repositorios mapeados por Doctrine para la comunicación con el servidor de datos. En la estructura propuesta por Symfony2 en el SITPC (Imagen 1) los repositorios se encuentran en el directorio vendor\Base\ComunBundle\Repository.

Datos

En esta capa se encuentra el gestor de base de datos PostgreSQL y en este un conjunto de tablas, secuencias y funciones que permiten persistir la información con la que trabaja la aplicación y gestionar su almacenamiento.

Módulos

Los módulos o *bundles* son directorios que contienen todo tipo de archivos dentro una estructura jerarquizada de directorios. Cada módulo se estructura según la arquitectura descrita y permite utilizar funcionalidades construidas por terceros o empaquetar funcionalidades para distribuirlas y reutilizarlas.

Complementos

Los complementos son un conjunto de facilidades y mejoras que permiten que la arquitectura sea más flexible y adaptable. Entre estas mejoras se encuentran el contenedor de servicios de Symfony2, la gestión de seguridad, tratamiento de excepciones, caché, validaciones, configuración y mensajería, entre otras.

2.3 Descripción de las Pruebas de Reconocimiento y Testifical

Los requisitos funcionales de los procesos Prueba Testifical y de Reconocimiento, se resumen en 18 casos de uso. En la metodología de desarrollo RUP, los casos de uso no son sólo una herramienta para especificar los requisitos del sistema, también guían su diseño, implementación y prueba. Los casos de uso constituyen un elemento integrador y una guía del trabajo (19). Las especificaciones de los casos de uso se encuentran en el documento CEGEL-SITPC-114_ECU_COM del expediente del proyecto. La siguiente tabla muestra un resumen de cada uno de estos casos de uso.

Caso de uso	Descripción	Complejidad
Crear acta de prueba testifical	Permite a la secretaria registrar los datos para crear el acta de prueba testifical.	Alta
Gestionar repreguntas en el acta de prueba testifical	Permite a la secretaria registrar los datos para crear el acta de prueba testifical.	Alta
Registrar declaración de testigo	Permite a la secretaria registrar los datos para crear el acta de prueba testifical.	Alta
Registrar tacha de testigo en el acto	Permite a la secretaria registrar tacha de testigo en el acto.	Media
Registrar escrito de tacha de testigo	Permite al abogado registrar escrito de tacha de testigo.	Alta
Disponer sobre escrito de tacha	El caso de uso se inicia cuando el Juez necesita disponer de un escrito de tacha, admitiéndolo o rechazándolo.	Media
Registrar escrito de impugnación de tacha de testigo	Permite al abogado registrar escrito de impugnación de tacha de testigo.	Media
Disponer sobre la impugnación de tacha	El caso de uso se inicia cuando el Juez necesita disponer de un escrito de impugnación de tacha, admitiéndolo o rechazándolo.	Media
Registrar escrito de repreguntas a testigo(s)	Permite al abogado registrar escrito de repreguntas para testigos.	Media

Dictar providencia a las repreguntas antes de la práctica	Permite al juez dictar providencia a las repreguntas antes de la práctica.	Baja
Disponer sobre preguntas antes de prueba testifical	El caso de uso se inicia cuando el Juez necesita declarar pertinencia de las preguntas del (de los) pliego(s) registrados en la solicitud de la prueba o antes de la prueba testifical.	Media
Crear acta de tacha de testigo	Permite a la secretaria registrar los datos para crear el acta de prueba de tacha de testigo.	Media
Gestionar pliego para testigo	Comienza cuando el abogado necesita gestionar pliego de preguntas para testigo en un escrito de proposición de pruebas. Con la posibilidad de adicionarlo modificarlo o eliminarlo del escrito.	Media
Gestionar testigo	Comienza cuando el abogado necesita gestionar un testigo en una prueba testifical. Con la posibilidad de adicionarlo modificarlo o eliminarlo del escrito.	Media
Disponer sobre prueba testifical	El caso de uso se inicia cuando el Juez necesita disponer sobre prueba testifical, admitiendo o no el testigo y/o declarando pertinencia de las preguntas y la fecha y hora de la práctica.	Alta
Gestionar prueba de reconocimiento	Comienza cuando el abogado necesita gestionar una prueba de reconocimiento en un escrito de proposición de pruebas. Con la posibilidad de adicionarla modificarla o eliminarla del escrito.	Alta
Señalar para prueba de reconocimiento	El caso de uso se inicia cuando el Juez necesita señalar la fecha y hora para realizar el reconocimiento solicitado.	Media
Crear acta de prueba de reconocimiento	Permite a la secretaria registrar los datos para crear el acta de prueba de reconocimiento.	Media

TABLA 2 RESUMEN DE LOS CASOS DE USO

2.4 Requisitos no funcionales

Los requisitos no funcionales del SITPC se encuentran especificados en el documento CEGEL-SITPC-0113-Requisitos No Funcionales-v1.3. En dicho documento se encuentran los requisitos no funcionales agrupados por tipos, encontrándose los de usabilidad, confiabilidad, eficiencia, soporte, restricciones de diseño, interfaz y seguridad.

Durante el diseño e implementación del componente para informatizar los procesos Prueba Testifical y de Reconocimiento se tuvieron en cuenta estos requisitos no funcionales (RnF), lo cual se pone de manifiesto en la solución obtenida. A continuación se muestran algunos de ellos.

RnF.05 Requisito de Usabilidad 5

En el sistema se deben visualizar todos los mensajes en idioma español. La tipografía debe ser uniforme, de un tamaño adecuado. Debe tener un contraste adecuado que resalte los textos.

Todos los mensajes de la aplicación se muestran en idioma español, así como las advertencias. Los textos son de color azul oscuro con fondo blanco lo que proporciona el contraste adecuado para una correcta visualización de los mismos.

RnF.19 Requisito de Soporte 1

Para garantizar un sistema con capacidad de mantenimiento, reparable y escalable se desarrollará orientado a componentes, es decir por bloques de construcción que conformarán las partes del sistema. Dichos componentes están representados por módulos y capas de abstracción que garanticen un código cohesionado con las responsabilidades bien delimitadas.

El sistema fue diseñado haciendo uso de la arquitectura en capas y se encuentra dividido en subsistemas. El componente desarrollado pertenece al módulo Común que se encuentra en el directorio src\SIT\SRV\ComunesBundle.

RnF.22 Requisito de Restricción del diseño 2

Se utilizará la herramienta CASE Visual Paradigm teniendo en cuenta sus ventajas para modelar los diferentes artefactos que se obtienen en los flujos de trabajo y sus diferentes fases. Las restricciones propias del diseño radican en las pautas que se establecerán, así como las diferentes relaciones que se formen durante el modelado del sistema.

Para realizar los artefactos de los flujos de trabajo se utilizó la herramienta Visual Paradigm en su versión 8.0. Con esta se realizaron los diagramas de clases del diseño y los diagramas de secuencias pertenecientes al modelo de diseño, el diagrama de componentes perteneciente al modelo de implementación y el diagrama de despliegue perteneciente al modelo de despliegue.

RnF.37 Requisito de Seguridad 8

En el sistema se deben emplear el protocolo https (Protocolo seguro de transferencia de hipertexto) para la transmisión de las credenciales de autenticación entre los clientes y el servidor.

El sistema utiliza el protocolo https para la transmisión de las credenciales de autenticación entre los clientes y el servidor. Esto se evidencia en el diagrama de despliegue para la comunicación entre los clientes y el Servidor de aplicaciones de cada una de las instancias de los Tribunales.

2.5 Patrones de diseño

Un patrón de diseño provee un esquema para refinar componentes de un sistema de software y la forma en que se relacionan entre sí. Describe una estructura generalmente recurrente de comunicación de componentes que resuelve un problema de diseño general dentro de un contexto particular (14).

A continuación se explican los patrones utilizados en la solución propuesta.

Inyección de dependencias

La inyección de dependencias junto al contenedor de servicios es lo que hace que Symfony2 sea tan rápido y flexible.

La inyección de dependencias consiste en pasar (inyectar) a las clases todos los objetos que necesitan (dependencias) ya creados y configurados. Estas dependencias se pueden inyectar mediante el constructor (“constructor injection”), que es el caso más común, mediante los métodos setter de la clase (“setter injection”), y directamente a través de las propiedades de las clases (“property injection”). Sin embargo, tiene la desventaja de que se debe recordar siempre antes de utilizar una clase, crear y configurar correctamente todas sus dependencias. La solución a este problema es el llamado contenedor de inyección de dependencias, que es un objeto que sabe cómo crear los objetos de una aplicación, evitando manejar todas las dependencias a mano. Para ello, conoce todas las relaciones entre las clases y las configuraciones necesarias para instanciar correctamente cada clase.

La utilización de un contenedor simplifica el uso de la inyección de dependencias en una aplicación, pero crear uno con cientos de dependencias es una tarea demasiado complicada por lo que Symfony2 incluye un completo contenedor de inyección de dependencias listo para usar.

Es en este ámbito que aparece el término **servicio**, el cual hace referencia a cualquier clase/objeto manejada por el contenedor de inyección de dependencias. En la práctica los servicios son clases PHP que realizan cualquier tarea global dentro de la aplicación.

Una ventaja añadida de la definición de servicios es que no penaliza el rendimiento de la aplicación, o sea, si el código no hace uso de algún servicio, el contenedor nunca lo crea, por lo que se puede definir tantos servicios como se requieran sin preocuparse por el rendimiento de la aplicación.

En aplicaciones complejas como el SITPC, es habitual que existan dependencias entre unos servicios y otros, por lo que para definir estas dependencias se inyectan a través del constructor mediante la opción *arguments* en la declaración de los servicios en los ficheros *services.yml*, que están ubicados en los diferentes módulos. En estos casos al valor del argumento se le agrega el prefijo @ y de esta forma el contenedor lo interpreta como el nombre de un servicio.

Como los servicios de Symfony2 se obtienen a través del contenedor de inyección de dependencias, resulta esencial tener acceso al contenedor desde cualquier punto de la aplicación. Por lo tanto, se inyecta automáticamente el contenedor a todos los controladores que heredan de la clase Controller y permiten el acceso al mismo a través de $this->container.

En la siguiente imagen se puede apreciar su uso en la clase controladora ReconocimientoJudicialController a la cual se asocia una clase gestor (ReconocimientoJudicialGtr), para realizar las acciones de acceso a datos, a través de un servicio que se obtiene del contendor gracias a:

$this->container->get('comunes.reconocimientojudicialgtr')

```
class ReconocimientoJudicialController extends BaseController {

    private function getGestor() {
        if (!$this->container->has('comunes.reconocimientojudicialgtr')) {
            throw new \LogicException('Este servicio no esta registrado en la aplicacion');
        }
        return $this->container->get('comunes.reconocimientojudicialgtr');
    }
```

IMAGEN 3 USO DEL CONTENEDOR DE DEPENDENCIAS

En este caso el contenedor de Symfony2 busca entre todos los servicios definidos aquel que se llame comunes.reconocimientojuidicialgtr y realiza todas las tareas necesarias para instanciar el objeto correspondiente y entregarlo.

Este servicio se encuentra publicado en "SIT/src/SIT/SRV/ComunesBundle/Resources/config/services.yml". Este fichero tiene dos secciones: *parameters*, donde se declaran los parámetros de configuración, y *services*, donde se declaran los servicios.

```
parameters:

    comunes.reconocimientojudicialgtr.class: SIT\SRV\ComunesBundle\Negocio\Gestor\ReconocimientoJudicialGtr

services:

    comunes.reconocimientojudicialgtr:
        class: %comunes.reconocimientojudicialgtr.class%
        parent: arquitectura.basegtr
```

IMAGEN 4 FRAGMENTO DEL FICHERO SERVICES.YML

Lo que ocurre en este caso es que el servicio llamado también obliga que se ejecute un servicio padre llamado arquitectura.basegtr, el cual se encuentra en "SIT/vendor/Base/ArquitecturaBundle/Resources/config/services.yml".

```
parameters:

    arquitectura.base.gtr.class: Base\ArquitecturaBundle\Negocio\BaseGtr

services:

    arquitectura.basegtr:
        class: %arquitectura.base.gtr.class%
        arguments: [@service_container]
        abstract: true
```

IMAGEN 5 FRAGMENTO DEL FICHERO SERVICES.YML

A este servicio padre se le pasa como argumento el nombre de un servicio (service_container, que representa al propio contenedor de servicios y por tanto, da acceso a cualquier otro servicio) con el prefijo @, que como se explicó anteriormente, es la manera de definir las dependencias entre servicios. De esta forma cuando se hace uso del servicio arquitectura.basegtr, el contenedor creará primero el servicio service_container y se lo pasará al constructor de la clase BaseGtr.

Lo anterior permite que en BaseGtr se pueda inyectar el propio contenedor de dependencias mediante el constructor, y de esta forma se puede acceder al contenedor desde este gestor y desde el gestor al que se le hizo la llamada a su servicio en primera instancia, en este caso, ReconocimientoJudicialGtr, que hereda de BaseGtr.

```php
abstract class BaseGtr
{

    /**
     * @var EntityManager
     */
    private $em;

    protected $container;

    function __construct(Container $container)
    {
        $this->em = $container->get('doctrine.orm.entity_manager');
        $this->container = $container;
    }
```

IMAGEN 6 FRAGMENTO DE LA CLASE BASEGTR

Una vez que ocurre todo esto, ya se puede utilizar cualquier servicio del contenedor de inyección de dependencias desde la clase gestora de la misma forma como se hacía en la controladora, con el $this->container, y a su vez, la controladora puede acceder a los métodos de la clase gestora.

Otro ejemplo de la utilización de este patrón es el uso del método getDoctrine() el cual es un atajo disponible en todos los controladores que extienden de la clase Controller de Symfony2, que en realidad es equivalente a $this->get('doctrine'), con el que se llama al servicio llamado “doctrine”.

Experto

Este patrón es el que más se utiliza para la asignación de responsabilidades, expresa la característica de que los objetos hacen cosas relacionadas con la información que poseen, por lo que se logra conservar el encapsulamiento, dando soporte a un bajo acoplamiento que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento (18).

En el componente propuesto se utiliza sobre todo en las clases controladoras cuando se renderizan las vistas, como en el ejemplo de ReconocimientoJudicialController, así como también en las clases gestoras que se encargan de la gestión de la información necesaria para las entidades definidas.

```
public function indexAction()
{
    $form = $this->createForm(new PruebaReconocerPersonaType());
    $modelo = $this->container->get('comunes.tm.reconocimientojudicial');
    $rutas = new RutasGrid();
    $rutas->setRutaActualizar('');
    $rutas->setRutaDatosAjax('comunes_pruebas_reconocimiento');
    return $this->render('ComunesBundle:Paginas\Pruebas\PruebasReconocimiento:PruebasReconocimiento
        'modelo' => $modelo,
        'ruta' => $rutas,
        'frm' => $form->createView()
    ));
}
```

IMAGEN 7 EJEMPLO DEL PATRÓN EXPERTO EN LA CLASE RECONOCIMIENTOJUDICIALCONTROLLER

Creador

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos (18). Su utilización en la solución propuesta se evidencia en los gestores responsables de la creación de los objetos de las clases que representan las entidades. También se utiliza en las clases controladoras para la creación de los formularios.

Un ejemplo de lo anterior se puede apreciar en la clase ReconocimientoJudicialGtr cuando se crea un objeto de la clase Litigante.

```
if ($idEncargado != null) {
    $prueba->setPracticaFueraSede(true);
    $personaencargada = new Litigante();
    $personae = $this->getEm()->getRepository('ComunBundle:AG\Persona')->find($idEncargado);
    $personaencargada->setPersona($personae);
    $contactoe = $personae->getContacto()->getCopia();
```

IMAGEN 8 CREACIÓN DE UN OBJETO DE LA CLASE LITIGANTE

Bajo acoplamiento

El bajo acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad (18). Esto lo logra al hacer que las distintas partes del software funcionen sin que dependan demasiado unas de otras. Se puede apreciar el uso de este patrón en las entidades que son las clases más reutilizadas y sin embargo no presentan ninguna asociación ni con la vista ni con el controlador.

Alta cohesión

Se logra una alta cohesión cuando los elementos de una clase colaboran para producir algún comportamiento bien definido. Una clase con alta cohesión colabora con otros objetos para compartir el esfuerzo si la tarea es grande (18).

En la solución propuesta se aprecia su uso en la asignación de responsabilidades a las clases según les corresponda, estableciendo las condiciones necesarias para que estas colaboren entre sí en aquellas tareas que las impliquen y que no son capaces de resolver por sí solas. Por ejemplo, los casos de uso cuentan con una clase controladora y una clase gestora, que dividen sus responsabilidades para no sobrecargar a ninguna con tareas excesivas. Para el caso de uso Gestionar prueba de reconocimiento la clase controladora es ReconocimientoJudicialController, a la que se asocia el gestor ReconocimientoJudicialGtr.

Controlador

Asigna la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase controladora que sirve como intermediaria entre la interfaz y el algoritmo que la implementa. Este patrón sugiere que la capa de negocio debe estar separada de la capa de presentación. Se evidencia su uso en las clases controladoras y en el controlador frontal de Symfony2. Las peticiones realizadas por el usuario son manejadas por el controlador frontal, único punto de entrada a la aplicación en un entorno determinado, el cual se localiza en el directorio “SIT/web/app.php”. Además en todas las clases controladoras está presente y sirve de intermediario entre las interfaces y los algoritmos que las implementan. Las clases controladoras para el módulo Común se encuentran en el directorio “SIT/src/SIT/SRV/ComunesBundle/Controller”.

Decorador

Pertenece al grupo de los patrones estructurales. Añade responsabilidades adicionales a un objeto dinámicamente. Provee una alternativa flexible a las subclases para extender las funcionalidades, permitiendo no tener que crear sucesivas clases que hereden de otra solo para incrementar una funcionalidad, sino otra clase que implementa dicha funcionalidad y se relaciona con la primera.

Este patrón se utiliza de manera implícita en el marco de trabajo Symfony2 con el motor de plantillas Twig que proporciona la herencia entre plantillas, la cual permite la creación de una plantilla base plantilla_SIT.html.twig que contiene los elementos comunes del sitio web y define los bloques que luego en las plantillas que hereden de esta se pueden redefinir. Las plantillas usadas en cada caso de uso heredan de la plantilla del subsistema al que pertenecen y redefinen el bloque de contenido. Esto proporciona una forma flexible de introducir o eliminar funcionalidades a las páginas de la aplicación.

En la siguiente imagen se aprecia una parte de Plantilla_ComunesOriginal.html.twig que hereda de plantilla_SIT.html.twig y a la vez redefine los bloques título, cabecera y contenido, que también pueden redefinirse en las plantillas que hereden de esta.

```
{% extends 'ComunBundle::plantilla_SIT.html.twig' %}

{%block titulo%}Común{%endblock%}
{% block cabecera %}
    {#CABECERA DE LA PAGINA, CAMBIA SEGUN LA MATERIA#}
<h3>Común</h3>
{% endblock %}

{%block contenido%}
<h1>Contenido por defecto CAMBIAME!!!</h1>
{%endblock%}
```

IMAGEN 9 APLICACIÓN DEL PATRÓN DECORADOR

Método de fabricación

Pertenece al grupo de los patrones creacionales. Define una interfaz para la creación de un objeto, pero deja a las subclases definir cuál clase instanciar (20). Utiliza una clase abstracta con varios métodos definidos y otros abstractos. Para los casos en que el contenedor de servicios de Symfony2 no proporcione todo lo necesario para construir un objeto, se puede utilizar una factoría para crear el objeto y decirle al contenedor que llame a un método en la factoría y no crear directamente una instancia del objeto.

En el caso de la solución propuesta la factoría es la clase EntityManager.php de Doctrine y el método al que se llama es el getRepository($entityName). De esta forma se está proporcionando una instancia de la clase Repository específica que se necesite, que permite hacer uso de los métodos de la misma.

Para hacer uso de este patrón se debe publicar el servicio que dé acceso al método de la factoría. En “SIT/vendor/Base/ComunBundle/ Resources/config/services.yml” están publicados este tipo de servicios que pueden ser utilizados desde cualquier bundle. En la siguiente imagen se muestra un ejemplo de la declaración de un servicio usando el método de fabricación.

```
comun.servicios.PersonaNatural:
    class: Base\ComunBundle\PersonaNaturalRepository
    factory_service: doctrine.orm.default_entity_manager
    factory_method: getRepository
    arguments: [Base\ComunBundle\Entity\AG\PersonaNatural]
```

IMAGEN 10 DECLARACIÓN DE UN SERVICIO USANDO EL MÉTODO DE FABRICACIÓN

De esta manera a cada clase entidad se le asocia una clase repositorio que se encarga de las operaciones de búsqueda y filtrado relacionadas con la base de datos, cuyos métodos pueden ser accedidos haciendo uso de este patrón que proporciona una instancia de la clase repositorio necesaria.

Cuando es necesario hacer uso de alguna consulta desde alguna clase gestora se llama al servicio del Repository donde se encuentra la consulta y se accede a la misma como se evidencia en el ejemplo de la imagen siguiente.

```
/**
 *
 * @param Integer $idPersonaNatural
 * @return Base\ComunBundle\Entity\AG\PersonaNatural
 */
public function obtenerPersonaNatural($idPersonaNatural) {
    return $this->get('comun.servicios.PersonaNatural')->find($idPersonaNatural);
}
```

IMAGEN 11 USO DE UN REPOSITORIO EN LA CLASE GESTORA

Cadena de Responsabilidades

Este patrón evita acoplar al emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición. Crea una cadena con los objetos receptores y pasa la petición a través de la cadena hasta que esta sea tratada por algún objeto.

La petición debe ser manejada por los receptores, lo cual quiere decir que esta petición queda al margen del uso exclusivo. Las peticiones serán filtradas por todos los receptores a medida que se van generando los resultados esperados.

Generalmente se utiliza una clase “Manejador” que constituye el nivel de la cadena donde se recibe la petición en primer lugar, y si esta no puede manejarla se redirige a otras clases “Manejadores Concretos” hasta que encuentre una o varias que puedan manejarla.

En la solución propuesta este patrón está presente cuando se va a escoger una fecha de señalamiento. En este caso la clase “Manejador” es Operador y las clases “Manejadores Concretos” son OperadorAbogadosJueces, OperadorDiaHabil, OperadorSala y OperadorTermino. Todas estas clases se encuentran en “SIT/src/SIT/SRV/ComunesBundle/Negocio/FechaSennalamiento”. Esta cadena analiza que la fecha de señalamiento que se escoja cumpla con todas las condiciones necesarias para que sea válida. Se comprueba con todos los manejadores concretos si se cumplen sus condiciones específicas y en caso contrario se adiciona un error a la lista de errores de la fecha de señalamiento. Finalmente, si la lista de

errores está vacía se asume como válida la fecha escogida, si no se imprimen los errores correspondientes y se toma la fecha como no válida.

2.6 Modelo de diseño

Un modelo es una abstracción del sistema, especificando el sistema modelado desde cierto punto de vista y en un determinado nivel de abstracción (21).

El modelo de diseño funciona como esquema para la implementación. Define clasificadores (clases, subsistemas e interfaces), relaciones entre estos clasificadores y colaboraciones que llevan a cabo los casos de uso (22).

2.6.1 Descripción del caso de uso tipo

El caso de uso tipo a utilizar será “Gestionar prueba de Reconocimiento”. La siguiente tabla muestra el resumen de la especificación de este caso de uso, que es uno de los artefactos que fueron generados durante el Análisis.

Objetivo	Gestionar prueba de reconocimiento.
Actores	Abogado: (Inicia) Adiciona, modifica, elimina pruebas de reconocimiento en un escrito de solicitud de pruebas.
Resumen	Comienza cuando el abogado necesita gestionar una prueba de reconocimiento en un escrito de proposición de pruebas. Con la posibilidad de adicionarla modificarla o eliminarla del escrito.
Prioridad	Alta
Complejidad	Alta
Precondiciones	Se ha registrado un escrito de proposición de prueba.
Postcondiciones	Se creó trámite en el expediente en el estado: pendiente a disponer pruebas.

TABLA 3 RESUMEN DEL CASO DE USO GESTIONAR PRUEBA DE RECONOCIMIENTO

2.6.2 Diagrama de clases del diseño

Un diagrama de clases es un modelo estático que representa la estructura estática del sistema en términos de las clases y sus relaciones (23).

Una clase es un descriptor de un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y comportamiento (24).

Las relaciones que se establecen entre clases pueden ser de generalización, asociación o dependencia.

La imagen 12 muestra el diagrama de clases del diseño del caso de uso Gestionar prueba de reconocimiento.

Entre las principales clases que se representan en este diagrama se encuentran:

La clase PruebasReconocimiento.html.twig como parte de la capa de presentación. Es la interfaz con la que interactúa el usuario, permitiéndole visualizar los datos necesarios y las acciones que puede realizar.

La clase controladora ReconocimientoJudicialController que se encarga de separar la presentación de la lógica del negocio y contiene aquellas funcionalidades que debe llevar a cabo el caso de uso.

La clase gestora ReconocimientoJudicialGtr que se encarga de listar, registrar, modificar, eliminar, generar documentos y otras series de acciones que permiten dar cumplimiento a las funcionalidades que debe llevar a cabo el caso de uso.

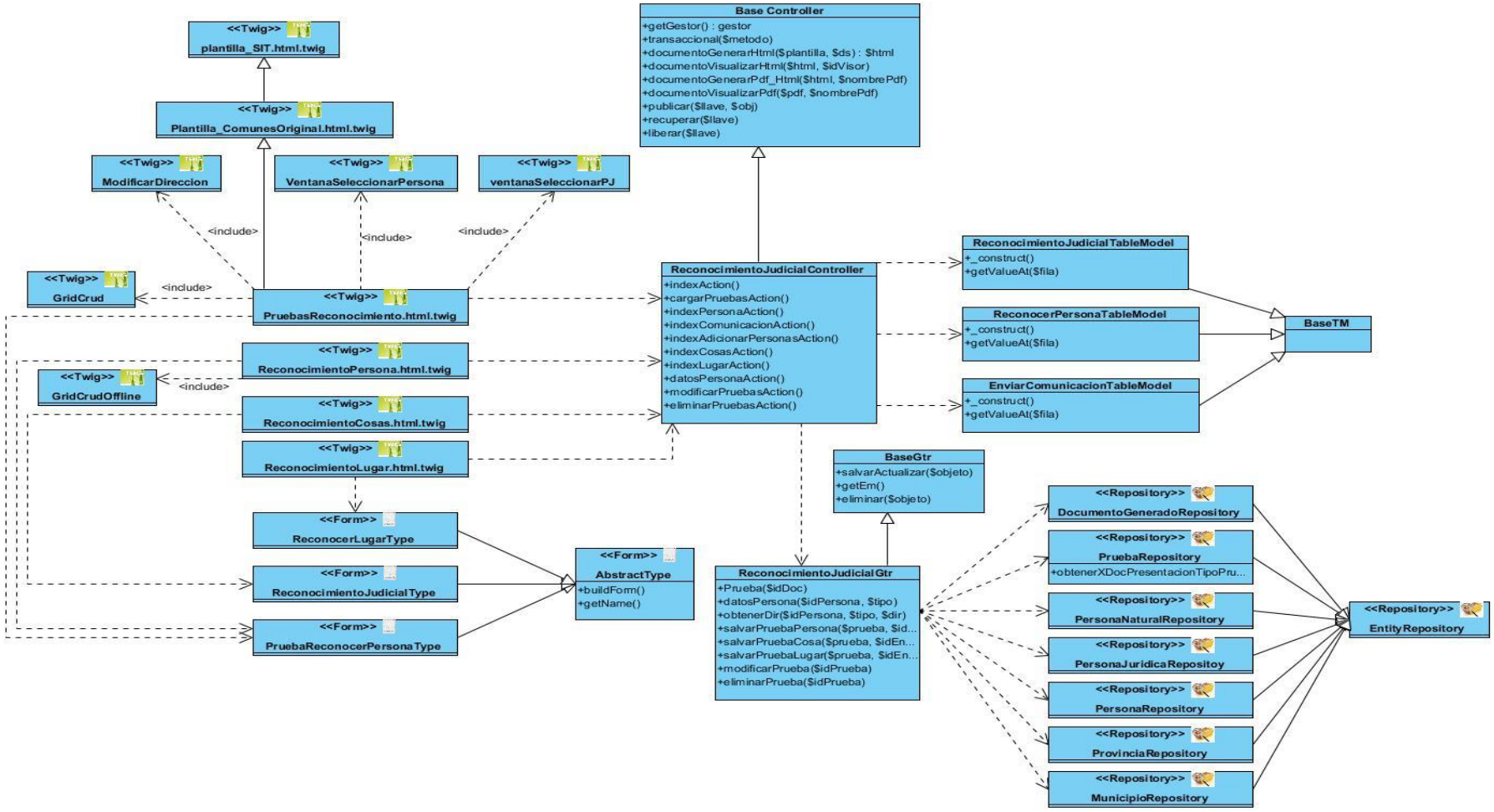


IMAGEN 12 DIAGRAMA DE CLASES DEL DISEÑO GESTIONAR PRUEBA DE RECONOCIMIENTO

2.6.3 Diagrama de secuencia

Los diagramas de secuencia se utilizan para modelar las interacciones entre objetos del diseño. Estos diagramas muestran cómo el control pasa de un objeto a otro a medida que se ejecuta el caso de uso y a medida que se envían mensajes entre objetos. Un mensaje enviado por un objeto dispara la toma del control en el objeto receptor y la realización de las operaciones de su clase (22).

La imagen 13 muestra el diagrama de secuencia del caso de uso Señalar prueba de reconocimiento. En este diagrama se puede apreciar cómo el Juez ponente, que es el actor del sistema, selecciona la pestaña Reconocimiento judicial para hacer el señalamiento, esta petición es enviada del controlador ReconocimientoJudicialController al gestor ReconocimientoJudicialGtr que se comunica con el repositorio para obtener la lista de pruebas de reconocimiento y mostrarla en un GridComun. Posteriormente el juez selecciona una prueba de reconocimiento de la lista y da clic en la opción Disponer para que se muestren los campos necesarios que son llenados con la información del nuevo señalamiento. Cuando ha terminado de llenar los datos selecciona la opción registrar para enviar los datos al controlador que a su vez los envía al gestor para que este último se comunique con el repositorio que guarda la información y actualiza el estado del trámite que se está realizando.

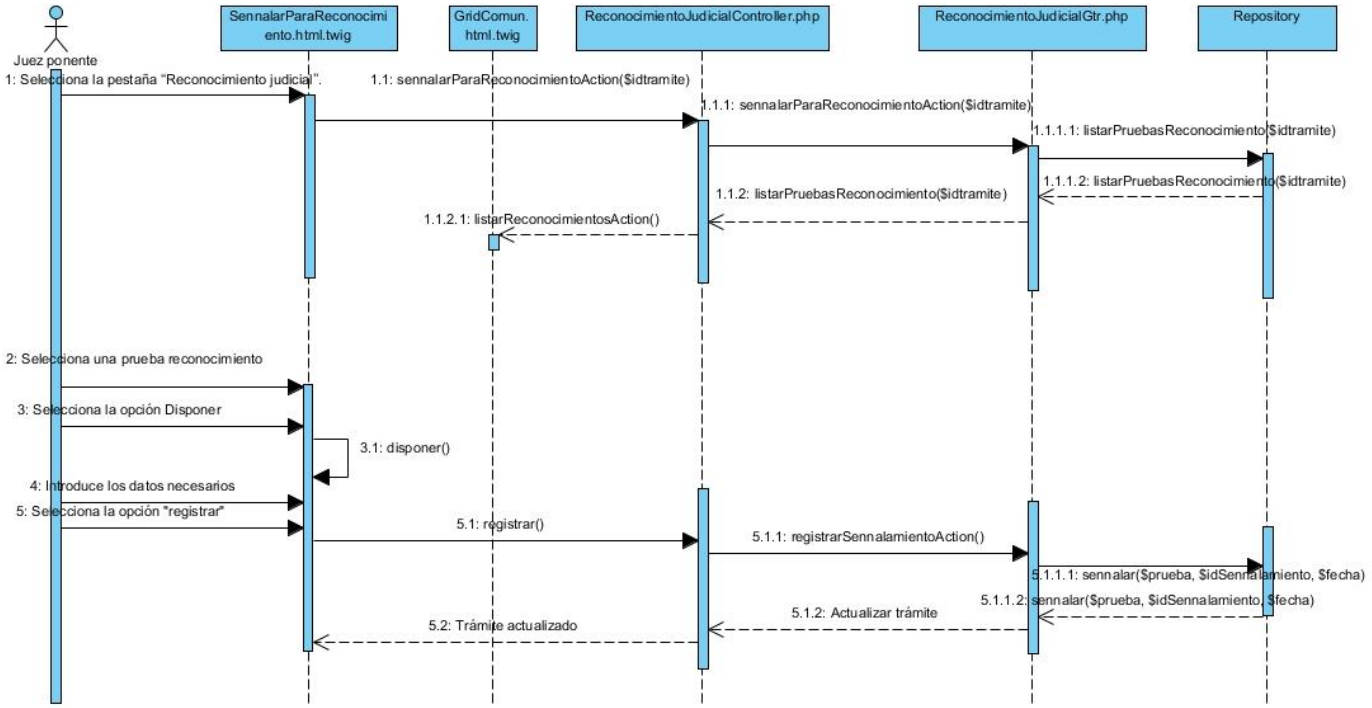


IMAGEN 13 DIAGRAMA DE SECUENCIA CASO DE USO SEÑALAR PRUEBA DE RECONOCIMIENTO

2.7 Diagrama de clases de las entidades del negocio

Con este diagrama se busca tener una representación de las relaciones existentes entre las clases entidades, que no son más que la abstracción de las tablas de la base de datos que son mapeadas previamente por el ORM Doctrine. Estas clases se encuentran en ComunBundle en la carpeta de dependencias de terceros (vendor) de Symfony2.

La imagen 14 muestra un fragmento del diagrama de clases de las entidades del negocio del proceso pruebas. El diagrama completo puede ser consultado en el Anexo 1.

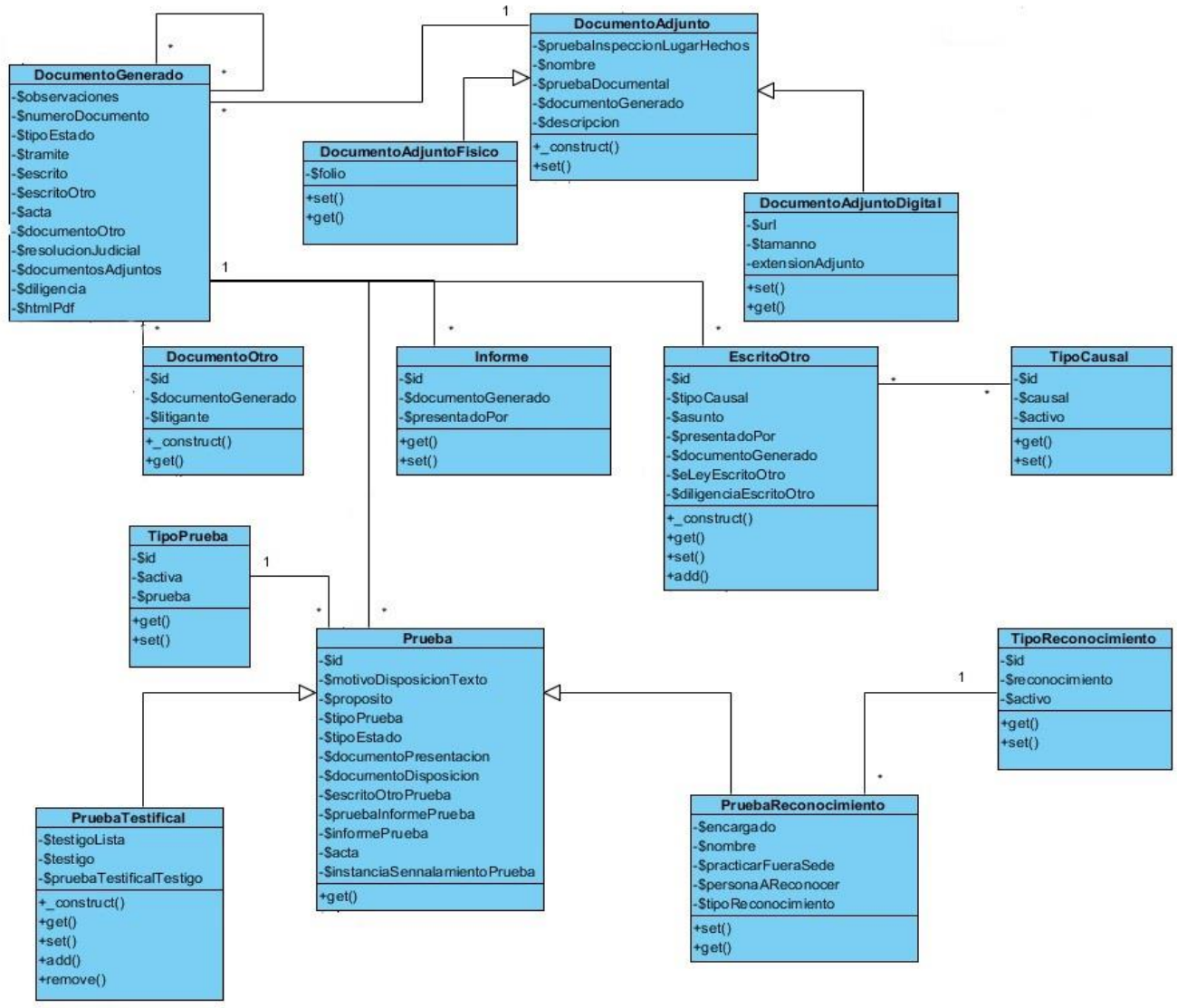


IMAGEN 14 FRAGMENTO DEL DIAGRAMA DE CLASE DE LAS ENTIDADES DEL NEGOCIO

2.8 Modelo de implementación

Durante el flujo de trabajo de implementación se desarrolla todo lo necesario para obtener un sistema ejecutable: componentes ejecutables, componentes de fichero, componentes de tabla, etc. Un componente es una parte física y reemplazable del sistema que cumple y proporciona la realización de un conjunto de interfaces. El modelo de implementación está formado por componentes que incluyen todos los ejecutables (22).

2.8.1 Diagrama de componentes

Un diagrama de componentes muestra las organizaciones y las dependencias entre tipos de componentes. Representa las dependencias entre componentes software, incluyendo componentes de código fuente, componentes del código binario, y componentes ejecutables (24).

El diagrama obtenido permitió modelar el sistema y la interacción entre sus principales componentes. Contiene los procesos Prueba Testifical y de Reconocimiento como parte del módulo Común que a su vez pertenece al SITPC. Se puede apreciar también el modelo donde se encuentran las clases Repository, las gestoras y las clases entidades mapeadas por el ORM Doctrine; así como la capa de datos que contiene a la base de datos. Aparecen además las clases controladoras que hacen uso de las rutas y los servicios pertenecientes al módulo Común, y la parte de la vista compuesta por las plantillas Twig, los formularios y las extensiones. Por otra parte se encuentra el controlador frontal, único punto de acceso a la aplicación desde un entorno determinado, con acceso a los ficheros de configuración general del sistema, y otros componentes que también interactúan con la aplicación y la complementan, como la mensajería, gestión de seguridad y tratamiento de excepciones, que permiten lograr una arquitectura más flexible y adaptable.

La imagen 15 muestra el diagrama de componentes realizado para la implementación de los procesos Prueba Testifical y de Reconocimiento.

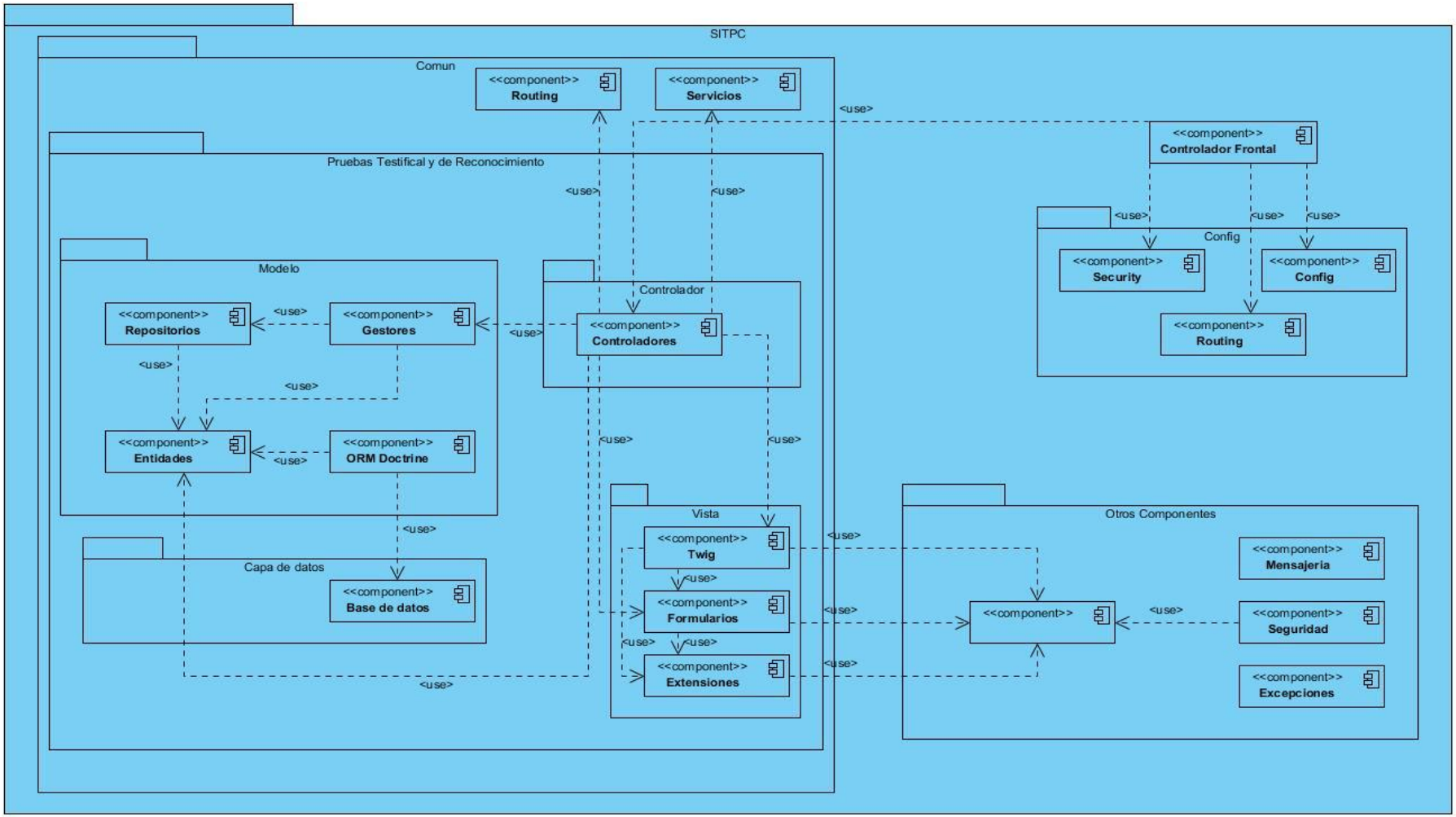


IMAGEN 15 DIAGRAMA DE COMPONENTES

2.9 Modelo de despliegue

Un diagrama de despliegue muestra la configuración de los nodos de proceso y las instancias de componentes y objetos que residen en ellos. Los componentes representan unidades de código en ejecución (24).

El diagrama de despliegue del SITPC está compuesto por Computadoras Personales (PC) clientes que se conectan a un servidor de aplicaciones a través del protocolo HTTPS, este servidor a su vez se conecta con el servidor de base de datos mediante el protocolo TCP/IP. También cuenta con las impresoras que se conectan mediante USB o el protocolo TCP/IP a las PC cliente. Los servidores de base de datos de las diferentes instancias de los tribunales establecen una conexión con el servidor del centro de datos a través de los protocolos TCP/IP, UDP o FTP.

La imagen 16 muestra el diagrama de despliegue realizado para el SITPC, en el cual está representada la distribución de los nodos que se repite para las diferentes instancias de los tribunales, ya sea el Tribunal Supremo Popular (TSP), los Tribunales Provinciales Populares (TPP), o los Tribunales Municipales Populares (TMP).

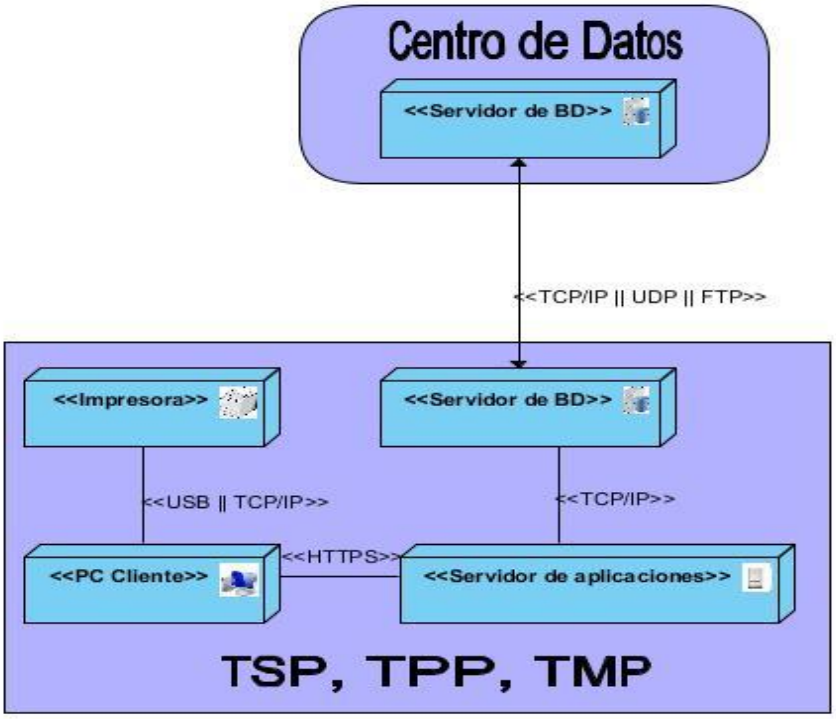


IMAGEN 16 DIAGRAMA DE DESPLIEGUE

Descripción de los nodos físicos

Los nodos son los elementos de hardware que van a soportar el software del sistema desarrollado. Los nodos que aparecen en la imagen 16 se describen a continuación:

- Servidor de Base de Datos (ubicado en el Centro de Datos): en este servidor se encontrará la base de datos que contendrá toda la información de las diferentes instancias de los tribunales.
- Servidor de Base de Datos (ubicado en los tribunales Supremo, Provinciales y Municipales): en este servidor se encontrará la base de datos que contendrá toda la información referente a su instancia respectiva. El mismo se sincronizará con el servidor del Centro de Datos en horario no laboral para enviar la información diaria.
- Servidor de aplicaciones: contendrá todo lo referente a la aplicación, incluyendo los archivos que sean necesarios para que los usuarios puedan tener acceso a esta; manejará la información específica de cada registro y sus clases; tendrá almacenada la configuración general del sistema, bibliotecas externas y los archivos de instalación de la aplicación.
- PC Cliente: tendrá instalado el sistema operativo Linux y el navegador web Mozilla Firefox a través del cual los usuarios podrán acceder a la aplicación y hacer uso de esta.
- Impresora: dispositivo de hardware necesario para imprimir los diferentes documentos que se generan como resultado de las distintas tramitaciones en los tribunales y que son necesarios para el funcionamiento normal de estos.

2.10 Conclusiones parciales

- La arquitectura definida para el sistema permitió obtener componentes independientes que son susceptibles de reutilizarse después en otros sistemas.
- El uso de patrones de diseño permitió asignar correctamente las responsabilidades a las clases, lo que permitió la creación de un código más fácil de comprender, mantener y extender.
- La elaboración del modelo de diseño sirvió como guía para la implementación a fin de materializar con precisión los requerimientos del cliente.
- El modelo de implementación permitió generar los elementos necesarios para un mejor entendimiento del proceso de implementación.
- El modelo de despliegue permitió obtener una visión general de la estructura que se requiere para la posterior fase de transición cuando el producto sea implantado en el entorno real de la organización.

Capítulo 3: Validación de la solución

3.1. Introducción

El presente capítulo tiene como objetivo realizar la validación de la solución propuesta para evaluar los resultados obtenidos tras realizar el diseño y la implementación del componente. Las pruebas y validaciones de software son de vital importancia para corroborar que la aplicación desarrollada no presente problemas de ejecución y se encuentre libre de errores. Se presentan los resultados de la aplicación de las métricas de diseño, así como los casos de prueba de los métodos de caja blanca y caja negra con los resultados de la aplicación de los mismos.

3.2. Validación del diseño

Para realizar la validación del diseño de la solución propuesta se estudiaron varias métricas de diseño y sus características. La finalidad del uso de métricas es evaluar sistemas para conseguir alta calidad y robustez en su diseño. Con el objetivo de medir la complejidad de las clases y las relaciones existentes entre las mismas se seleccionaron cuatro métricas: Tamaño de Clase (TC), Relaciones entre Clases (RC), Carencia de Cohesión en los Métodos (CCM) y Árbol de Profundidad de Herencia (APH).

3.2.1. Métricas de diseño

El glosario de estándares del Instituto de Ingeniería Eléctrica y Electrónica (IEEE) define métrica como una "medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo determinado".

Tamaño de Clase (TC)

El objetivo de esta métrica es medir el tamaño de una clase tomando la cantidad de operaciones y atributos que están encapsulados dentro de la misma. Una clase grande indica alta responsabilidad para la clase y baja reutilización, lo que dificulta los procesos de implementación y prueba de dicha clase; para lograr una mayor reutilización el tamaño de la clase debe ser pequeño.

Para evaluar los resultados de la aplicación de la métrica TC se aplicarán los siguientes umbrales definidos por Lorenz y Kidd.

TC	Umbral
Pequeño	TC<=20
Medio	20<TC<=30
Grande	TC>30

TABLA 4 UMBRALES PARA LA MÉTRICA TC

La siguiente tabla muestra un ejemplo de las medidas de los parámetros de calidad obtenidas luego de aplicar la métrica TC.

Clase	Procedimientos	Responsabilidad	Complejidad	Reutilización
BaseController	9	Baja	Baja	Alta
CrearActaReconocimientoController	23	Media	Media	Media
BaseGtr	3	Baja	Baja	Alta
ActaPruebaTestificalGtr	24	Media	Media	Media
RegistrarTachaTestigoController	15	Baja	Baja	Alta
ReconocimientoJudicialController	11	Baja	Baja	Alta
DisponerSobrePreguntaAntesPruebaTestificalController	8	Baja	Baja	Alta
DisponerSobrePreguntaAntesPruebaTestificalGtr	10	Baja	Baja	Alta
PruebaTestifical	9	Baja	Baja	Alta
InstanciaSennalamientoPruebaRepository	2	Baja	Baja	Alta
PliegoRepository	7	Baja	Baja	Alta
PruebaTestificalRepository	1	Baja	Baja	Alta
TestigoRepository	10	Baja	Baja	Alta
RegistrarEscritoImpugnacionTachaTestigoController	12	Baja	Baja	Alta
RegistrarEscritoImpugnacionTachaTestigoGtr	14	Baja	Baja	Alta
RegistrarEscritoRepreguntasTestificalController	19	Media	Media	Media
ActaPruebaTestificalController	23	Media	Media	Media
Tramite	42	Alta	Alta	Baja
Expediente	37	Alta	Alta	Baja
Litigante	47	Alta	Alta	Baja
TramiteExpediente	9	Baja	Baja	Alta
DocumentoGenerado	50	Alta	Alta	Baja
TipoEstado	8	Baja	Baja	Alta
PersonaNatural	66	Alta	Alta	Baja
PersonaJuridica	29	Media	Media	Media
DocumentoAdjunto	12	Baja	Baja	Alta

TABLA 5 RESULTADOS DE LA APLICACIÓN DE LA MÉTRICA TC

La métrica TC se aplicó a 26 clases, obteniéndose un total de 500 procedimientos, para un promedio de 19,23 procedimientos. De las clases analizadas se obtiene un total de 17 clases de tamaño pequeño, 4 de tamaño medio y 5 de tamaño grande.

Umbral	Tamaño	Cantidad de clases
TC<=20	Pequeño	21
20<TC<=30	Medio	4

TC>30	Grande	5

TABLA 6 CANTIDAD DE CLASES POR TAMAÑO

Estudiando los resultados obtenidos para cada uno de los parámetros de calidad analizados, se puede constatar que el 62% de las clases tienen responsabilidad y complejidad baja. A su vez, el 19% tienen responsabilidad y complejidad media. El 19% restante tienen responsabilidad y complejidad alta. En cuanto a la reutilización el 62% tienen reutilización alta, el 19% tienen reutilización media y el 19% tienen reutilización baja. Con estos datos se obtiene un resultado positivo en la validación del diseño de la solución pues presenta bajos niveles de responsabilidad y complejidad, así como altos índices de reutilización, lo que facilitará la implementación y realización de las pruebas de estas clases.

Los siguientes gráficos de pastel muestran la distribución de los resultados obtenidos.

GRÁFICO 1 RESULTADOS DE RESPONSABILIDAD EN TC

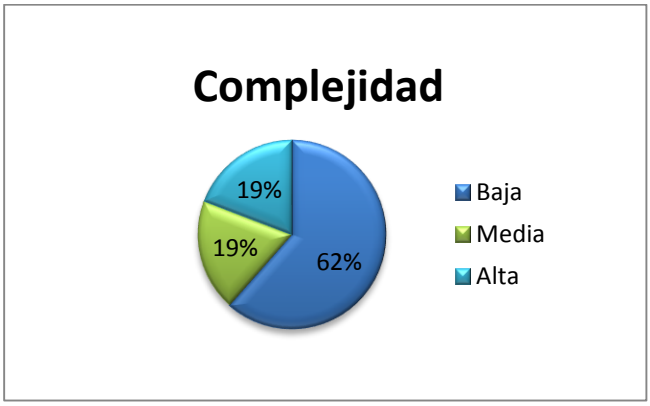


GRÁFICO 2 RESULTADOS DE COMPLEJIDAD EN TC

GRÁFICO 3 RESULTADOS DE REUTILIZACIÓN EN TC

Relaciones entre Clases (RC)

Esta métrica está dada por el número de relaciones de uso de una clase con otra. Para ello mide 4 atributos de calidad que son los que se explican a continuación.

Acoplamiento: un aumento de RC indica un aumento del acoplamiento de la clase.

Complejidad de mantenimiento: un incremento del RC implica un aumento de la complejidad del mantenimiento de la clase.

Reutilización: un aumento del RC implica una disminución en el grado de reutilización de la clase.

Cantidad de pruebas: un incremento del RC implica un aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

En la siguiente tabla se muestran las medidas utilizadas para evaluar estos parámetros de calidad.

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	RC=0
	Bajo	RC=1
	Medio	RC=2
	Alto	RC>2
Complejidad de mantenimiento	Baja	RC <= Promedio
	Media	Promedio<=RC<=2*Promedio
	Alta	RC>2*Promedio
Reutilización	Baja	RC>2* Promedio
	Media	Promedio<=RC<=2*Promedio
	Alta	RC<= Promedio
Cantidad de pruebas	Baja	RC <= Promedio

	Media	Promedio<=RC<=2*Promedio
	Alta	RC>2*Promedio

TABLA 7 CRITERIOS DE EVALUACIÓN PARA LA MÉTRICA RC

La siguiente tabla muestra el resultado de la aplicación de la métrica RC al diseño de la aplicación.

Clase	RC	Acoplamiento	Complejidad de mantenimiento	Cantidad de Pruebas	Reutilización
BaseController	1	Bajo	Baja	Baja	Alta
CrearActaReconocimientoController	2	Medio	Baja	Baja	Alta
BaseGtr	1	Bajo	Baja	Baja	Alta
ActaPruebaTestificalGtr	6	Alto	Alta	Alta	Baja
RegistrarTachaTestigoController	1	Bajo	Baja	Baja	Alta
ReconocimientoJudicialController	4	Alto	Media	Media	Media
DisponerSobrePreguntaAntesPruebaTestificalController	2	Medio	Baja	Baja	Alta
DisponerSobrePreguntaAntesPruebaTestificalGtr	5	Alto	Alta	Alta	Baja
PruebaTestifical	1	Bajo	Baja	Baja	Alta
InstanciaSennalamientoPruebaRepository	2	Medio	Baja	Baja	Alta
PliegoRepository	5	Alto	Alta	Alta	Baja
PruebaTestificalRepository	5	Alto	Alta	Alta	Baja
TestigoRepository	2	Medio	Baja	Baja	Alta
RegistrarEscritoImpugnacionTachaTestigoController	1	Bajo	Baja	Baja	Alta
RegistrarEscritoImpugnacionTachaTestigoGtr	7	Alto	Alta	Alta	Baja
RegistrarEscritoRepreguntasTestificalController	5	Alto	Alta	Alta	Baja
ActaPruebaTestificalController	2	Medio	Baja	Baja	Alta
Tramite	1	Bajo	Baja	Baja	Alta
Expediente	1	Bajo	Baja	Baja	Alta
Litigante	1	Bajo	Baja	Baja	Alta
TramiteExpediente	1	Bajo	Baja	Baja	Alta
DocumentoGenerado	1	Bajo	Baja	Baja	Alta
TipoEstado	1	Bajo	Baja	Baja	Alta
PersonaNatural	1	Bajo	Baja	Baja	Alta
PersonaJuridica	1	Bajo	Baja	Baja	Alta
DocumentoAdjunto	1	Bajo	Baja	Baja	Alta

TABLA 8 RESULTADO DE LA APLICACIÓN DE LA MÉTRICA RC

Al realizar el análisis de los resultados obtenidos para cada uno de los atributos de calidad que mide esta métrica se evidencia que el 27% de las clases presentan un acoplamiento alto, el 19% presenta un acoplamiento medio y el 54% presenta un acoplamiento bajo; con esto queda demostrado que en general el acoplamiento es bajo. En cuanto a la complejidad de mantenimiento el 73% de las clases tienen complejidad de mantenimiento baja, el 4% tienen complejidad de mantenimiento media y el 23% tiene

complejidad de mantenimiento alta. Respecto a la cantidad de pruebas el 73% de las clases presentan una cantidad de pruebas baja, el 3% tiene una cantidad de pruebas media y el 23% presenta una cantidad de pruebas alta. En cuanto a la reutilización de estas clases se puede constatar que el 73% de las clases tienen reutilización alta, el 4% tiene reutilización media y el 23% tiene reutilización baja. Estos datos demuestran que las clases están bien diseñadas ya que al presentar un acoplamiento relativamente bajo junto con una complejidad de mantenimiento baja y una reutilización alta facilita la implementación de estas clases. El hecho de presentar una cantidad de pruebas baja implica que se necesitará menos esfuerzo a la hora de realizar pruebas a estas clases.

Los siguientes gráficos de pastel muestran la distribución de los resultados obtenidos.

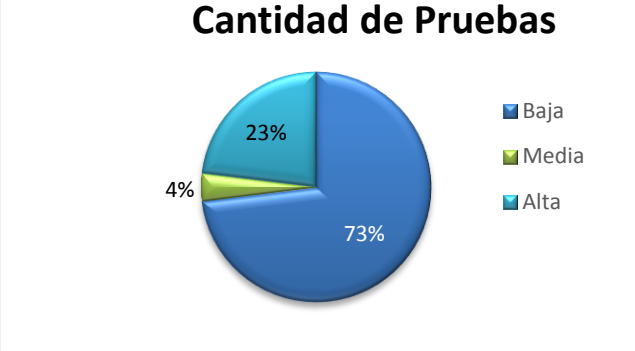


GRÁFICO 4 RESULTADOS CANTIDAD DE PRUEBAS EN RC

GRÁFICO 5 RESULTADOS REUTILIZACIÓN EN RC

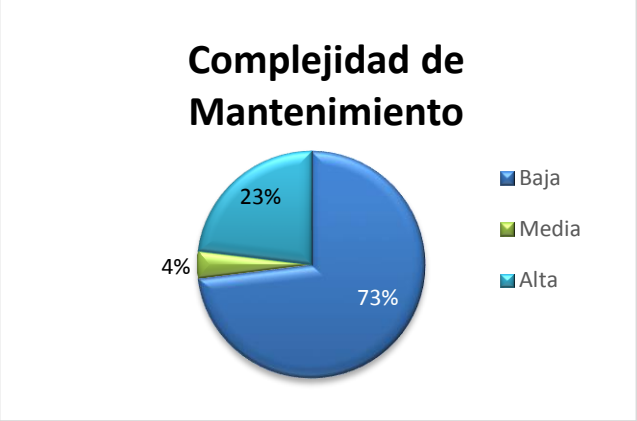


GRÁFICO 6 RESULTADOS COMPLEJIDAD DE MANTENIMIENTO EN RC

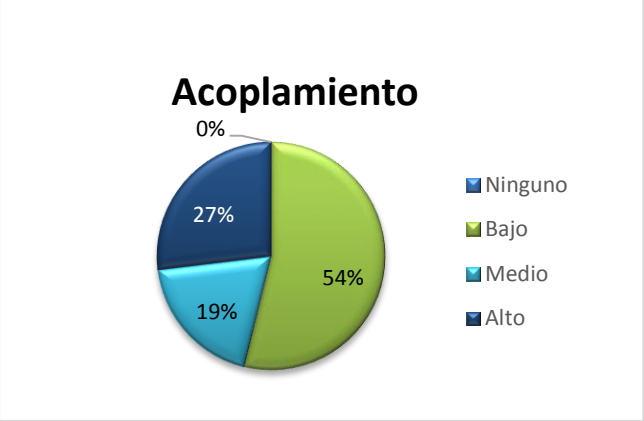


GRÁFICO 7 RESULTADOS ACOPLAMIENTO EN RC

Carencia de Cohesión en los Métodos (CCM)

Esta métrica establece en qué medida los métodos hacen referencia a los atributos. CCM es una medida de la cohesión de una clase midiendo el número de atributos comunes usados por diferentes métodos, indicando la calidad de la abstracción hecha en la clase (25).

Un valor alto de CCM (mayor que ocho) implica falta de cohesión, es decir, escasa similitud de los métodos. Esto puede indicar que la clase está compuesta de elementos no relacionados, incrementando la complejidad y la probabilidad de errores durante el desarrollo. Es deseable una alta cohesión en los métodos dentro de una clase ya que ésta no puede ser dividida fomentando la encapsulación. Se considera que cuando los valores de CCM son altos (mayor que ocho), las clases deben ser rediseñadas descomponiéndolas en dos o más clases distintas. Si ningún método accede a los mismos atributos, entonces el CCM será 0.

Esta métrica fue aplicada al diseño de clases realizado. A continuación se muestra un ejemplo de su aplicación a la clase entidad Prueba, una de las más importante en los procesos Prueba Testifical y de Reconocimiento.

Atributos	Identificador
id	a
motivoDisposicionTexto	b
proposito	c
tipoPrueba	d
tipoEstado	e
documentoPresentacion	f
documentoDisposicion	g
escritoOtroPrueba	h
pruebaInformePrueba	i
informePrueba	j
acta	k
instanciaSennalamientoPrueba	l

TABLA 9 ATRIBUTOS DE LA CLASE PRUEBA

Métodos	Atributos
getId	a
setMotivoDisposicionTexto	b
getMotivoDisposicionTexto	b
setTipoPrueba	d
getTipoPrueba	d
setTipoEstado	e
getTipoEstado	e
setDocumentoPresentacion	f
getDocumentoPresentacion	f
setDocumentoDisposicion	g
getDocumentoDisposicion	g
setProposito	c

getProposito	c
addEscritoOtroPrueba	h
removeEscritoOtroPrueba	h
getEscritoOtroPrueba	h
addInformePrueba	j
removeInformePrueba	j
getInformePrueba	j
_construct	h, i, j, k
addPruebaInformePrueba	i
removePruebaInformePrueba	i
getPruebaInformePrueba	i
addActa	k
removeActa	k
getActa	k
addInstanciaSennalamientoPrueba	l
removeInstanciaSennalamientoPrueba	l
getInstanciaSennalamientoPrueba	l

TABLA 10 MÉTODOS DE LA CLASE PRUEBA

Luego de aplicar esta métrica a la clase Prueba se obtuvo como resultado un valor de CCM igual a 4, lo cual representa un valor aceptable según lo planteado por los autores de la misma. Al aplicarla sobre otras clases, los valores de CCM se mantuvieron coherentes con el resultado obtenido para Prueba, siendo en su mayoría inferiores a 4, lo que demuestra la baja complejidad del diseño realizado y favorece una alta cohesión entre sus clases.

Árbol de Profundidad de Herencia (APH)

Esta métrica mide el máximo nivel en la jerarquía de herencia. APH es la cuenta directa de los niveles en la jerarquía de herencia. En el nivel cero de la jerarquía se encuentra la clase raíz. Cuanto más profunda se encuentra una clase en la jerarquía, mayor será la probabilidad de heredar un mayor número de métodos (25). Se consideran valores altos de APH aquellos que sean mayores que seis.

El uso de la herencia es visto como un compromiso ya que:

- Altos niveles de herencia indican objetos complejos, los cuales pueden ser difíciles de testear y reusar.

- Bajos niveles en la herencia pueden indicar que el código está escrito en un estilo funcional sin aprovechar el mecanismo de herencia proporcionado por la orientación a objetos.

A continuación se muestra un ejemplo del resultado de la aplicación de esta métrica a las clases del diseño de la solución propuesta.

Clases	Valor de APH
BaseController	0
ReconocimientoJudicialController	1
BaseGtr	0
ReconocimientoJudicialGtr	1
ReconocimientoJudicialTableModel	1
ReconocerPersonaTableModel	1
EnviarComunicacionTableModel	1
BaseTableModel	0
DocumentoGeneradoRepository	2
PruebaRepository	2
PersonaNaturalRepository	2
PersonaJuridicaRepository	2
PersonaRepository	2
ProvinciaRepository	2
MunicipioRepository	2
EntityRepository	0
ReconocerLugarType	1
ReconocimientoJudicialType	1
PruebaReconocerPersonaType	1
AbstractType	0

TABLA 11 EJEMPLO DE APLICACIÓN DE LA MÉTRICA APH

Analizando los resultados obtenidos se concluye que el 25% de la muestra presenta un valor de APH igual a 0, el 40% tiene una APH de 1 y el resto (35%) presenta un APH igual a 2. Este resultado evidencia un nivel óptimo de acuerdo a lo planteado por la métrica y teniendo en cuenta que el resto de las clases se comportan de manera similar a la muestra se puede concluir que existe una baja complejidad en el diseño por lo que se puede predecir fácilmente el comportamiento de las clases.

En el siguiente gráfico se muestran los resultados de la aplicación de esta métrica a las clases de la muestra.

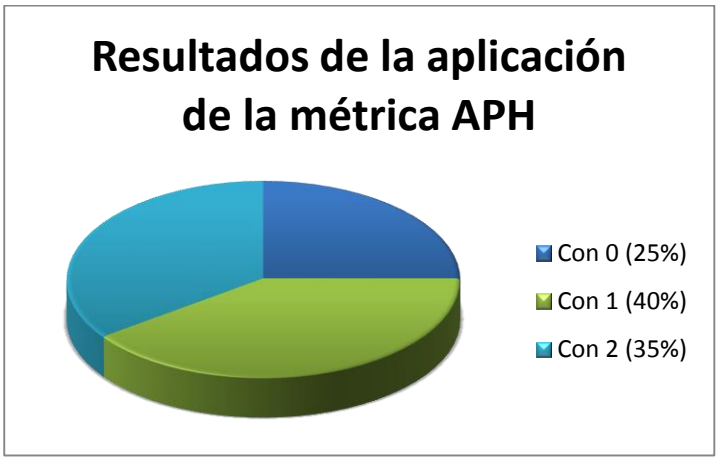


GRÁFICO 8 RESULTADOS DE LA APLICACIÓN DE APH

3.3. Pruebas de software

Las pruebas del software son un elemento crítico para la garantía de la calidad del software y representan una revisión final de las especificaciones, del diseño y de la codificación (26).

Concretamente la prueba de software se puede definir como una actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas (configuración de la prueba), registrándose los resultados obtenidos (27).

Los objetivos de la prueba son:

- Planificar las pruebas necesarias en cada iteración.
- Diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar, creando los procedimientos de prueba que especifican cómo realizar las pruebas.
- Realizar las diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Las construcciones en las que se registran defectos son probadas de nuevo y posiblemente devueltas a otro flujo de trabajo, como diseño o implementación, de forma que los defectos importantes puedan ser arreglados.

Para que una prueba se considere buena existen ciertos atributos que debe cumplir:

- Una buena prueba tiene una alta probabilidad de encontrar un error.
- Una buena prueba no debe ser redundante. No hay motivo para realizar una prueba que tiene el mismo propósito que otra.
- Una buena prueba no debería ser ni demasiado sencilla ni demasiado compleja.

3.3.1. Pruebas de caja blanca

Las pruebas de caja blanca se basan en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado (26).

Según Pressman la prueba de caja blanca se considera como uno de los tipos de pruebas más importantes que se le aplican al software, logrando como resultado que disminuya en un gran porciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad.

La prueba del camino básico es una técnica de prueba de caja blanca propuesta por Tom McCabe. Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico. Se definen una serie de caminos básicos de ejecución, diseñando casos de prueba que garanticen al menos una vez su ejecución.

A continuación se muestran los resultados de aplicar esta técnica al método **obtenerDir($idPersona, $tipo, $dir)** de la clase ReconocimientoJudicialGtr.

```
public function obtenerDir($idPersona, $tipo, $dir) {
    if ($tipo == 1) {
        $persona = $this->getEm()->getRepository('ComunBundle:AG\PersonaNatural')->find($idPersona);
        $datos['nombre'] = $persona->getNombreCompleto();
        if ($dir->getDireccion() != null) {
            $datos['direccion'] = $dir->getDireccion()->getDireccionCompleta();
        } else {
            $datos['direccion'] = "";
        }
        $datos['idPersona'] = $idPersona;
        return $datos;
    } else {
        $persona = $this->getEm()->getRepository('ComunBundle:AG\PersonaJuridica')->find($idPersona)
        $datos['nombre'] = $persona->getNombreCompleto();
        if ($dir->getDireccion() != null) {
            $datos['direccion'] = $dir->getDireccion()->getDireccionCompleta();
        } else {
            $datos['direccion'] = "";
        }
        $datos['idPersona'] = $idPersona;
        return $datos;
    }
}
```

IMAGEN 17 CÓDIGO DEL MÉTODO OBTENERDIR($IDPERSONA, $TIPO, $DIR)

Para el código anterior resultó el siguiente grafo de flujo.

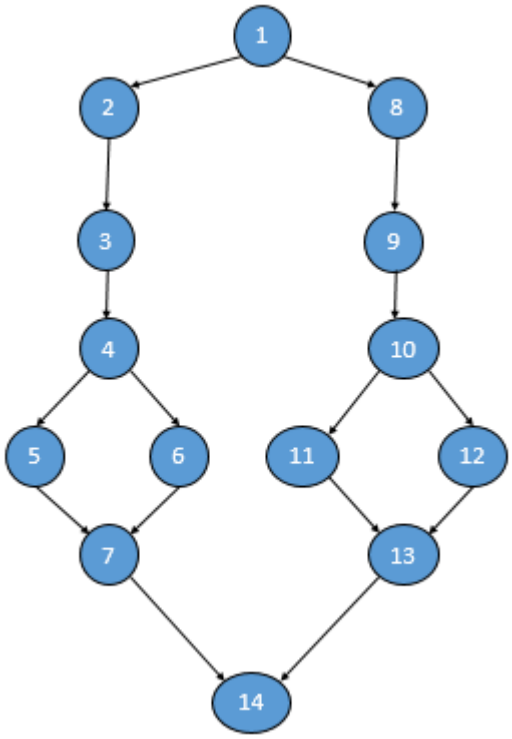


IMAGEN 18 GRAFO DE FLUJO PARA EL MÉTODO OBTENERDIR($IDPERSONA, $TIPO, $DIR)

El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y da un límite inferior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez.

La complejidad ciclomática V(G) se puede calcular de varias formas:

V(G)=A-N+2, donde A es la cantidad de aristas y N la cantidad de nodos.

V(G)=P+1, donde P es la cantidad de nodos predicados.

V(G)= #Regiones.

En el grafo para el método obtenerDir($idPersona, $tipo, $dir), A=14, N=12 y P=3. De acuerdo a las fórmulas anteriores:

V(G)=14-12+2=**4**

V(G)=3+1=**4**

V(G)=**4**

Sabiendo que la complejidad de este método es 4 se sabe que existen 4 caminos básicos por lo que basta probar estos para tener la seguridad de haber ejercitado todas las proposiciones y condiciones del método.

Para obtener el conjunto de caminos básicos se utilizó el método simplificado el cual comienza seleccionando el camino más corto de principio a fin y luego va buscando segmentos no recorridos hasta completar el número de caminos necesarios.

Los caminos resultantes son:

Camino 1: 1-2-3-4-5-7-14

Camino 2: 1-2-3-4-6-7-14

Camino 3: 1-8-9-10-11-13-14

Camino 4: 1-8-9-10-12-13-14

Luego de obtenidos los caminos independientes, se realizaron los casos de prueba para cada uno de ellos. La siguiente tabla muestra el caso de prueba generado para uno de los caminos de ejecución, específicamente para el camino 1-2-3-4-5-7-14.

Entrada	Para que la función se ejecute debe recibir por parámetro el identificador de la persona ($idpersona), el tipo de persona que es ($tipo) y la dirección de la misma ($dir)
Resultados esperados	Devuelve los datos de la dirección completa de una persona
Condiciones	$tipo =1, $dir->getDireccion()='calle 15 #4502'

TABLA 12 CASO DE PRUEBA CAMINO BÁSICO #1

Luego de ejecutados los casos de prueba diseñados se demostró que cada sentencia del código se ejecuta al menos una vez, ejecutándose todas las condiciones lógicas en sus variantes verdaderas y falsas por lo que todos los caminos básicos identificados fueron probados satisfactoriamente demostrando que no existe código innecesario.

3.3.2. Pruebas de caja negra

Las pruebas de caja negra se llevan a cabo sobre la interfaz del software. Los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene (26).

Según Pressman para realizar este tipo de pruebas existen varias técnicas entre las que se encuentran:

Técnica de la Partición de Equivalencia: divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.

Técnica del Análisis de Valores Límites: prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.

Técnica de Grafos de Causa-Efecto: permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

Pressman considera que dentro de estas técnicas la Partición de Equivalencia es de las más efectivas, por lo que fue seleccionada para probar la solución propuesta. Esta técnica trata cada parámetro como un modelo algebraico donde unos datos son equivalentes a otros. Logra reducir un rango amplio de posibles valores reales a un conjunto reducido de clases de equivalencia, entonces es suficiente probar un caso de cada clase, pues los demás datos de la misma clase son equivalentes.

El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada. Una clase de equivalencia representa un conjunto de estados válidos o inválidos para condiciones de entrada. Regularmente, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

RUP es una metodología dirigida por casos de usos pues estos guían todo el proceso de desarrollo. Por tal razón se utilizan los casos de uso como principal elemento para llevar a cabo el proceso de pruebas.

Los casos de prueba correspondientes a cada uno de los casos de uso desarrollados se presentan como uno de los artefactos generados en el presente trabajo.

La siguiente tabla muestra parte del diseño del caso de prueba que fue aplicado en el caso de uso Gestionar prueba de reconocimiento.

Escenario	Descripción	Reconocer	Datos de la persona	Propósito de la prueba	Practicarla fuera de la sede del tribunal	Enviar comunicación a	Respuesta del sistema	Flujo central
EC 1.1 Adicionar prueba de reconocimiento de Persona.	El sistema luego de escoger en el campo de Reconocer el valor Persona, permitirá adicionar los datos necesarios para registrar la prueba.	*V*	*V*	*V*	*V*	*V*	Guarda la prueba de reconocimiento de persona.	Selecciona la pestaña de Reconocimiento, Selecciona la opción Adicionar reconocimiento, llena los datos correspondientes
		Persona	Adiciona una persona a reconocer	Conocer a la persona en conflicto	Chequeado	Selecciona una persona.		
		V	*V*	*V*	*V*	*V*		
		Persona	Adiciona una persona a reconocer	Conocer a la persona en conflicto	Chequeado	No selecciona ninguna persona.		
		V	*V*	*V*	*V*	*V*		
		Persona	Adiciona una persona a reconocer	Conocer a la persona en conflicto	No chequeado	[Este campo no aparece]		

EC 1.2 Los datos son incorrectos	Introduce datos incorrectos	V	I	V		V	Indica que los datos son incorrectos y no guarda los cambios.	y luego la opción Registrar.
		Persona	No adiciona una persona a reconocer	Conocer a la persona en conflicto	Chequeado	Selecciona una persona.		
		V	V	I		V		
		Persona	Adiciona una persona a reconocer	[Vacío]	Chequeado	Selecciona una persona.		
		V	V	I		V		
		Persona	Adiciona una persona a reconocer	[Vacío]	Chequeado	No selecciona ninguna persona.		
		V	I	V	V	V		
		Persona	No adiciona una persona a reconocer	Conocer a la persona en conflicto	No chequeado	[Este campo no aparece]		

TABLA 13 CASO DE PRUEBA ASOCIADO AL CASO DE USO GESTIONAR PRUEBA DE RECONOCIMIENTO

Durante la ejecución de las pruebas se realizaron tres iteraciones por cada caso de uso, en las que se detectaron errores de tipo ortográficos, de validación de interfaces, de funcionalidad y de carga de datos entre otros. Todos estos errores fueron corregidos y en la tercera iteración no se detectó ninguno. En el siguiente gráfico se presentan los resultados obtenidos en cada una de las iteraciones.

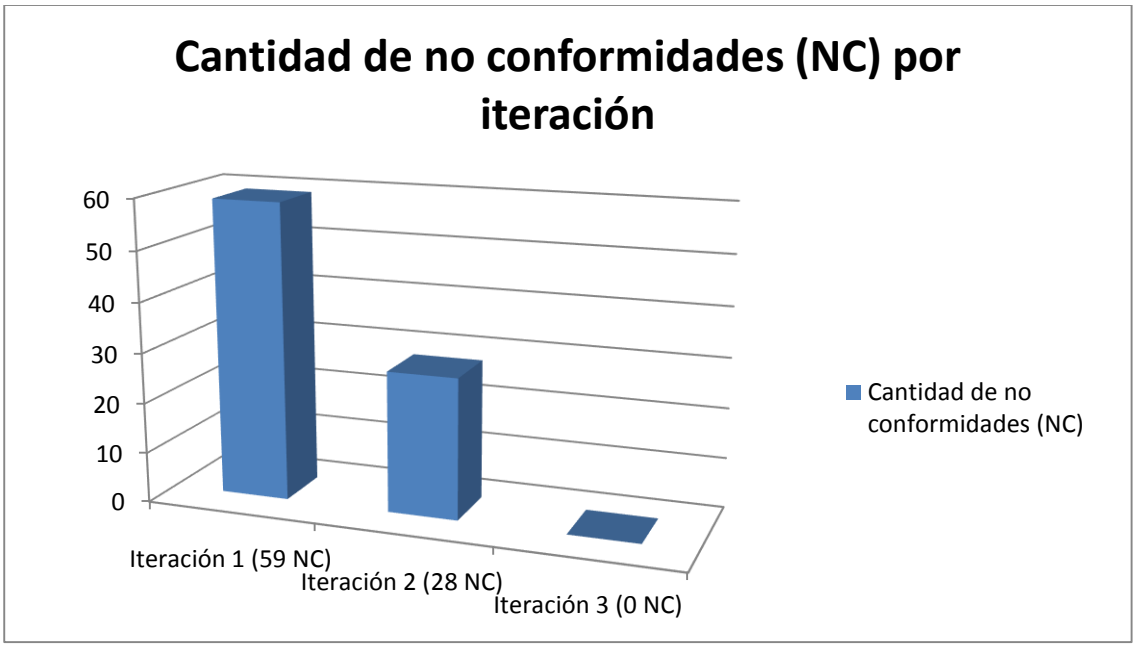


GRÁFICO 9 CANTIDAD DE NO CONFORMIDADES POR ITERACIÓN

Con el componente desarrollado se cumple con los requisitos firmados inicialmente por el Consejo de Gobierno del Tribunal Supremo (máximo órgano que instruye la actividad procesal en el país) para la tramitación de los procesos de Pruebas de Reconocimiento y Testifical; que corrobora que el procedimiento se haga según la ley con todas las variantes y puntos de vista permisibles, lo que evitará trámites y documentos incorrectos. La captura de los datos se hará solamente cuanto sea necesario y se contará además con una carga inicial del Carné de Identidad y Registro de Población, propiciando no duplicar la información en el sistema.

3.4. Conclusiones parciales

- La validación del modelo de diseño mediante el uso de métricas permitió determinar que la mayor parte de las clases son reutilizables, poco dependientes entre sí y colaboran para compartir el esfuerzo si la tarea es grande.
- Se corrigieron los errores detectados durante las pruebas lo que permitió obtener un componente con mayor calidad y confiabilidad.
- Se demostró que el código funciona correctamente y que el sistema realiza las funcionalidades requeridas.

Conclusiones generales

Con la realización del presente trabajo se desarrolló un componente que informatiza los requisitos existentes de los procesos Prueba Testifical y de Reconocimiento. Los resultados alcanzados permiten concluir:

- Se elaboró el marco teórico de la investigación, lo que permitió conceptualizar los elementos fundamentales que fueron utilizados y evidenció la falta de sistemas que gestionen los procesos Prueba Testifical y de Reconocimiento en Cuba y el mundo por lo que se pone de manifiesto la necesidad de la creación de un componente que informatice los mismos.
- Se diseñaron los modelos de la propuesta de solución para realizar una representación técnica de los requisitos existentes, logrando un mejor entendimiento de los procesos de diseño e implementación.
- Se implementó la solución propuesta, lo que permitió obtener un componente de software que puede contribuir a la mejora de la uniformidad en la tramitación y la gestión de la información de los procesos Prueba Testifical y de Reconocimiento en los Tribunales Populares Cubanos.
- Se validó la solución propuesta mediante pruebas de software y el cumplimiento de los objetivos de la investigación, lo que permitió comprobar la correcta funcionalidad del componente obtenido.

Recomendaciones

Luego de la realización de este trabajo se recomienda:

- Realizar las pruebas de aceptación para validar satisfacción de los usuarios con los resultados obtenidos.
- Realizar un estudio una vez desplegado el sistema que permita demostrar estadísticamente el impacto social en los Tribunales Populares Cubanos.

Referencias

1. Asamblea Nacional del Poder Popular. Parlamento cubano. [En línea] 2013. [Citado el: 3 de Diciembre de 2013.] http://www.parlamentocubano.cu/index.php/labor-legilativa/leyes/284-ley-no-70-de-los-tribunales-populares.html.

2. Albán, Dr. Francisco Iturralde. SlideShare. [En línea] 09 de 08 de 2012. [Citado el: 20 de 09 de 2013.] www.slideshare.net/videoconferencias/1-historia-de-la-prueba-judicial.

3. González Montes, José Luis . Informativo Jurídico. [En línea] 01 de 06 de 2012. [Citado el: 21 de 09 de 2013.] http://m.informativojuridico.com/la-prueba-de-reconocimiento-judicial-admisi%C3%B3n-y-pr%C3%A1ctica-en-el-proceso-civil.

4. Ámbito jurídico. [En línea] 1998-2013. [Citado el: 3 de Diciembre de 2013.] http://www.ambito-juridico.com.br/site/?n_link=revista_artigos_leitura&artigo_id=10802&revista_caderno=21.

5. López Gallardo, José Ramón. Iuris Civilis. Blog jurídico de derecho civil. [En línea] 2008-2013. [Citado el: 3 de Diciembre de 2013.] http://www.iuriscivilis.com/2009/03/la-regulacion-legal-de-la-prueba.html.

6. Pericias Grafotécnicas. [En línea] 31 de 01 de 1997. [Citado el: 23 de 06 de 2013.] http://periciasgrafotecnicas.com/index.php?option=com_content&view=article&id=70&catid=25&Itemid=1.

7. Gacitúa Bustos, Ricardo A. *Métodos de desarrollo de software: El desafío de la estandarización.* 2003.

8. Figueroa, Roberth G., Solis, Camilo J. y Cabrera, Armando A. *Metodologías tradicionales vs metodologías ágiles.* Universidad Técnica Particular de Loja : s.n.

9. Kruchten, P. *The Rational Unified Process: An Introduction.* s.l. : Addison Wesley, 2000.

10. Kozaczynski, Wojtek, Brown, Alan W. y Wallnau, Kurt C. *"The Current State of CBSE".* 1998.

11. Szyperski, Clemens. *"Component Software Component Software Beyond Object – Oriented Programming".* 1998.

12. Microsoft. Microsoft Developer Network. [En línea] 2014. [Citado el: 1 de abril de 2014.] http://msdn.microsoft.com/es-es/library/bb972268.aspx.

13. IEEE-SA. IEEE Standards Association. [En línea] 2014. [Citado el: 16 de Marzo de 2014.] https://standards.ieee.org/findstds/standard/1471-2000.html.

14. Almeira, Adriana Sandra y Perez Cavenago, Vanina. *Arquitectura de Software:Estilos y Patrones.* Facultad de Ingeniería. Universidad Nacional De La Patagonia San Juan Bosco. Argentina : s.n., 2007.

15. symfony.es. [En línea] [Citado el: 25 de 09 de 2013.] http://symfony.es/que-es-symfony.

16. Eguiluz, Javier. *Desarrollo web ágil con Symfony2.* 2011.

17. Pacheco, Nacho. *Manual de Twig.* 2011.

18. Larman, Craig. *UML y patrones.* Mexico : Prentice Hall, 1999. ISBN 970-17-0261-1.

19. Departamento de Sistemas Informáticos y Computación. *Rational Unified Process (RUP).* Universidad Politécnica de Valencia : P.Letelier.

20. Data & Object Factory, LLC. [En línea] 2001. [Citado el: 27 de octubre de 2013.] http://dofactory.com/Patterns/Patterns.aspx.

21. Jacobson, Ivar. *Object-oriented development in an industrial enviroment.* s.l. : Proceeding of OOPSLA ´87, 1998.

22. Jacobson, Ivar, Booch, Grady y Rumbaugh, James. *El proceso unificado de desarrollo de software.* Madrid : Addison Wesley, 2000. ISBN: 84-7829-036-2 .

23. Sommerville, Ian. *Ingeniería del software. Séptima edición.* Madrid : Addison Wesley, 2005. ISBN: 84-7829-074-5.

24. Rumbaugh, James, Jacobson, Ivar y Booch, Grady. *El lenguaje unificado de modelado. Manual de referencia.* Madrid : Addison Wesley, 1998.

25. Rodríguez, Daniel y Harrison, Rachel . *Medición en la orientación a objetos.* School of Computer Science, Cybernetics & Electronic Engineering : s.n.

26. Pressman, Roger S. *Ingeniería del Software, un enfoque práctico. 5ta edición.* s.l. : Mc Graw Hill, 2002.

27. Juristo, Natalia, Moreno, Ana M. y Vegas, Sira . Técnicas de evaluación de software. 2005.

28. Flower, Martin. Programación en Castellano. Contenedores de Inversión de Control y el patrón de Inyección de Dependencias. [En línea] 2011. http://www.programacion.com/articulo/contenedores_de_inversion_de_control_y_el_patron_de_inyeccion_de_dependencias_304..

29. Bass, L., Clements, P. y Kazman, R. *Software Architecture in practice.* s.l. : Addison Wesley, 1998.

30. Bosch, J. *Design & Use of Software Architectures.* s.l. : Addison Wesley, 2000.

31. Sergi Santos, Sandra. IBM. Developers Works. *Comparing the Rational Unified Process (RUP) and Microsoft Solutions Framework (MSF).* [En línea] [Citado el: 20 de octubre de 2013.] http://www.ibm.com/developerworks/rational/library/apr07/santos/.

32. Gorphe. *De la apreciación de las pruebas.* Buenos Aires : Ediciones Jurídicas Europa-América, 1955.

33. Carnelutti, Francisco. *La prueba civil.* Buenos Aires : Ediciones Acayú, 1955.

34. Casado Gandolff, Jorge Luis. Aplicaciones Web vs. Aplicaciones de Escritorio. [En línea] 14 de 03 de 2008. [Citado el: 15 de 09 de 2013.] http://webvsdesktop.blogspot.com/.

35. Ministerio de Justicia. Gaceta oficial de la República de Cuba. [En línea] 2008. [Citado el: 27 de octubre de 2013.] http://www.gacetaoficial.cu/html/legislacion_cubana.html. ISSN 1682-7511.

36. Alonso, F.J. *Curso de metodología de la investigación.* Santa Clara, Cuba : Facultad de Ciencias Sociales y Humanísticas. Universidad Central de Las Villas, 1998.

37. Castro, Fernando Lorenzo. *Modelo de Datos. Conceptos y clasificación.* s.l. : Universidad de Castilla-La Mancha, Escuela Superior de Informática, 1999.

38. Diccionario panhispánico de dudas. [En línea] Real Academia Española, 2005. [Citado el: 1 de abril de 2014.] http://lema.rae.es/dpd/srv/search?key=componente.

39. Ariza Rojas, Maribel y Molina García, Juan Carlos. *Introducción y principios básicos del desarrollo de software basado en componentes.* Bogotá, Colombia : Pontificia Universidad Javeriana, 2004.

40. Portal de Administración de Justicia. [En línea] Gobierno de España. [Citado el: 5 de abril de 2014.] https://www.administraciondejusticia.gob.es/paj/publico/ciudadano/informacion_institucional/modernizacion/modernizacion_tecnologica/infolexnet/que_es/!ut/p/c4/04_SB8K8xLLM9MSSzPy8xBz9CP0os3g_A1cjCydDRwMLY2cTA08ndwtnJw9XQwN3A_2CbEdFAHoJ5w0!/.

41. Gobierno de España. Ministerio de Justicia. [En línea] 5 de abril de 2014. https://lexnet.justicia.es.

42. Justice online. [En línea] [Citado el: 5 de abril de 2014.] http://justiceonline.com.sg/index.html.

43. Finlex. [En línea] [Citado el: 5 de abril de 2014.] https://www.finlex.fi/fi/.

44. Singapore Academy of Law. Lawnet Comprehensive Legal Solutions. [En línea] 2014. [Citado el: 5 de abril de 2014.] http://www.lawnet.com.sg/lnrweb/c/portal/layout?p_l_id=1.

45. Gómez Baryolo, Oiner , Rivero Pino, Noel Jesús y López Méndez., Daniel E. Sistema de gestión integral de seguridad Acaxia. *http://publicaciones.uci.cu/index.php/SC.* 2011. Vol. IV, 7.

46. Beatriz Pérez , Lamancha. Gestión de las pruebas funcionales. Centro de Ensayos de Software. Universidad de la República, Montevideo, Uruguay : s.n., 2004. Vol. 1, 4.

47. Tuya, Javier. Las Pruebas del Software. Universidad de Oviedo : s.n., 2007.

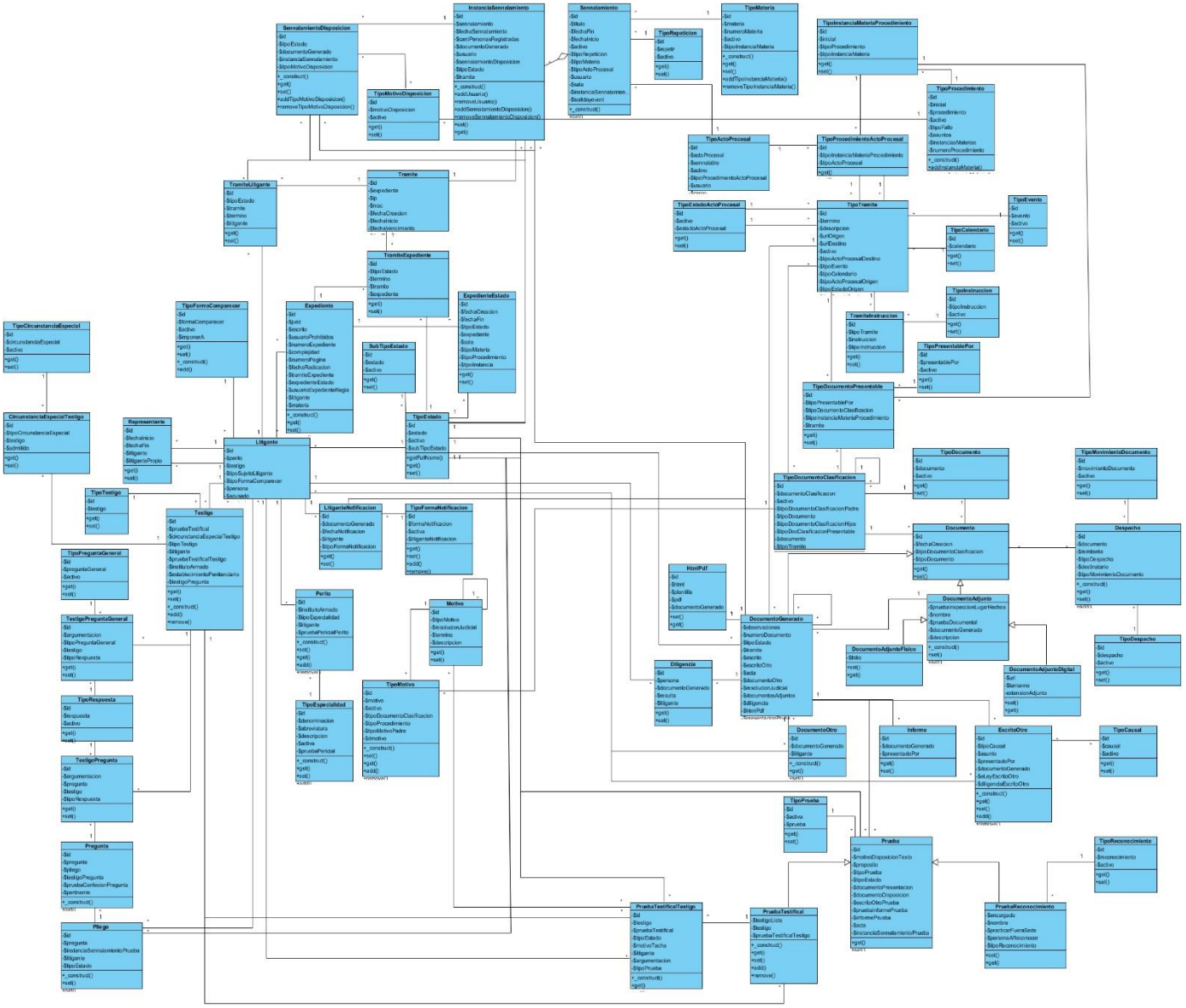
48. Lincke, R., Lundberg, J. y Löwe, W. Comparing Software Metric Tools. s.l. : Compilation Proceedings of the 2008 International Symposium on Software Testing and Analysis and Co-Located Workshops., 2008.

49. Lillo Lobos, Ricardo. *El Uso de Nuevas Tecnologías en el Sistema Judicial: experiencias y precauciones.* Brasilia : Centro de Estudios de Justicia de las Américas. VIII Seminario de Gestión Judicial, 2010.

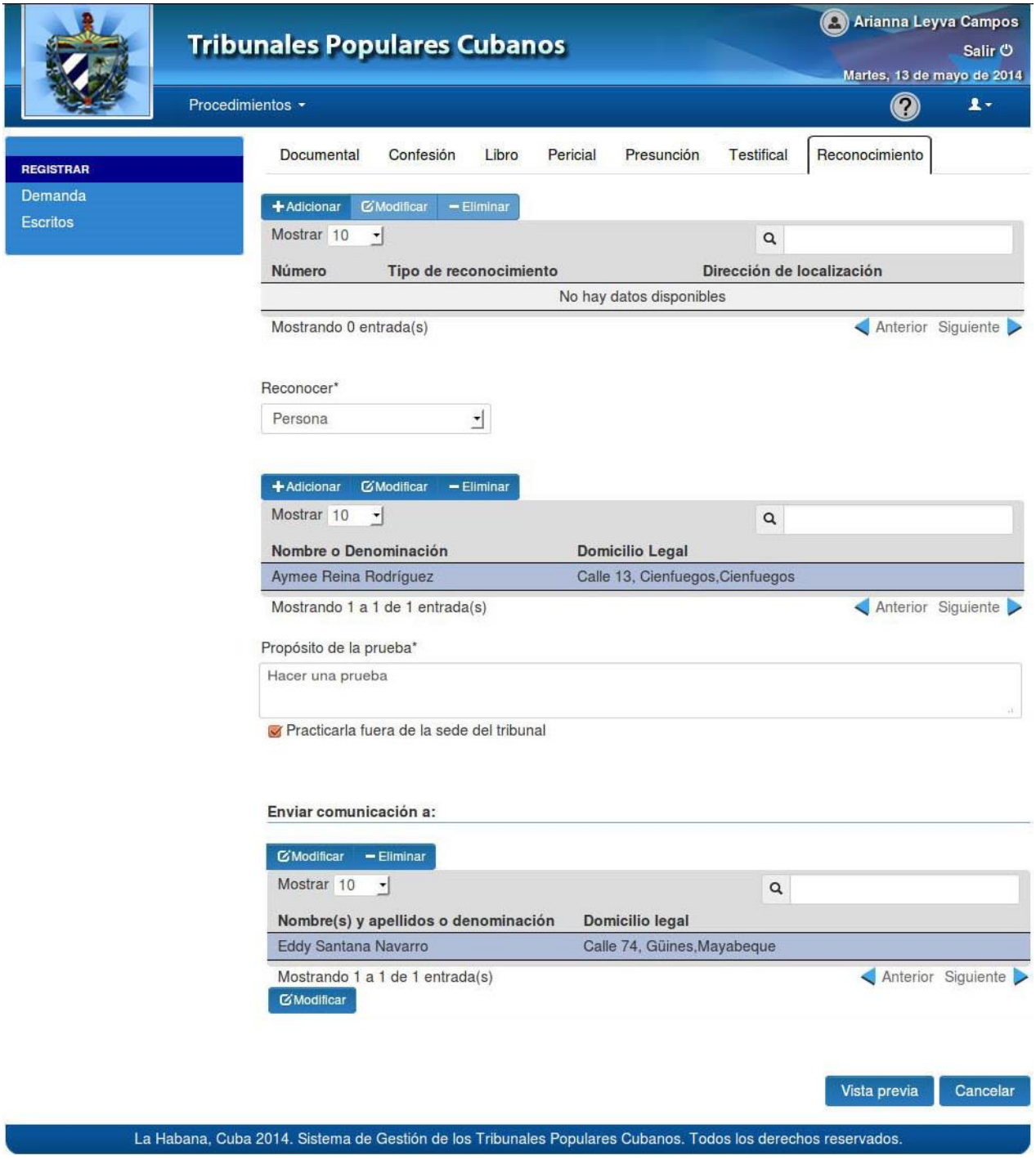
50. Londoño Sepúlveda, Néstor Raúl. El uso de las TIC en el proceso judicial: una propuesta de justicia en línea. Medellín, Colombia. : Revista Facultad de Derecho y Ciencias Políticas, 2010. Vol. 40. ISSN 0120-3886.

51. Davis, G. y Olsón. *Management Information Systems: Conceptual foundations, Structure and Development.* New York : McGrawhill, 1985.

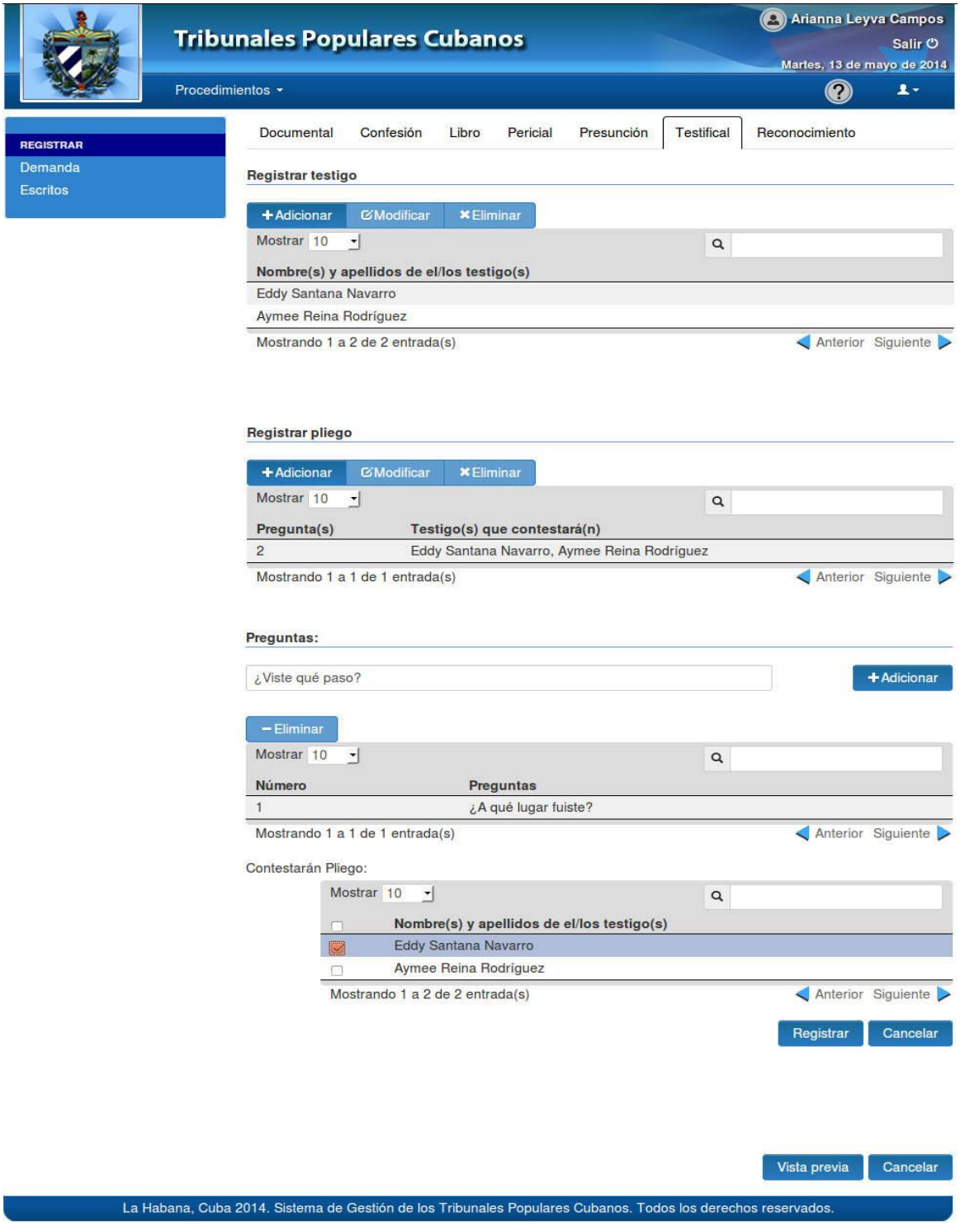
Anexos



ANEXO 1 DIAGRAMA DE CLASES DE LAS ENTIDADES DEL NEGOCIO



ANEXO 2 VISTA DEL CASO DE USO GESTIONAR PRUEBA DE RECONOCIMIENTO



ANEXO 3 VISTA DEL CASO DE USO GESTIONAR PRUEBA TESTIFICAL