Universidad de las Ciencias Informáticas

Facultad 3

Título: Repositorio de Esquemas y Metadatos

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autor: Jorge César Labrador Pérez-Castañeda

Tutor: Ing. Jorge Luis Valdés Gonzáles

Junio, 2011.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Jorge César Labrador Pérez Castañeda

_____

(Autor)

Jorge Luis Valdés Gonzáles

_____

(Tutor)

AGRADECIMIENTOS

Gracias a mis amigos, sin ustedes no concibo mi vida.

Gracias a mi novia, por estar ahí desde el principio y apoyarme en todo, por motivarme a ser mejor cada día.

Gracias a mis compañeros por nutrirme con su ejemplo y brindarme siempre su ayuda.

Gracias a Machín, por ser tantas cosas al mismo tiempo.

Gracias a Tía Dora, me enseñaste que la familia no se mide por lazos de sangre.

Gracias a mi abuela, tu ejemplo perdurará por mucho más tiempo en mi corazón.

Gracias al resto de mi familia, parte indisoluble de todo lo que soy.

Gracias a todas las personas que han pasado por mi vida dejando siempre algo positivo.

DEDICATORIA

Le dedico mi tesis, mi carrera, lo que he sido y lo que seré, a las dos personas más importantes de mi vida: a mi mamá y a mi tío. Tan infinitos… más grandes que la vida misma.

Solamente aquel que construye el futuro tiene derecho a juzgar el pasado.

Friedrich Nietzsche

RESUMEN

El intercambio de información entre sistemas es un elemento fundamental para el Gobierno Electrónico. Para obtener esta característica, en el Órgano de Justicia-MININT se ha concebido una plataforma de interoperabilidad. Esta plataforma utiliza esquemas XML para definir el formato de intercambio de datos. Sin embargo, no existe una forma en que los sistemas puedan acceder a los esquemas, ni de registrar y validar los mismos de forma centralizada. Por este motivo, se propone el desarrollo del Repositorio de Esquemas y Metadatos para la plataforma de interoperabilidad del Órgano de Justicia-MININT de Cuba, siendo ese el tema del presente informe. Esta propuesta puede facilitar el acceso a los esquemas y la capacidad de realizar procesos de validación y actualización de los mismos. Para el desarrollo de la solución se efectuó un estudio de iniciativas similares en otros lugares del mundo y se transitó por el proceso de desarrollo de software definido por RUP. Finalmente se validó la propuesta a través de pruebas que aseguraron su confiabilidad.

Palabras clave: esquemas XML, Gobierno Electrónico, interoperabilidad, Repositorio de esquemas y metadatos.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La gestión gubernamental a nivel global en los últimos años ha estado matizada por búsqueda en cada país de una solución viable para implementar una plataforma de Gobierno Electrónico (eGobierno) que satisfaga sus necesidades. Esta búsqueda es motivada por las ventajas que ofrece una plataforma de ese tipo en cuanto a la agilización de los procesos de gobierno, el ahorro de recursos, la transparencia de la gestión y el acercamiento que provee con el ciudadano. Es por eso que el eGobierno puede verse como un instrumento para facilitar la interacción entre los ciudadanos y el Estado a través los medios informáticos.

"El gobierno electrónico – anteriormente una osada aventura y hoy una herramienta importante para la transformación en el sector público – ha progresado hasta el punto en que actualmente constituye una importante pieza para una gobernabilidad y participación ciudadana efectivas…" (UNDESA, 2010).

Cuba, en su esfuerzo por reducir la brecha digital, se ha mantenido dando pasos hacia la implementación de una plataforma de eGobierno viable. En su informe sobre presencia en Internet, publicado en el año 2010, las Naciones Unidas reflejan lo anterior. Con un índice de desarrollo web de 0.3990 en el 2008, Cuba se encontraba en el puesto 111 a nivel global. Ya en el 2010, habiendo ascendido su índice a 0.4321, se encontraba en la posición 96 (UNDESA, 2010). Sin embargo, a pesar de que el avance es innegable, es evidente que falta mucho camino por recorrer.

Es usual encontrar que los datos necesitados por los directivos a cualquier nivel para tomar decisiones más acertadas, se encuentran disponibles pero son de difícil acceso. Además de existir datos duplicados, se evidencia también la ausencia de términos comunes de referencia y de formas de representar dichos datos. Esto provoca el costo en tiempo y recursos de la ardua tarea que representa comparar datos que están representados de manera diferente. Los ejemplos de eGobierno actuales muchas veces se asemejan a un mosaico de informaciones y soluciones tecnológicas de comunicación incompatibles entre sí. El Órgano de Justicia-MININT no escapa a esta situación.

El mencionado Órgano está compuesto por el Ministerio de Justicia, la Fiscalía General de la República de Cuba, el Tribunal Supremo Popular (TSP) y el Ministerio de Interior (MININT). Cuenta con una Secretaría Ejecutiva dirigida por la Fiscalía General de la República (FGR) y con un Grupo Técnico. Su objetivo

principal es contribuir a la toma de decisiones sobre los aspectos más significativos del Sistema de Justicia en Cuba nutriéndose con los datos obtenidos de los organismos antes citados. Sin embargo, estos datos no presentan uniformidad para su intercambio, dado que provienen de los disímiles sistemas informáticos que emplea cada organismo, algunos de los cuales están ya obsoletos y en todos los casos actúan aislados unos de otros. Esto impide que los mismos sean capaces de interactuar entre sí, constituyendo esa incapacidad un obstáculo para la rapidez y calidad de la gestión. Teniendo en cuenta lo anterior se puede decir que no existe interoperabilidad entre los sistemas.

La interoperabilidad contribuye a que los datos obtenidos de diferentes fuentes puedan ser utilizados en conjunto para tomar decisiones más rápidas y mejores. Por esa razón, constituye un factor fundamental para la implementación de una infraestructura de eGobierno, y como tal la identificó la Comisión Económica para América Latina y el Caribe (CEPAL, 2007).

Según estudios europeos, el fenómeno de la interoperabilidad se aborda desde cuatro aspectos fundamentales: semánticos, organizacionales, técnicos y de gobernanza (Study on Interoperability at Local and Regional Level, 2007). Los anteriores aspectos se cimentan sobre la capacidad de intercambiar datos entre sistemas, y dicha capacidad es imposible si estos datos no son compatibles entre sí. Una forma de lograr esa compatibilización es la adopción de estándares –"acuerdo entre partes independientes acerca de cómo hacer para realizar alguna tarea" (Bloomberg, y otros, 2006)– que definan de manera centralizada el formato de los datos que se intercambian.

Teniendo en cuenta lo anterior, se decidió en el Órgano de Justicia-MININT la implementación de una plataforma de interoperabilidad que permita crear un entorno colaborativo para sus sistemas informáticos. Para lograrlo, se precisó la homogenización del formato de intercambio de datos, utilizándose los esquemas XML.

Pero la adopción de esquemas comunes es solo la mitad del camino. Aun cuando se definan los mismos, queda pendiente la labor de garantizar la accesibilidad a gran escala. Enviar a los usuarios uno por uno las actualizaciones de un esquema no es factible, pues es muy difícil poseer una lista de todos los interesados, y a medida que aumenta la cantidad de estos se vuelve más complejo. Por otro lado causaría una excesiva duplicación del esquema. Tampoco garantiza el uso de la versión más actualizada ni que

estén validados por entidades competentes. Otro elemento negativo es la disgregación que se produce, dificultando la tarea de contabilizar y analizar los esquemas en conjunto.

En Cuba no existe ningún mecanismo que centralice esa tan necesaria estandarización, pues no hay ninguna entidad que provea la funcionalidad de publicar u obtener esquemas XML.

A raíz de lo anteriormente planteado, este trabajo pretende solucionar el siguiente **problema**: *es preciso que todos los sistemas del Órgano de Justicia-MININT tengan acceso a todos los estándares que definen el formato de intercambio de los datos, para que de este modo puedan procesar y visualizar un determinado dato obtenido a partir de la interacción con otro sistema a través de la plataforma de interoperabilidad.*

Para ello se persigue como **objetivo general** d*esarrollar un repositorio que permita el registro, gestión y publicación de esquemas y metadatos, a partir del cual los sistemas informáticos de la plataforma de interoperabilidad puedan acceder a los esquemas XML que definen el formato de intercambio de datos.*

El **objeto de estudio** en el que se enmarca es: *El proceso de desarrollo de software*.

Los **objetivos específicos** de este trabajo son los siguientes:

1- Identificar mecanismos de interoperabilidad basados en el uso de XML para la estandarización a partir del estudio de marcos normativos y experiencias en el ámbito internacional.
2- Desarrollar el análisis y diseño del Repositorio de Esquemas y Metadatos.
3- Implementar el sistema a partir del diseño realizado.
4- Validar el diseño y funcionamiento del sistema a partir de métricas de diseño y pruebas de caja negra para evaluar el cumplimiento de los requerimientos definidos.

Campo de acción: Desarrollo de sistemas de interoperabilidad basados en el estándar XML.

Las **tareas** a realizar, encaminadas al cumplimiento de los objetivos planteados, son las siguientes:

Número	Tarea	Objetivo	Capítulo
1.	Realizar un análisis sobre los frameworks de interoperabilidad de una plataforma de eGobierno.	1	1
2.	Evaluación del sistema Administrador de Esquemas y Metadatos de Chile	1	1
3.	Evaluación del lenguaje común de intercambio de información de Colombia	1	1
4.	Modelación del negocio.	2	2
5.	Definición de los requerimientos del sistema.	2	2
6.	Realización del Diagrama de Casos de uso del Negocio.	2	2
7.	Definición del Diseño Teórico de la Tesis	1	1
8.	Estudio de factibilidad tecnológica.	2	1
9.	Implantación y configuración de las tecnologías (IDEs, SGBD, otras herramientas)	2	3
10.	Diagrama de Actividades del Negocio	2	2
11.	Implantación de la Línea Base de la Arquitectura.	2	3
12.	Definición de las clases del análisis del sistema.	2	3
13.	Definición de las clases del Diseño del Sistema	2	3
14.	Realización del Diagrama de Clases del Diseño	2	3
15.	Realización del Diagrama de Componentes.	3	3
16.	Implementación de la solución informática.	3	4
17.	Modelo de Despliegue	3	4
18.	Realización de pruebas pilotos para comprobar el cumplimiento de los requerimientos establecidos.	4	4

Tabla 1: Tareas de la Investigación.

Resultado esperado: Repositorio de Esquemas y Metadatos.

El presente trabajo de diploma quedará estructurado en cuatro capítulos:

- **Capítulo 1**: Fundamentación teórica.

En este capítulo se describen los principales conceptos relacionados con el tema. Se realiza un estudio sobre los elementos teóricos y las soluciones similares en otros países, un estudio del software, tecnologías y herramientas utilizadas, analizando sus características, ventajas y desventajas.

- **Capítulo 2:** Características del sistema.

Se abordan las características principales del sistema a construir. Se explica la manera en que son realizados los diferentes procesos y los que serán automatizados. Incluye la modelación del negocio y la modelación del sistema, la especificación de los requisitos funcionales y no funcionales como elementos indispensables y deseables con que debe cumplir la aplicación.

- **Capítulo 3:** Diseño del sistema.

Comprende fundamentalmente la realización del Modelo del Diseño y la obtención de sus principales artefactos, como los diagramas de clases del diseño y la especificación de las operaciones.

- **Capítulo 4:** Implementación y validación de la propuesta.

Se realiza la implementación del sistema y se valida a través del método de prueba de caja negra y los casos de pruebas correspondientes.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

Hoy en día no se puede hablar de interoperabilidad entre sistemas informáticos sin tener en cuenta la estandarización de los documentos y los datos que intercambian dichos sistemas. A medida que el mundo avanza hacia la informatización en todas las esferas, el peso de la documentación electrónica como equivalente legal de la verbal o escrita, se hace cada vez mayor. Esto acarrea conflictos con la visión tradicional de la documentación, y requiere cierta voluntad jurídica para su cumplimiento.

En Cuba, a pesar de los esfuerzos, aún resulta insuficiente el marco legal y técnico o tecnológico, lo cual hace impracticable hasta el momento el establecimiento de entornos colaborativos para el intercambio de datos, requisito este indispensable para una plataforma de interoperabilidad. A la hora de establecer esquemas sobre la forma y contenido de los datos emitidos, cada entidad lo hace de manera independiente, sin que exista una institución que evalúe la validez de dicho esquema. Al mismo tiempo, las entidades que precisan conocer el esquema establecido, deben remitirse para ello a la propia entidad que lo emite, sin que haya procedimientos legales que regulen estos procesos. Con la introducción de un sistema como el que se pretende desarrollar en este trabajo, este panorama cambiaría: una institución centralizada y avalada legalmente puede actuar como elemento regulador y facilitador tanto del proceso de establecimiento de esquemas de datos como del acceso a los mismos.

Este capítulo versa sobre los conceptos fundamentales que se relacionan con el tema y que argumentan el desarrollo del trabajo. También incluye un estudio sobre los sistemas adoptados por otras naciones en sus plataformas de eGobierno. Además se describen las tecnologías y herramientas utilizadas en la implementación del sistema propuesto.

1.2. Conceptos importantes

1.2.1. Interoperabilidad.

Según el EIFv2 (European Interoperability Framework versión 2), "*la interoperabilidad en el contexto de los sistemas informáticos es la habilidad de los sistemas TIC, y de los procesos de negocios que ellas soportan, de intercambiar datos y posibilitar el compartimiento de información y conocimientos."* (European

Commission, 2010) Un framework de interoperabilidad puede ser definido como un conjunto de estándares y directrices que describen la forma por la cual organizaciones han acordado, o pueden acordar, interactuar unas con las otras.

1.2.2. Gobernanza de Interoperabilidad.

La gobernanza de interoperabilidad, según la Comisión Europea y la European Public Administration Network (EUPAN), "*se ocupa de las condiciones políticas, legales y estructurales que sean relevantes para el desarrollo y utilización de aplicaciones interoperables*", así como "*de la coordinación y alineación de los procesos de negocios y arquitecturas de información que traspasan los límites intra y extra organizacionales. Su propósito es identificar y enderezar / remover cualquier posible barrera, incluyendo las legislativas, culturales y otras, buscando añadir servicios y compartir información*".

1.2.3. Interoperabilidad semántica.

Se basa en que el significado preciso de la información intercambiada es inteligible por cualquier otra aplicación que no haya sido desarrollada para este propósito. Posibilita a los sistemas combinar la información recibida con otros recursos de información y procesarla de una manera con significado (European Commission, 2010).

1.2.4. Interoperabilidad organizacional.

La interoperabilidad organizacional se ocupa de "*definir procesos de negocios y hacer acontecer la colaboración entre administraciones que desean intercambiar información y que pueden tener diferentes estructuras y procesos internos, así como aspectos relacionados a los requerimientos de los usuarios de la comunidad*" (European Commission, 2010).

1.2.5. Interoperabilidad técnica.

La interoperabilidad técnica es la encargada de cubrir las cuestiones técnicas de conectar sistemas computacionales y servicios (European Commission, 2010).

1.2.6. Framework de Interoperabilidad Gubernamental.

Conjunto de estándares y guías que utiliza un gobierno para especificar la forma recomendada en que sus agencias, ciudadanos, y miembros interactúan entre sí (Bloomberg, y otros, 2006). Incluye "las especificaciones técnicas básicas que todas las agencias relevantes para la implementación de la estrategia de eGobierno deben adoptar" (Interoperability frameworks and enterprise architectures in e-government initiatives in Europe and the United States, 2007).

1.2.7. Estándar abierto.

Un estándar abierto es un estándar que se encuentra disponible públicamente y que tiene varios derechos de uso asociados, y puede incluir propiedades sobre cómo fue diseñado. Los estándares abiertos proveen apertura tanto en el proceso de adopción como en el acceso a las especificaciones. Estos elementos marcan la diferencia con estándares propietarios. Bruce Perens define las siguientes características:

- **Disponibilidad:** Los estándares abiertos se encuentran disponibles públicamente para la implementación y lectura.
- **Maximizar la libertad de elección del usuario final:** Crean un mercado justo y competitivo para las implementaciones del estándar. No atan al usuario a un desarrollador o grupo específico.
- **Sin realeza:** Los estándares abiertos son libres para la implementación sin realeza o costo. La certificación de cumplimiento de la organización que provee el estándar puede incluir un cobro.
- **Sin discriminación:** Los estándares abiertos y las organizaciones que los administran no favorecen a una implementación sobre otra por cualquier razón que no sea el cumplimiento de los estándares técnicos por el desarrollador. Las organizaciones de certificación deben proveer vías para que las implementaciones de bajo o ningún costo sean validadas, pero también pueden proveer servicios de certificación avanzados.
- **Extensión o subconjunto:** Las implementaciones de estándares abiertos pueden ser extendidas u ofrecidas como subconjunto. Sin embargo, las organizaciones certificantes pueden declinar certificar subconjuntos de los estándares, y pueden establecer requerimientos sobre las extensiones (ver prácticas predatorias).
- **Prácticas predatorias:** Los estándares abiertos pueden emplear términos de licencia que protegen contra la desnaturalización del estándar por tácticas de adoptar-y-extender. Las licencias adjuntas al estándar pueden requerir la publicación de información de referencia para las

extensiones, y una licencia para el resto para crear, distribuir y vender software compatible con las extensiones (Perens, 2007).

En cualquier otro caso, un estándar abierto no puede prohibir extensiones.

1.3. Estado del arte.

1.3.1. Interoperabilidad en el Gobierno Electrónico.

El uso de las tecnologías para automatizar el servicio público puede ahorrarle a cualquier país una cantidad significativa de dinero y a la vez permitirles a los ciudadanos acceder más fácilmente a los servicios disponibles. Muy frecuentemente, sin embargo, ese progreso se obstaculiza por dificultades relacionadas con la implementación del mismo.

La interoperabilidad entre sistemas de eGobierno permite servicios online abarcadores y en “ventanilla única” tanto para los ciudadanos como para los negocios, mediante el vínculo de los diversos servicios que ofrecen las diferentes agencias. Además, incrementando la facilidad con la que la información se comparte entre agencias (dentro del marco permitido por la ley) mejora los servicios y crea otros nuevos. Por ejemplo, la administración de Justicia sería más rápida y más efectiva si los sistemas de información de varias agencias del sistema judicial (policía, fiscalía, cortes, prisiones) fueran capaces de compartir información.

Incluso gobiernos que no están automatizados por completo y apenas comienzan a digitalizar datos (como el nuestro) deberían tener en cuenta la interoperabilidad. En ese caso, establecer estándares y planear la arquitectura antes de la digitalización a gran escala sería una manera de evitar los problemas descritos anteriormente y de preparar el camino para un gobierno más eficiente a través del uso de las TIC.

1.3.1.1. Estándares abiertos

Como se expresó anteriormente, los estándares, particularmente los estándares abiertos, juegan un rol fundamental en la obtención de interoperabilidad. Los estándares abiertos permiten a los productos trabajar juntos. También propician la diversidad de proveedores/desarrolladores y el avance tecnológico. Además aseguran la calidad. Es comúnmente aceptado que los estándares abiertos deben constituir el

núcleo de los estándares que los gobiernos adopten para lograr la interoperabilidad de GE. Al respecto Bob Sutor expone:

"Para que la interoperabilidad funcione, se necesita todo lo necesario en aras de comprender e implementar por completo el procesamiento de la información. Por ende, si se obtiene la información e incluye algo proveniente de otro sitio, entonces se requiere acceso total a todo lo necesario para manejar esos datos extra. Esto tiene implicaciones en la apertura e independencia de los obstáculos legales que supone la propiedad intelectual de todos los formatos de intercambio implicados" (Sutor, 2007).

1.3.1.2. Software Código Abierto (OSS)

Estándar abierto no es sinónimo de código abierto. El primero es un conjunto de especificaciones, el otro es una implementación. Lo que ambos comparten es el compromiso con la "apertura" –ausencia de control por un individuo, grupo o corporación, y oportunidades iguales de participación para todos.

El libro blanco de Interoperabilidad de la Asociación Europea de la Industria de las Tecnologías de la Información y las Comunicaciones (EICTA) enumera la forma en que el OSS beneficia la Interoperabilidad:

- La ventaja del OSS es que la apertura del código fuente permite a cualquier usuario modificarlo para asegurar el cumplimiento de los estándares abiertos de interoperabilidad.
- Una fuente natural de desarrolladores de código abierto son los estándares abiertos, los cuales son "nativamente" implementados en el OSS. El resultado es un soporte de facto de los estándares abiertos en el OSS.
- Ciertos ejemplos bien conocidos de licencias código abierto permiten la distribución y el uso de software sin restricción. Este efecto de red es capaz de acelerar la propagación del uso de los estándares y por tanto puede ser un factor contribuyente para una mejor interoperabilidad.
- La accesibilidad del código fuente y el diseño de la información así como los derechos de modificar, extender, y distribuir OSS, soporta la reusabilidad de buenas implementaciones.
- El OSS aumenta la confianza en la interoperabilidad mediante la transparencia. Cuando el código fuente y el compilador son accesibles, los usuarios son capaces de verificar que el software interopera como debería y las organizaciones tienen una solución cuya seguridad, privacidad y transparencia no depende de las acciones o el soporte continuo de los proveedores.

- El modelo de derechos del código abierto soporta la portabilidad de la plataforma: la adaptabilidad de una función a diferentes sistemas operativos o elementos de otras plataformas. Esto puede permitir una amplia diseminación en muchas plataformas resultando en un amplio despliegue de implementaciones interoperables (EICTA, 2007).

1.3.1.3. Frameworks de Interoperabilidad de eGobierno (GIF)

Basándose en el análisis de los GIF de varios países se puede concluir que típicamente incluyen:

- **Contexto:** Puede incluir definiciones, metas, objetivos, principios, trasfondo, audiencia, beneficios y relaciones con otras iniciativas, como el caso de los GIF del Reino Unido (RU).Dinamarca, muy similarmente, tiene tres secciones: Principios, actores, enfoque y trasfondo.
- **Contenido técnico:** Puede incluir: políticas técnicas claves, estándares, categorías de estándares, criterios de selección de estándares y estado de los estándares.
- **Proceso de desarrollo:** Puede incluir proceso de desarrollo y revisión, actores y responsabilidades, así como mecanismos de consulta, como el caso del GIF de Nueva Zelanda.
- **Implementación:** Puede incluir herramientas de soporte a la implementación como los GIF de Dinamarca y el RU.
- **Regímenes de cumplimiento:** Puede incluir indicadores de interoperabilidad, responsabilidad de cumplimiento, stakeholders, herramientas de guía y no cumplimiento, como el GIF del RU.

No todos los GIF abarcan todos los aspectos de la interoperabilidad mencionados antes en este informe. Las iniciativas de Australia, Brasil, Dinamarca, Malasia, Nueva Zelanda y RU solo enfrentan la interoperabilidad técnica, probablemente debido a que es la más fácil de obtener. En cambio los framework de la Unión Europea y de Alemania, que agrupan los estándares según los servicios, (por ejemplo, búsqueda de empleo, impuesto de declaración de la renta, matrícula en Universidades) incluyen todos los niveles de interoperabilidad.

1.3.2. Sistema Administrador de Esquemas y Metadatos de Chile.

Es el organismo público encargado de administrar y mantener operativo, el procedimiento de inscripción de esquemas basales y documentales por parte de los Órganos de la Administración del Estado, su

evaluación técnica y posterior publicación en el Repositorio de Acceso Público de Esquemas de Gobierno, de acuerdo con lo señalado en el Decreto Supremo Nº 271 de 2008.

El Administrador gestiona:

- Un sitio Web de dominio público.
- Una sección del sitio Web donde los usuarios de los órganos del Estado entran con nombre de usuario y contraseña.
- Un repositorio de esquemas.
- Un foro para comentar esquemas.
- Guía de Desarrollo y uso de esquemas de Gobierno.
- Otros documentos de apoyo.

Constituye la solución técnica al proceso de estandarización para la gestión de documentos electrónicos en los ámbitos público y privado. Este sistema permite el registro, administración y publicación de todos los esquemas XML que definen la estructura de aquellos documentos que se intercambian entre sistemas informáticos en la plataforma de Gobierno Electrónico de Chile.

El 13 de enero del 2009 se publica en el Diario Oficial, el Decreto Supremo N° 271 de 2008, en el que se aprueba el reglamento sobre la Inscripción de Esquemas Documentales en el Repositorio del Repositorio de Esquemas y Metadatos para los Órganos de la Administración del Estado (Ministerio de Economía y Fomento, 2009).

El Repositorio de Esquemas y Metadatos tiene por objeto regular el procedimiento de inscripción de esquemas XML necesarios para la definición de estándares de intercambio de datos entre Organismos Públicos. Los esquemas inscritos estarán sujetos a una evaluación técnica y posteriormente serán publicados en este mismo sitio.

Es un sistema sencillo desde el punto de vista de la implementación. Su interfaz es simple y fácil de usar. Introduciendo criterios de búsqueda se puede acceder a los esquemas registrados así como a la información asociada al mismo. En definitiva cumple su propósito satisfactoriamente.

1.3.3. Portal del lenguaje común de intercambio de información de Colombia.

Es el estándar definido por el Estado Colombiano para intercambiar información entre organizaciones, facilitando el entendimiento de los involucrados en los procesos de intercambio de información, el mismo contempla:

- Definición estructural y funcional, de un organismo responsable de la administración y gestión del estándar.
- Definición de una arquitectura de datos, que soporte funcional y conceptualmente la definición, documentación, adopción y uso de los diferentes conceptos susceptibles de ser intercambiados.
- Definición de un proceso de mantenimiento y evolución del estándar que abarque tanto la especificación funcional del mismo, como la técnica.

Para organizar los elementos de dato, el Lenguaje común de intercambio de información cuenta con una arquitectura de datos, basada en capas. Cada una de las capas incluye los elementos de dato que tienen el ámbito de aplicación de la capa.

Los ámbitos de aplicación de las capas de la arquitectura son:

- **Uso Común:** conceptos que tienen la misma interpretación o significado, independiente de la localización geográfica o proceso de intercambio de información en el que se utilice.
- **Uso Local:** conceptos que tienen la misma interpretación o significado, independiente de la entidad, trámite, servicio o proceso de intercambio de información en el que se utilice, a nivel nacional.
- **Uso Sectorial:** conceptos que tienen una interpretación particular, o son de uso específico, en un sector.
- **Uso Proyectos:** conceptos que son de uso específico en un determinado proyecto de una organización.
- **Uso Servicios de la Plataforma de Interoperabilidad – PDI:** conceptos que son de uso específico de la Plataforma de Interoperabilidad.
- **Uso Internacional:** Concepto que tiene elementos de datos que hacen parte o se encuentran definidos en estándares internacionales y que cubren las necesidades del Lenguaje común de intercambio de información.

El portal ofrece diversos servicios relacionados con el enfoque colombiano del lenguaje común de intercambio de información. Las entidades pueden solicitar el servicio de certificación de cumplimiento del estándar en uno de dos niveles, según los criterios:

- Si su necesidad de uso del estándar está enmarcada en la conceptualización de elementos de dato, el tipo de solicitud es: **Nivel 1**.
- Si ya desarrolló su necesidad de intercambio de información y desea evaluar el uso del estándar en un servicio de intercambio de información, el tipo de solicitud es: **Nivel 2** (Ministerio de Tecnologías de la Información y las Comunicaciones, Colombia, 2011).

1.3.4. Resultado del estudio.

Habiendo analizado las iniciativas chilena y colombiana, existen elementos interesantes a tener en cuenta para la implementación de la solución cubana. La simplicidad de la interfaz del sistema AEM chileno, la facilidad en el acceso a los esquemas y la ligereza del sitio, se encuentran entre los más importantes.

1.4. Metodología adoptada para el desarrollo.

Se ha decidido adoptar la metodología definida en el Expediente de Proyectos de la UCI, RUP (Rational Unified Process). Entre las razones para la selección se encuentra el importante hecho de que el autor está familiarizado con la metodología. Otro elemento es la abundante bibliografía asociada a esta, así como la robustez y consistencia que la caracteriza.

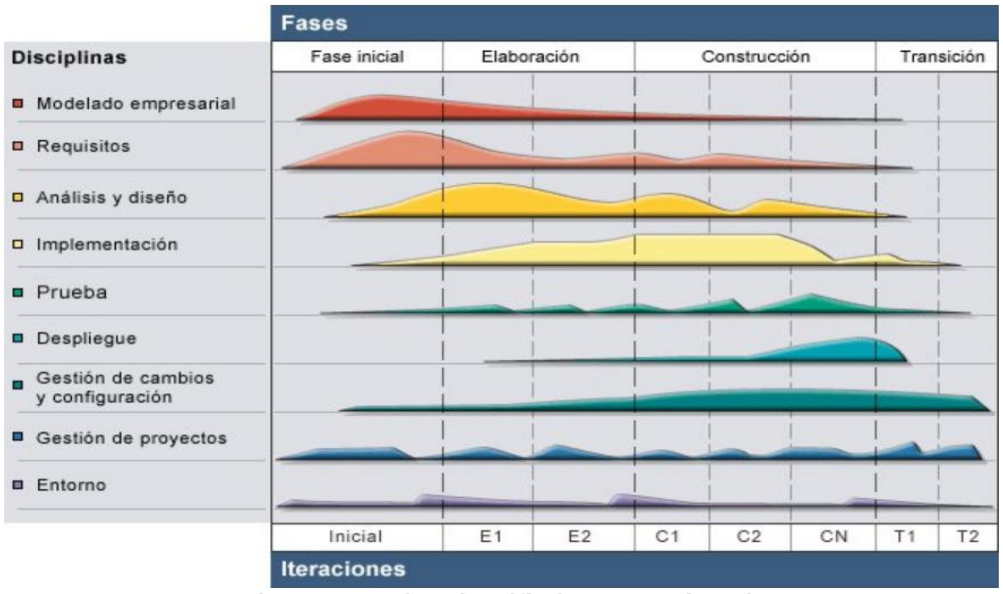

Figura 1: Rational Unified Process (RUP)

Sin embargo, teniendo en cuenta que RUP es una metodología pesada, el tiempo disponible para el desarrollo, así como la relativamente baja complejidad del proyecto, es preciso adaptarla a las necesidades propias del sistema a desarrollar. Es decir, enfocarla en una perspectiva que tenga en cuenta las características de tiempo y complejidad que marcan el entorno en que se desarrolla el proyecto. *"Más proceso, como el uso de más artefactos, la producción de documentación más detallada, el desarrollo y mantenimiento de más modelos que deben sincronizarse y más revisiones formales no es necesariamente una solución mejor. Por el contrario, se debe ajustar el tamaño del proceso a las necesidades del proyecto"* (IBM Corporation, 2007).

Al efecto RUP provee una serie de recomendaciones y guías para tener en cuenta cuando el tamaño del proyecto que se desea desarrollar no justifica la elaboración de una documentación demasiado compleja.

La directriz "Prácticas del proceso de personalización" se describe cómo personalizar RUP teniendo en cuenta lo siguiente (IBM Corporation, 2007):

- **No incluir tareas y productos de trabajo que no se puedan justificar claramente**
- **Proteger a los desarrolladores del proceso**
- **Minimizar los productos de trabajo intermedios formales**
- **Utilizar formatos cómodos**

- **Generar cuando sea posible**
- **Utilizar la web**
- **Utilizar herramientas integradas**
- **Revisar regularmente el proceso**
- **Personalizar manteniendo las prácticas**

RUP provee un adecuado nivel de compatibilidad con metodologías ágiles, como XP, pudiendo solaparse roles y artefactos entre ambas metodologías, por lo cual su utilización desde un punto de vista menos pesado y más orientado a la funcionalidad es posible.

1.5. Tecnologías Utilizadas.

En el año 2010 fue emitido un informe sobre las tecnologías utilizadas en producción en la UCI, titulado Informe Tecnológico de la Producción (Arcia Carrazana, y otros, 2010). En este documento se realiza un estudio a fondo sobre el tema, y realiza evaluaciones y recomendaciones muy interesantes. Dichas recomendaciones fueron tenidas en cuenta para la elaboración de este epígrafe. Al igual que en el tema anterior, la experiencia por parte del autor en muchas de las herramientas y lenguajes fue un factor determinante en el proceso de selección. De esa forma se buscó evitar el costo en tiempo y recursos que conlleva familiarizarse con herramientas y lenguajes que posean una curva de aprendizaje demasiado lenta.

1.5.1. XML.

Es un metalenguaje (lenguaje que describe los datos y cómo estos se estructuran) mediante el cual los desarrolladores pueden crear sus propios elementos para satisfacer sus propias necesidades de información (W3C). XML es el estándar escogido para el intercambio de datos entre sistemas en la definición de la arquitectura de la plataforma de interoperabilidad para Cuba. Entre sus ventajas se encuentra que es un estándar abierto, y que su uso está muy difundido.

1.5.2. Herramienta CASE.

La tecnología CASE supone la automatización del desarrollo del software, de la documentación, la generación de código, el chequeo de errores y la gestión de proyecto, contribuyendo a mejorar la calidad y

la productividad en el desarrollo de sistemas de información. Entre las ventajas de su utilización se encuentra que permiten aumentar la portabilidad y la reutilización del software, así como la estandarización de la documentación.

1.5.2.1. Visual Paradigm.

Visual Paradigm Suite es una librería de herramientas de modelado CASE creada por la compañía Visual Paradigm, con sede en Hong Kong, China (Visual Paradigm International, 2011). Es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software, análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Entre sus funcionalidades se encuentran:

- Soporte de UML versión 2.1.
- Diagramas de Procesos de Negocio.
- Modelado colaborativo con CVS y Subversion (nueva característica).
- Interoperabilidad con modelos UML2 (meta modelos UML 2.x para plataforma Eclipse) a través de XMI (nueva característica).
- Ingeniería inversa - Código a modelo, código a diagrama. Ingeniería inversa Java, C++, Esquemas XML, XML, .NET exe/dll, CORBA IDL, PHP.
- Generación de código - Modelo a código, diagrama a código.
- Editor de Detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Diagramas de flujo de datos.
- Soporte ORM - Generación de objetos Java desde la base de datos.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.
- Generador de informes para generación de documentación.

- Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.
- Se integra con IDEs como NetBeans, Eclipse y Visual Studio.

1.5.2.2. Enterprise Architect.

Enterprise Architect es una herramienta de modelado basada en UML 2.1 flexible, completa y poderosa para plataformas Windows. Es orientada a objetos y provee amplias funcionalidades para el desarrollo de sistemas, administración de proyectos y análisis de negocios. Ayuda al desarrollador de software a trazar especificaciones de alto nivel a modelos de análisis, diseño, implementación, pruebas y mantenimiento, usando UML, SysML, BPMN y otros estándares abiertos para modelado. Soporta un impresionante rango de lenguajes de desarrollo, incluyendo Action Script, C, C++, C# y VB.NET, Java, Visual Basic 6, Python, PHP, XSD y WSDL, entre otros. Entre sus principales características se pueden destacar (SPARX Systems, 2011):

- Con un repositorio de alto desempeño, facilita que los grandes equipos compartan la misma visión del desarrollo del producto.
- Provee trazabilidad completa desde los modelos de requisitos, análisis y diseño, hasta la implementación y despliegue.
- Generación potente de documentación.
- Generación e ingeniería inversa de código fuente. Soporta la generación e ingeniería inversa de código fuente para muchos lenguajes populares. Las integraciones para Eclipse y Visual Studio .NET, les proveen a los desarrolladores un acceso directo a los diseños y capacidades de modelado justo dentro del IDE. Las plantillas de generación de código le permiten personalizar el código fuente generado de acuerdo con las especificaciones de su compañía.
- Permite que se construya, pruebe, depure, active y ejecuten scripts de despliegue, todo dentro del entorno de desarrollo de Enterprise Architect. Con la capacidad de generar clases de prueba NUnit y JUnit desde clases origen usando transformaciones MDA y la integración del proceso de pruebas directamente en el IDE de Enterprise Architect, ahora puede integrar el modelado con UML y el proceso de compilación/prueba/ejecución/despliegue.
- Se integra con IDEs como Eclipse y Visual Studio.

1.5.2.3. Selección de la Herramienta CASE.

Habiendo analizado lo anterior, y teniendo en cuenta las innegables ventajas de Enterprise Architect como herramienta, se decidió optar por el uso de Visual Paradigm Suite 3.4. Entre las razones de su selección se encuentran:

- Es multiplataforma, por lo que se puede emplear tanto en sistemas operativos Windows como Linux.
- Se integra con el IDE NetBeans.
- La UCI posee una licencia para su uso.
- El autor tiene experiencia en el trabajo con la herramienta.

1.5.3. Lenguaje de modelado.

El Lenguaje Unificado de Modelado (Unified Modelling Language, UML) fue creado por Jim Rumbaugh, Ivar Jacobson y Grady Booch. Es un lenguaje de modelado visual multipropósito usado para especificar, visualizar, construir y documentar los artefactos de un sistema de software. Captura las decisiones y la comprensión de los sistemas a construir, por lo que comprende, diseña, analiza, configura, mantiene y controla información de tales sistemas (Rumbaugh, y otros, 1998). Se le considera un estándar en la industria del software.

1.5.4. Gestor de base de datos.

Un Sistema Gestor de Base de Datos es un conjunto de programas que permiten crear y mantener una BD asegurando su integridad, confidencialidad y seguridad.

1.5.4.1. MySQL.

La historia del MySQL se remite a principios de la década de 1980 cuando programadores de IBM lo desarrollaron para contar con un código de programación que permitiera generar múltiples BD para empresas y organizaciones de diferentes tipos.

Una de las características de MySQL es que permite recurrir a BD multiusuario a través de la web y en diferentes lenguajes de programación que se adaptan a diferentes necesidades y requisitos. Por otro lado, es notorio por desarrollar altas velocidades en la búsqueda de datos e información.

1.5.4.2. PostgreSQL.

PostgreSQL es un potente sistema de base de datos relacional libre (su código fuente está disponible) liberado bajo licencia Berkeley Software Distribution (BSD). Fue desarrollado en la Universidad de California, en el departamento de Ciencias de la Computación. Posee más de 15 años de activo desarrollo y arquitectura probada que se ha ganado una muy buena reputación por su confiabilidad e integridad de datos. Funciona en los principales sistemas operativos, incluyendo las distribuciones de GNU/Linux, las variantes de UNIX y Windows (PostgreSQL Global Development Group , 2011). Es el SGBD de código abierto más potente del mercado y en sus últimas versiones está a la par de otras soluciones comerciales.

Entre sus características más importantes se encuentran:

- Es una base de datos segura.
- Tiene integridad referencial.
- Tiene múltiples métodos de autentificación.
- Acceso encriptado vía SSL.
- Provee actualización integrada momentánea.
- Posee documentación completa.

1.5.4.3. Selección del SGBD.

Por ser un SGBD open source, con amplia documentación y robusto, se ha decidido emplear PostgreSQL. Es importante tener en cuenta que la UCI pertenece a la Comunidad Iberoamericana de PostgreSQL.

1.5.5. Lenguaje de programación.

1.5.5.1. PHP.

PHP es un acrónimo recursivo que significa Hypertext Pre-processor (inicialmente PHP Tools, o, Personal Home Page Tools) es un lenguaje open source interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor. Fue escrito originalmente por Rasmus Lerdorf (The PHP Group, 2011). Se utiliza para la generación de páginas web dinámicas, embebidas en páginas HTML y ejecutadas en el servidor. Para su funcionamiento necesita tener instalado Apache o IIS con las librerías de PHP. La mayor parte de su sintaxis ha sido tomada de C, Java y Perl con algunas características específicas. Es un lenguaje multiplataforma, fácil de aprender y no requiere definición de tipos de variables, siendo estas sus mayores ventajas. Entre sus desventajas se encuentra la pobre programación orientada a objetos (aunque a partir

de PHP 5 esto se ha revertido en cierto grado), además de dificultar la organización por capas de la aplicación.

- **Rendimiento:** El motor de PHP 5 fue rediseñado completamente con un administrador de memoria optimizado para mejorar el rendimiento.
- **Portabilidad**: PHP está disponible para sistemas operativos como UNIX, Microsoft Windows y Mac OS. Los programas escritos en este lenguaje son portables entre plataformas.
- **Facilidad de uso:** Su sintaxis es clara y consistente, cuenta además con documentación exhaustiva para todas las funciones de su núcleo.
- **Código Abierto:** El lenguaje es desarrollado por una comunidad de programadores voluntarios que publican su código libremente en la Web y puede ser usado sin el pago de licencias o inversiones en hardware costoso. Esto reduce el costo de la producción del software sin afectar la flexibilidad o confiabilidad.
- **Soporte comunitario:** Cuenta con amplio soporte gracias a la numerosa comunidad de programadores que lo usan en todo el mundo.

1.5.5.2. Java.

Java es un lenguaje de programación orientado a objetos, de propósito general, que presenta características especiales que lo hacen idóneo para su uso en Internet (Mesa, 2005):

- **Orientado a objetos:** Java organiza sus programas en una colección de objetos, lo que permite estructurar los programas de una manera más eficiente y en un formato fácil de comprender.
- **Distribuido:** Java dispone de una serie de librerías para que los programas se puedan ejecutar en varias máquinas y puedan interactuar entre sí.
- **Robusto:** Java está diseñado para crear software altamente fiable.
- **Seguro:** Java cuenta con ciertas políticas que evitan que se puedan codificar virus con este lenguaje, sin olvidar que existen otras restricciones que limitan lo que se puede o no hacer con los recursos críticos de una máquina.
- **Interpretado:** la interpretación y ejecución se hace mediante la Máquina Virtual de Java (JVM) que es el entorno en el que se ejecutan los programas Java; su misión principal es la de garantizar la ejecución de las aplicaciones Java en cualquier plataforma.

- **Independiente de la Arquitectura:** el código compilado de Java se puede usar en cualquier plataforma.
- **Multiejecución:** Java permite elaborar programas que permitan ejecutar varios procesos al mismo tiempo sobre la misma máquina.

1.5.5.3. Selección del lenguaje de programación.

A pesar de que Java es un lenguaje más completo, con mayor soporte del paradigma de programación orientada a objetos, se decidió utilizar PHP 5.3 por las siguientes razones:

- Es muy fácil de utilizar, de sintaxis sencilla.
- Existe un buen número de frameworks para PHP que simplifican el desarrollo de aplicaciones Web.
- El autor está familiarizado con el lenguaje.

1.5.6. Framework de desarrollo.

"Un framework simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, un framework proporciona estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener. Por último, un framework facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas" (Potencier, y otros, 2007).

1.5.6.1. Zend Framework.

Es un framework de código abierto para desarrollar aplicaciones y servicios Web con PHP 5. Se ejecuta utilizando un 100% de código orientado a objetos. Cada uno de sus componentes ha sido diseñado con independencia de los otros. Esta flexibilidad permite a los desarrolladores de la arquitectura utilizar los componentes individualmente. Ofrece una alta capacidad y robustez para la implementación del patrón Modelo Vista Controlador.

1.5.6.2. Symfony.

Symfony es un completo framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Separa la lógica de negocio, la lógica de servidor y la presentación de la

aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada producto. El resultado de todas estas ventajas es que evita "reinventar la rueda" cada vez que se crea una nueva aplicación web (Potencier, 2010).

1.5.6.3. Selección del framework de desarrollo.

Por ser más ligero, recomendable para proyectos pequeños, sin que eso prejuicie su escalabilidad, por ser más fácil de aprender, y por tener una amplia documentación, se ha decidido utilizar Symfony como framework de desarrollo. Gracias a la fácil integración de componentes externos en Symfony, se puede utilizar el motor de búsqueda Zend Lucene en el proyecto, el cual es provisto por Zend Framework.

1.5.7. Entorno de desarrollo.

Un entorno de desarrollo integrado (IDE) proporciona un marco de trabajo amigable para gran cantidad de lenguajes de programación. Además es posible que un mismo entorno de desarrollo integre varios lenguajes de programación.

1.5.7.1. NetBeans.

NetBeans es una plataforma de programación desarrollada por Oracle. La arquitectura de plugins de NetBeans permite, además, integrar diversos lenguajes sobre un mismo IDE (Oracle Corporation, 2011). Es multiplataforma, tiene una interfaz muy amigable e intuitiva, posee todas las herramientas para crear aplicaciones profesionales ya sean de escritorio, empresariales, web, móviles y aplicaciones SOA, no solo en Java o PHP sino también en C/C++ y Ruby. Gracias a la integración con Visual Paradigm, se pueden realizar diagramas UML dentro del mismo IDE. Otro elemento importante es que integra frameworks de PHP como Symfony y Zend.

1.5.7.2. Eclipse.

Es un IDE de código abierto que fue desarrollado originalmente por IBM Canadá. Actualmente es publicado por la Fundación Eclipse. Posee una arquitectura de plugins que le permite a los

desarrolladores agregar diversas funcionalidades. Proporciona soporte para una amplia gama de lenguajes de programación, así como una interfaz amigable y altamente configurable.

1.5.7.3. Selección del entorno de desarrollo.

Existen muchas similitudes entre los dos IDEs analizados. Ambos cuentan con amplio soporte y herramientas que facilitan el desarrollo. Se ha decidido emplear NetBeans por su integración con el CASE seleccionado y el framework Symfony.

1.6. Pruebas a realizar.

"El modelo de pruebas describe principalmente cómo se prueban los componentes ejecutables (como las construcciones) en el modelo de implementación con pruebas de integración y de sistema (…) puede describir también cómo han de ser probados aspectos específicos del sistema; por ejemplo, si la interfaz de usuario es utilizable y consistente" (Jacobson, y otros, 2000).

Existen dos métodos de pruebas fundamentales para probar la calidad de un producto de software:

- **Pruebas de caja negra:** Conociendo la funcionalidad específica para la cual fue diseñado el producto, se pueden llevar a cabo pruebas que demuestren que cada función es completamente operativa.
- **Pruebas de caja blanca:** Conociendo el funcionamiento del producto se pueden desarrollar pruebas que aseguren que "todas las piezas encajen", o sea, que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada.

Las pruebas de caja negra se centran fundamentalmente en los requisitos funcionales del software. Por esta razón se decide utilizar este tipo de prueba en la validación del sistema, quedando el resto como recomendación para otras instancias.

Para realizar pruebas de caja negra, no se precisa saber cómo se implementó la función a probar. Su objetivo es verificar que la misma se comporte de la manera requerida. Se basan en la entrada y la salida permitiendo de esa forma detectar:

- Funciones incorrectas o ausentes.

- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las Bases de Datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Existen varias técnicas de prueba de caja negra, entre ellas se encuentran (Pressman, 2002):

- **Técnica de la Partición de Equivalencia:** esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del *software*.
- **Técnica del Análisis de Valores Límites:** esta Técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- **Técnica de Grafos de Causa-Efecto:** es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

La **Partición de equivalencia** es una técnica para reducir el número de pruebas necesarias. Para cada operación, debe identificar las clases de equivalencia de los argumentos y los estados de objeto (IBM Corporation, 2007). Se dirige a la definición de casos de pruebas que descubran clases de errores, reduciendo así el número de clases de prueba que hay que desarrollar. 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 (Pressman, 2002). Por tanto, se decidió aplicar esta técnica.

1.7. Conclusiones del capítulo

Como resultado del estudio de las definiciones relacionadas se ha logrado obtener una base conceptual que permite un mejor entendimiento de los temas a tratarse posteriormente en el informe en general. El análisis de documentación consultada ha servido para fundamentar la necesidad de que el Órgano de Justicia-MININT cuente con un sistema Repositorio de Esquemas y Metadatos, así como buenas prácticas y elementos postivos a tener en cuenta en el desarrollo de la solucion propuesta. También se han presentado las herramientas que mejor pueden contribuir a la obtención del resultado esperado.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

2.1. Introducción.

En el presente capítulo se describe la solución propuesta. Se hace referencia a las reglas del negocio asociadas al dominio del problema, ofreciendo una visión general del funcionamiento de la entidad. Se describen los requisitos funcionales y no funcionales que debe cumplir el sistema que se propone, se realiza su modelación en términos de casos de uso y los mismos son detallados en sus descripciones. Esto provee una concepción general de la aplicación a desarrollar.

2.2. Modelación del negocio.

“El modelado del negocio es una técnica para comprender los procesos del negocio de la organización” (Jacobson, y otros, 2000).

2.2.1. Descripción de procesos del negocio.

Un proceso de negocio es un conjunto de actividades o tareas estructuradas y relacionadas, que producen un servicio o producto para un cliente determinado.

Procedimiento de registro de esquema.

Cuando una institución requiere del servicio de registro de un esquema inicia el procedimiento de registro, el cual está desglosado en varias tareas:

i. **Solicitud de registro de esquema:** La Comisión de Registro de esquemas recibe una solicitud, la cual contiene la ficha del solicitante legal y el esquema propuesto para el registro.
ii. **Procedimiento de admisibilidad formal:** Se validan los documentos requeridos en el proceso de solicitud. No incluye validación técnica del esquema, solo se limita a los documentos formales para procesar la solicitud.
    a. **Admisibilidad formal:** la Comisión de Registro revisa el cumplimiento de la solicitud con los requerimientos formales, en caso de aceptarla se le notifica a la entidad que la realizó. El esquema pasa a tener entonces un estado de “**revisión**”.
    b. **Inadmisibilidad formal:** si se rechaza la solicitud entonces se le notifica a la entidad implicada, la cual puede entonces volver a realizar la solicitud una vez corregidos los errores.

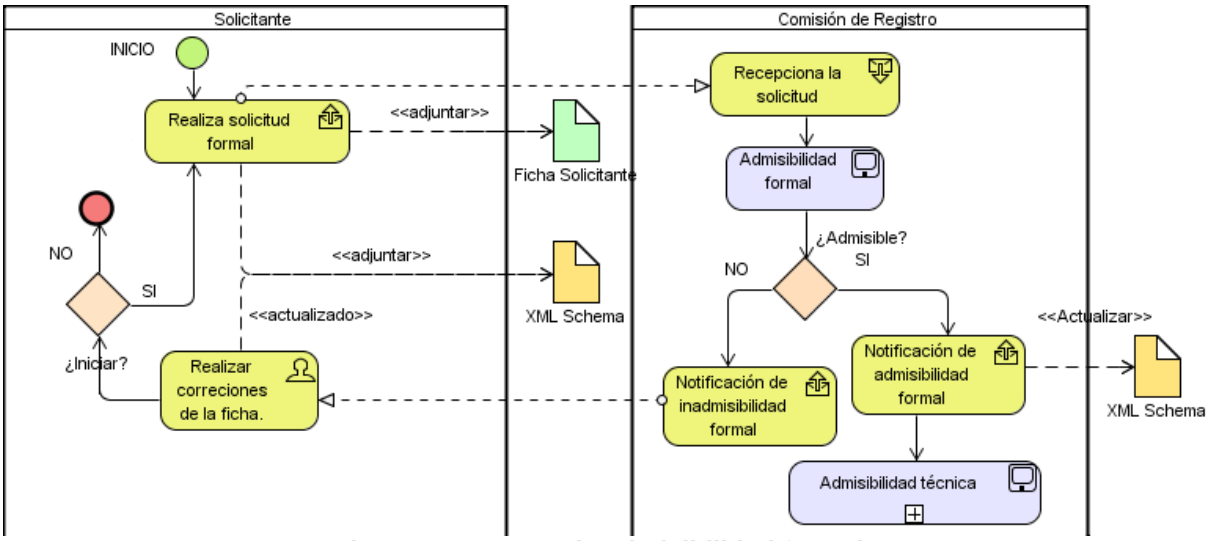

Figura 2: Proceso de admisibilidad formal.

iii. **Procedimiento de admisibilidad técnica:** Se valida el esquema que se pretende registrar desde un punto de vista técnico.

a. Admisibilidad técnica: si la Comisión de Registro considera al esquema como técnicamente válido, se le notifica al solicitante. El esquema pasa al estado "**preliminar**", donde con la participación de los órganos implicados se evalúa la pertinencia de la estructura que define el esquema. Al finalizar el proceso se emite un reporte con las observaciones y recomendaciones realizadas.

b. **Inadmisibilidad técnica:** si el esquema no es válido, a partir de criterios técnicos específicos y el cumplimiento de un nivel de calidad mínimo, se declara la solicitud inadmisible técnicamente y se le notifica al solicitante. Dicha entidad, cumplido un plazo determinado, puede realizar nuevamente la solicitud luego de las correspondientes correcciones.

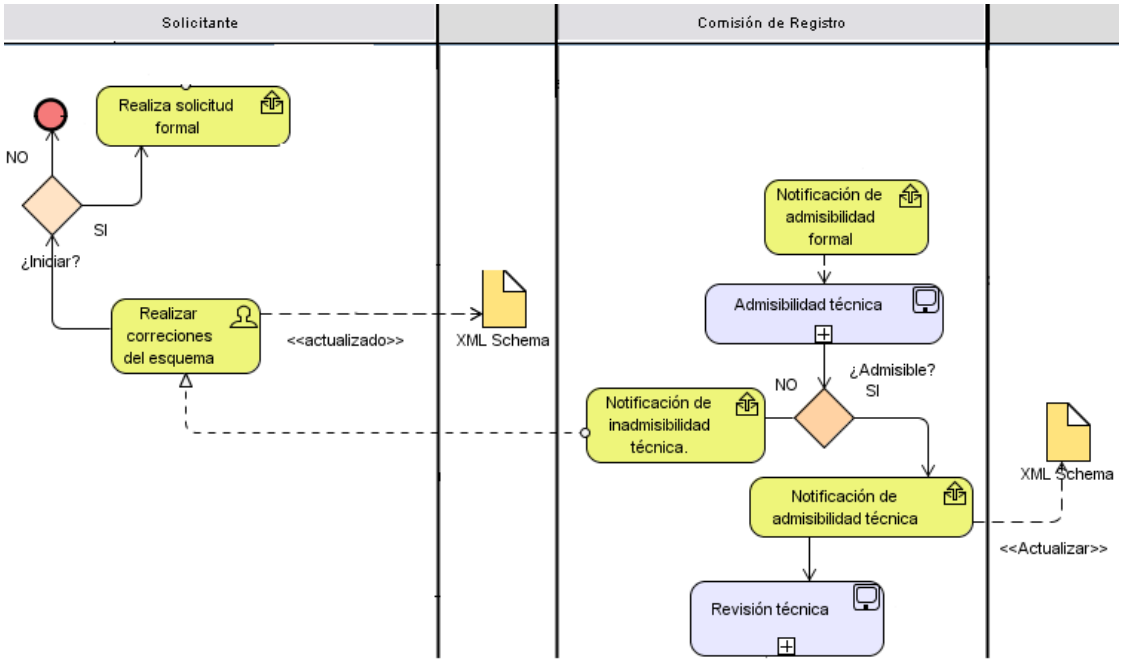

Figura 3: Proceso de admisibilidad técnica.

iv. **Incorporación final del esquema:** cumplidas las recomendaciones realizadas en el paso anterior, el esquema pasa a tener un estado "**consolidado**", incorporándose definitivamente al repositorio.

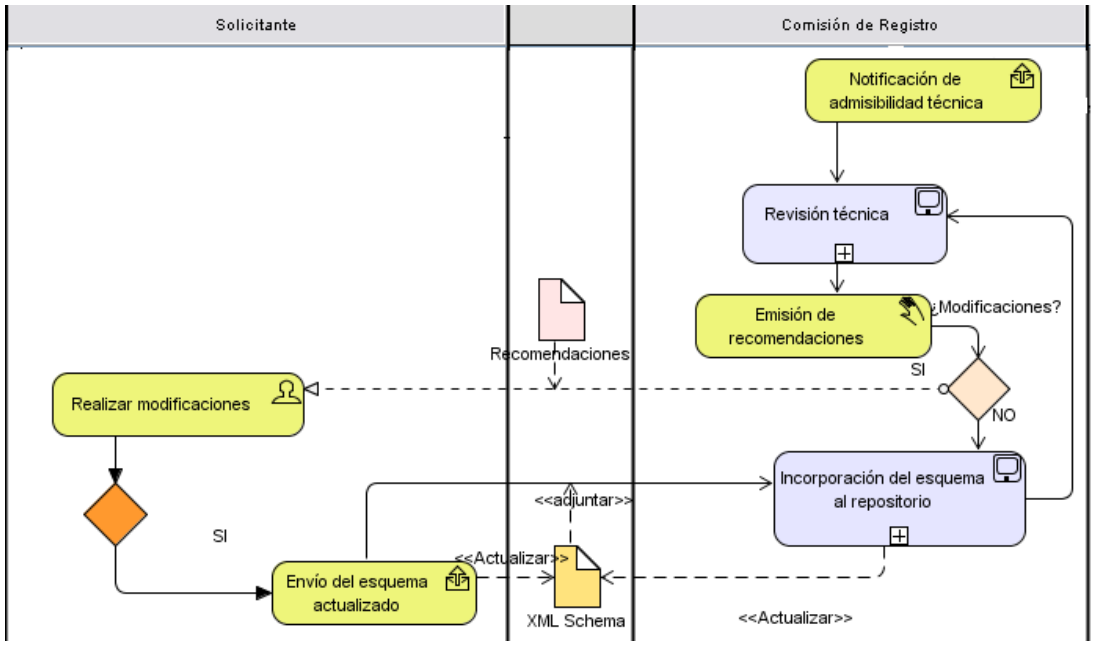

Figura 4: Incorporación del esquema al repositorio.

a. Obsolescencia de esquemas: cualquier cambio o modificación que se le desee realizar a un esquema en estado "**consolidado**", se deberá solicitar a la Comisión de Registro de acuerdo a lo establecido en el procedimiento de solicitud de registro de esquema y cumplir con los requerimientos establecidos para dicho registro. El esquema resultante del nuevo registro no reemplazará al esquema correspondiente en estado "**consolidado**", sino que dará lugar a una nueva versión de dicho esquema, dejando al anterior en un estado de "**desuso**", sin poder ser eliminado del repositorio.

2.2.2. Definición de las Reglas del negocio.

Las reglas de negocio son elementos que describen o enmarcan algunos aspectos del negocio. Describen las reglas, operaciones y definiciones que se aplican a una organización. Pueden afectar a personas, procesos, comportamientos corporativos y sistemas de cómputo.

2.2.2.1. Informaciones básicas que deberán contener los esquemas:

- Título
- Nombre
- Estado
- Fecha de publicación;
- Descripción
- Origen de registro (Organismo del Estado, Órgano de la Administración del Estado o Institución).

2.2.2.2. Condiciones de registro:

a. Deberá tratarse en todo momento de un esquema XML.
b. Cada esquema candidato al registro, no deberá estar contenido en el repositorio en alguno de los estados de uso preliminar, uso consolidado o en trámite de registro.
c. El esquema deberá cumplir al menos con las informaciones básicas especificadas para la definición de esquemas.
d. Cada solicitud de registro deberá realizarse acompañado de los siguientes documentos:
  i. Descripción del esquema y procedimientos de uso.

ii. Al menos una propuesta de plantilla (XSL) de transformación del esquema, que garantice la representación de los documentos XML válidos para este, en XHTML.

iii. Ficha del solicitante.

2.2.2.3. Requerimientos para el retiro de esquemas:

En caso de que un órgano de conformidad con sus competencias y en correspondencia con lo que estipula la ley, requiera el retiro de uso de un terminado esquema registrado en el repositorio, deberá notificarlo formalmente a la Comisión de Registro la cual asignará el estado de desuso.

2.2.2.4. Datos de la Ficha del Solicitante:

- Nombre del Organismo, Órgano o Institución.
- Número de identificación del Organismo, Órgano o Institución (RUT, ID)
- Domicilio del Organismo, Órgano o Institución.
- Nombre e identificación del representante legal que realiza la solicitud.
- Nombre e identificación del especialista técnico responsable de la definición del esquema.
- Dirección o direcciones de correo electrónico del responsable legal y el especialista técnico.

2.2.2.5. Otras reglas:

- Las entidades autorizadas para realizar solicitudes de registros de esquemas y metadatos, deberán estar debidamente certificadas por las autoridades competentes según lo establezca la legislación vigente.
- La Comisión de Registro deberá aceptar o rechazar las solicitudes realizadas al amparo de la legislación vigente al respecto.
- La Comisión de Registro será definida por el organismo competente según lo establezca la ley, y funcionará a partir de la supervisión de dicho organismo y en consecuencia de las normas establecidas.

2.2.3. Identificación de actores y trabajadores

- **Solicitante:** Actor del negocio que representa a las entidades que solicitan el registro de sus esquemas.

- **Interesado:** Representa a las entidades que requieren el uso de los esquemas registrados. A diferencia de las entidades solicitantes, las entidades beneficiadas no precisan estar registradas.
- **Comisión de Registro:** Trabajador del sistema que representa un panel de especialistas encargado de evaluar la aceptación o rechazo de las solicitudes hechas por las entidades; encargados de validar los esquemas propuestos.
- **Organismo Competente (Stakeholder):** Representa al organismo de gobierno que rige el funcionamiento de la Comisión de Registro y establece todas las normas legales para el registro de esquemas.

2.2.4. Casos de uso del negocio (CUN).

"Un modelo del negocio describe los procesos de negocio (…) en términos de casos de uso del negocio y actores del negocio que se corresponden con los procesos del negocio y los clientes, respectivamente" (Jacobson, y otros, 2000).

Se han identificado los siguientes CUN:

- **CUN 1:** Solicitar registro de esquema.
- **CUN 2:** Validar Esquema.
- **CUN 3:** Modificar Esquema.
- **CUN 4:** Consultar Esquema.

2.2.4.1. Diagrama de casos de uso del negocio.

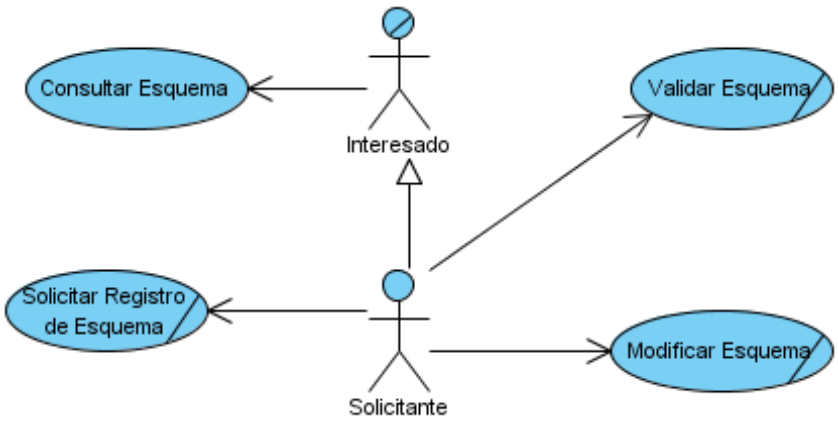

Figura 5: Diagrama de casos de uso de negocio.

2.2.4.2. Descripción de casos de uso del negocio.

En las descripciones de los CUN se detallan los procesos que se ejecutan para una mejor comprensión de los mismos. Se presenta la descripción del CUN 4: “Consultar Esquema” (para la descripción del resto de los CUN ver el documento “Especificación de Casos de Uso”).

Caso de Uso:	**Consultar Esquema**
Actores:	Interesado
Trabajadores:	
Resumen:	La entidad interesada en realizar una consulta de algún esquema inscrito en el repositorio especifica los criterios de búsqueda y selección, obteniendo los esquemas que cumplan con los mismos.
Precondiciones:	El esquema debe encontrarse en estado de uso “Consolidado”
Flujo Normal de Eventos	
Sección “A”	
Acción del Actor	**Respuesta del Negocio**
1. El interesado especifica los criterios de búsqueda.	2. Se presenta una lista con todos los esquemas registrados que cumplan con los criterios especificados. **Ver Flujo Alterno 2.1** (En caso de que no existan esquemas que cumplan con los criterios especificados)
3. Se selecciona el esquema que se desea consultar.	4. Se entrega el esquema seleccionado.
Flujos Alternos	
Acción del Actor	**Respuesta del Negocio**
	2.1. Se informa que no se encontró ningún esquema que cumpla con los datos especificados
2.2. El interesado especifica nuevos criterios de búsqueda y regresa a la acción (1).	
Poscondiciones	

Tabla 2: Descripción del caso de uso del negocio “Consultar Esquema”.

2.2.5. Especificación de requisitos.

Según la IEEE un requerimiento puede definirse como:

"Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente." (IEEE)

2.2.5.1. Requisitos funcionales.

Un requisito funcional *"especifica una acción que debe ser capaz de realizar el sistema"* (Jacobson, y otros, 2000).

Habiendo definido las especificidades del negocio, se procedió a identificar los requisitos funcionales del sistema a modelar. Una vez establecidos, los requisitos fueron descritos y analizados en busca de elementos que pudieran enriquecer lo que se describe, finalmente se mostraron al cliente para obtener su conformidad. En el proceso se utilizaron las técnicas:

Entrevista: es la más utilizada, y es prácticamente inevitable en cualquier desarrollo pues es una de las formas de comunicación más naturales entre personas; estructuradas por preguntar cómo y a quién, favoreciendo el contacto directo y la validación. Las entrevistas no se deben improvisar, por lo que se debe ir con ella redactada para que tenga éxito (Pressman, 2002).

Tormenta de ideas: Reunión de varios interesados en la que todos expresan sus ideas sobre el problema y su solución. La forma de llevarla a cabo es que cada participante diga su idea sin ser interrumpido por otro. Al finalizar la sesión de lluvia de ideas se puede hacer una recolección de ideas sin duplicidad (Pressman, 2002).

Los requisitos funcionales identificados fueron:

- **RF 1: Autenticar usuario.**

El sistema debe permitir que los usuarios puedan identificarse introduciendo su nombre de usuario y su contraseña, pudiendo establecerse así los permisos del usuario específico según el rol que desempeña.

- **RF 2: Buscar esquema.**

El sistema debe permitir la búsqueda de esquemas introduciendo algunos de los siguientes datos:

- Título.
- Órgano.
- Estado.
- Descripción.

- **RF 3: Descargar esquema.**

El sistema debe permitir la descarga de esquemas en estado de uso consolidado.

- **RF 4: Mostrar datos de esquema.**

El sistema debe permitir que se muestren los datos de un esquema determinado.

- **RF 5: Solicitar registro de esquema.**

El sistema debe permitir que los usuarios interesados realicen la solicitud de registro de esquema, introduciendo los datos de la ficha de solicitante y el esquema especificados en la descripción del procedimiento de registro.

- **RF 6: Consultar estado de esquema.**

El sistema debe permitir que el solicitante pueda conocer el estado de los esquemas de las solicitudes realizadas.

- **RF 7: Actualizar estado de solicitud.**

El sistema debe permitir la actualización del estado de los esquemas de las solicitudes:

- Tramitación.
- Revisión.
- Rechazado.
- Preliminar.
- Consolidado.
- Desuso.

- **RF 8: Eliminar solicitud.**

El sistema debe permitir la eliminación de una solicitud que haya sido caracterizada como no válida.

- **RF 9: Modificar esquema.**

El sistema debe permitir la modificación de un esquema.

- **RF 10: Mostrar datos de ficha de solicitante.**

El sistema debe permitir que se muestren los datos de una determinada ficha de solicitante.

- **RF 11: Modificar ficha de solicitante.**

El sistema debe permitir que se puedan editar los datos de una ficha de solicitante.

- **RF 12: Crear usuario.**

El sistema debe permitir la creación de usuarios introduciendo los siguientes datos:

- Nombre.
- Apellido.

- Usuario.
- Correo electrónico.
- Contraseña.
- Permisos.

➢ **RF 13: Editar usuario.**

El sistema debe permitir editar los datos de los usuarios así como su asignación de permisos.

➢ **RF 14: Eliminar usuario.**

El sistema debe permitir la eliminación de un determinado usuario del sistema.

2.2.5.2. Requisitos no funcionales.

Un requisito no funcional "especifica propiedades del sistema" (Jacobson, y otros, 2000).

La aplicación debe cumplir con los siguientes requisitos no funcionales:

2.2.5.2.1. Apariencia o interfaz externa:

- El diseño debe ser sencillo, amigable e intuitivo, permitiendo de esta manera una navegación rápida y fácil por parte del usuario.
- Debe ser adaptable a cualquier tipo de resolución de pantalla.

2.2.5.2.2. Seguridad:

- Identificar al usuario antes de que efectúe cualquier acción.
- Cada usuario contará con diferentes permisos en el sistema en correspondencia con el rol que ocupe.
- Protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.

2.2.5.2.3. Software:

- Para una máquina cliente:
  - Navegador compatible con IE 6, Mozilla Firefox 3.0, o superior.
- Para un servidor:
  - Sistema operativo Linux en cualquiera de sus distribuciones.

- Un servidor Apache 2.2.17 o superior con módulo PHP 5.3.5 o superior disponible.
- Un servidor de base de datos PostgreSQL 8.3 o superior.

2.2.5.2.4. Hardware.

- Para una máquina cliente:
  - Requerimientos mínimos: Procesador Pentium III a 1GHz con 256Mb de memoria RAM.
  - Tarjeta de red.
- Para un servidor:
  - Requerimientos mínimos: Procesador Pentium IV a 2GHz y 1Gb de memoria RAM.
  - Al menos 40Gb de espacio libre en disco duro.
  - Tarjeta de red.

2.3. Modelación del sistema.

2.3.1. Casos de uso del sistema (CUS).

"Cada forma en que los actores usan el sistema representa un caso de uso. Los casos de uso son "fragmentos" de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. De manera más precisa, un caso de uso especifica una secuencia de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de la secuencia" (Jacobson, y otros, 2000).

2.3.1.1. Patrones de Casos de Uso

Las técnicas y diseños exitosos que se utilizan una y otra vez en modelos de casos de uso se formalizan como patrones que expresan buenos modelos de casos de uso. Un patrón de caso de uso capta una técnica para hacer el modelo mantenible, reutilizable y comprensible (Övergaard, y otros, 2004).

- **El nombre revela la intención:** Hace un llamado al uso de nombres descriptivos para los casos de usos, que ubiquen expectativas en el lector. Es decir que revelen la intención del caso de uso y refleje el único objetivo que el actor está intentando lograr. Para ello este patrón propone como buena práctica nombrar los casos de uso comenzando con un verbo activo que describa el uso

del caso de uso y seguido del verbo con una frase que describa su propósito. Ser conciso pero lo suficientemente descriptivo para capturar la esencia del caso de uso.

- **Completar una única meta**: Consiste en escribir cada caso de uso dirigido hacia una completa y bien definida meta.
- **Alternativas exhaustivas, íntegras:** Trata de capturar todos los fallos y alternativas que deben ser manejados en el caso de uso. Una vez se tienen identificados todos los casos de uso y su flujo principal, se identifican y capturan todas las variaciones del flujo principal que se quiera que el sistema maneje.
- **Claro Conjunto de Roles:** Plantea identificar los actores y el rol que cada uno juega con respecto al sistema. Describir claramente cada uno de los actores con que el sistema debe actuar recíprocamente.
- **Múltiples actores. Roles diferentes:** Captura la concordancia entre actores manteniendo roles separados. Consiste de un caso de uso y por lo menos dos actores. Es utilizado cuando dos actores juegan diferentes roles en un caso de uso, o sea, interactúan de forma diferente con el caso de uso.

Teniendo en cuenta los requisitos funcionales, así como los patrones de casos de uso enunciados, se identificaron los casos de uso:

- **CUS 1: Autenticarse.**

Inicia cuando el usuario accede a la aplicación e introduce su nombre de usuario y contraseña. Finaliza cuando el sistema reconoce la validez de los datos suministrados por el usuario.

- **CUS 2: Consultar esquema.**

Inicia cuando el usuario solicita al sistema los datos de un esquema determinado. Finaliza cuando el sistema devuelve dichos datos.

- **CUS 3: Buscar esquema.**

Inicia cuando el usuario introduce los criterios de búsqueda en la herramienta y finaliza cuando el sistema muestra una lista con los esquemas que corresponden a los criterios especificados.

- **CUS 4: Realizar solicitud.**

Inicia cuando el solicitante ingresa en la aplicación los datos de la solicitud definidos en las reglas del negocio y finaliza cuando dichos datos se registran exitosamente en la aplicación.

- **CUS 5: CRUD usuarios.**

Inicia cuando el administrador realiza una de las acciones: eliminar, crear o modificar usuario. Finaliza cuando el sistema realiza con éxito las operaciones solicitadas.

- **CUS 6: CRUD esquema.**

Inicia cuando la comisión de registro realiza una de las siguientes acciones: modificar o eliminar esquema. Finaliza cuando el sistema notifica el éxito de las operaciones realizadas.

- **CUS 7: Revisar solicitud.**

Comienza cuando la comisión de registro accede a los datos de la solicitud con el objetivo de revisarla. Concluye cuando el sistema muestra los datos solicitados.

CUS/RF	RF 1	RF 2	RF 3	RF 4	RF 5	RF 6	RF 7	RF 8	RF 9	RF 10	RF 11	RF 12	RF 13	RF 14
CUS 1	X													
CUS 2			X	X		X				X				
CUS 3		X												
CUS 4					X									
CUS 5												X	X	X
CUS 6		X	X	X			X		X					
CUS 7			X	X			X	X		X	X			

Tabla 3: Relación entre CU y RF.

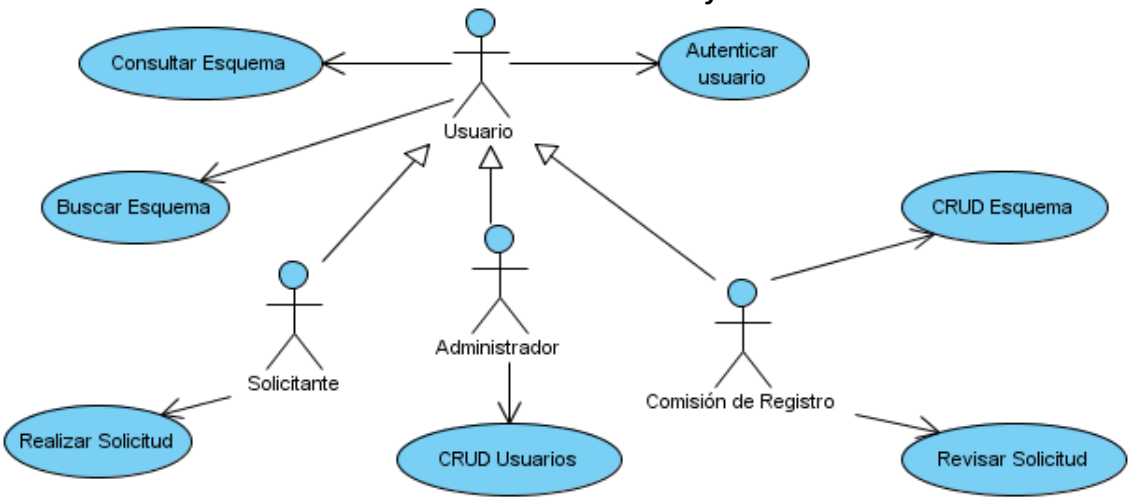

Figura 6: Diagrama de casos de uso del sistema.

2.4. Conclusiones del capítulo.

A lo largo de este capítulo se fueron abordando elementos que posibilitan una mejor comprensión del sistema a desarrollarse. Transcurriendo desde la modelación del negocio, la identificación de requisitos

tanto funcionales como no funcionales, hasta la modelación del sistema a desarrollar, se fue ganando en comprensión y claridad sobre el producto que se pretende obtener.

CAPÍTULO 3: ANÁLISIS Y DISEÑO DEL SISTEMA

3.1. Introducción.

En el presente capítulo se aborda el tema del análisis y el diseño del sistema. Se muestran diagramas que faciliten la comprensión del sistema y se enumeran los principales patrones utilizados en la obtención del diseño.

3.2. Patrones de arquitectura y diseño.

Como se mencionó en el capítulo 1 del informe, para la implementación del sistema se utilizará el framework Symfony. Este framework está diseñado de tal forma que integra muchos de los patrones de arquitectura y diseño más conocidos y probados. Esto le permite al desarrollador hacer uso extensivo de los mismos y dotar al sistema de una mayor calidad.

3.2.1. Patrón Arquitectónico MVC (Modelo – Vista – Controlador)

Es un patrón clásico del diseño web que está formado por tres niveles:

- El modelo representa la información con la que trabaja la aplicación, es decir, su lógica de negocio.
- La vista transforma el modelo en una página web que permite al usuario interactuar con ella.
- El controlador se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista.

La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. El controlador se encarga de aislar al modelo y a la vista de los detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes.

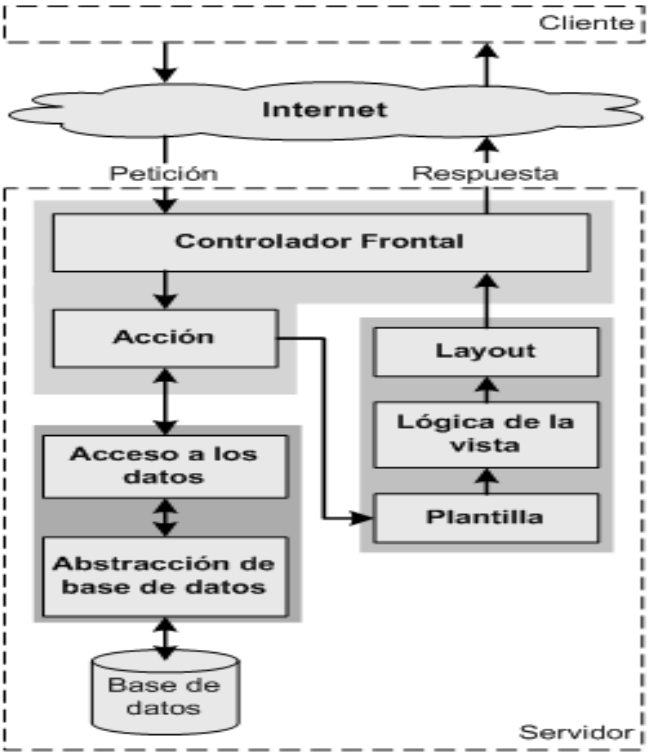

Figura 7: El patrón MVC (Potencier, 2010)

El controlador frontal es un componente del framework que, unido con el layout son comunes para todas las páginas de la aplicación. Las clases de la capa del modelo son creadas en función de la estructura de datos de la aplicación.

El uso de este patrón resulta bastante útil. Además promueve la disciplina en el desarrollo, pues obliga a dividir y organizar el código de acuerdo a las convenciones establecidas por el patrón. El código de la presentación corresponde a la vista, el código de manipulación de datos al modelo y la lógica de procesamiento de las peticiones constituye el controlador.

3.2.2. Aplicación de los patrones GRASP.

En la implementación se utilizan numerosos patrones GRASP (General Responsibility Assignment Software Patterns), a continuación se mencionan algunos ejemplos, ubicándolos en las capas de Modelo y Control que plantea el patrón arquitectónico MVC.

- **Experto en información:** En el modelo de la arquitectura propuesta, existen dos tipos de clases fundamentales:
  - Las encargadas de la abstracción de datos (entidad, realizan todas las operaciones con la BD).
  - Las de acceso a datos (modelo, interactúan con las clases de abstracción de datos y devuelven los objetos necesarios para los controladores).

De ambos tipos de clases se obtienen todos los datos de los objetos, por ello los métodos que usan estos datos, se encuentran implementados dentro de estas clases, logrando así una mayor encapsulación y cohesión.

- **Patrón Creador:** Las clases ***actions*** son las que utilizan las instancias de otras clases por ser controladoras, e incluso manejan colecciones de estas. Por ello son las más indicadas para crear los objetos que utilizan.
- **Patrón Bajo Acoplamiento:** Las clases que implementan la lógica de negocio y de acceso a datos se encuentran en el modelo, estas clases no tienen asociaciones con las de la vista o el controlador por lo que la dependencia en este caso es baja, demostrándose de esta manera el uso de este patrón.
- **Alta Cohesión:** La aplicación de este patrón se logra estableciendo que toda la información que maneja una clase, esté relacionada estrechamente con esta. Para ellos, se establecen las clases entidad ***AemEsquema***, que contiene los datos propios del esquema, y la clase ***AemFicha***, con los datos de la ficha de solicitud.
- **Patrón Controlador:** En consonancia con la separación del proyecto en capas, se establece el controlador como el intermediario entre las funcionalidades que provee la interfaz de usuario, y los algoritmos que implementan dicha funcionalidad. Un ejemplo de esto en el sistema son las clases ***actions***, que reciben los datos desde la vista, y los envían hacia el modelo haciendo uso de los métodos y objetos definidos en el mismo.

3.2.3. Aplicación de patrones GoF.

Además de los patrones antes mencionados, también se utilizan patrones GoF (Gang of Four) en el desarrollo, ejemplos de los cuales son:

- **Patrón Decorator**: Este patrón define elementos generales de la vista en los cuales se enmarcan otros más específicos. El archivo llamado ***layout.php*** contiene el layout de la página, por lo cual se le

llama plantilla global, y almacena el código HTML y la lógica de presentación común a todas las páginas de la aplicación. El contenido especializado de la vista se integra en este layout de la forma en que se muestra:

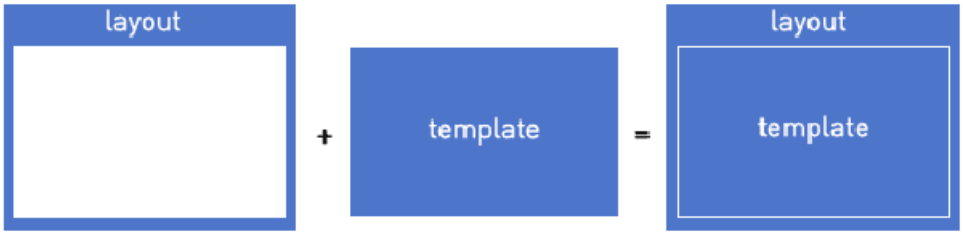

Figura 8: Forma en que funciona el patrón Decorator (Potencier, y otros, 2007).

- **Patrón Singleton:** En ocasiones es importante definir clases de las cuales se requiere el uso de una única instancia a lo largo de toda la ejecución de la aplicación. Un ejemplo de esto es la clase ***myUser***, la cual se emplea para definir datos de sesión del usuario que utiliza la aplicación. De esta clase se accede siempre a una única instancia, pues según la lógica del sistema solo un usuario puede estar autenticado a la vez por cada ejecución del sistema.
- **Patrón Registry:** Provee un medio simple y eficiente de compartir datos y objetos sin tener que preocuparse de mantener numerosos parámetros o hacer uso de variables globales. La aplicación de este patrón se observa en la clase ***sfConfig***, clase encargada de almacenar todas las variables de uso global en la aplicación. Basta con hacer uso de los métodos ***set*** para establecer valores o registrar variables, y acceder a ellas posteriormente desde cualquier lugar a través de los métodos ***get***.

3.3. Clases de Diseño

3.3.1. Diagrama de clases de Diseño.

Una clase de diseño y sus objetos participan en varias realizaciones de casos de uso. También puede suceder que algunas operaciones, atributos y asociaciones sobre una clase específica sean solo relevantes para una sola realización de caso de uso. Para manejar todo esto se utilizan los diagramas de clases conectados a una realización de caso de uso, mostrando así sus clases participantes, subsistemas y sus relaciones (Jacobson, y otros, 2000).

La figura 9 muestra el diagrama de clases de diseño correspondiente al módulo Solicitud. En él se muestra la separación por paquetes, evidenciando la arquitectura MVC; y las relaciones entre los diferentes componentes del mismo.

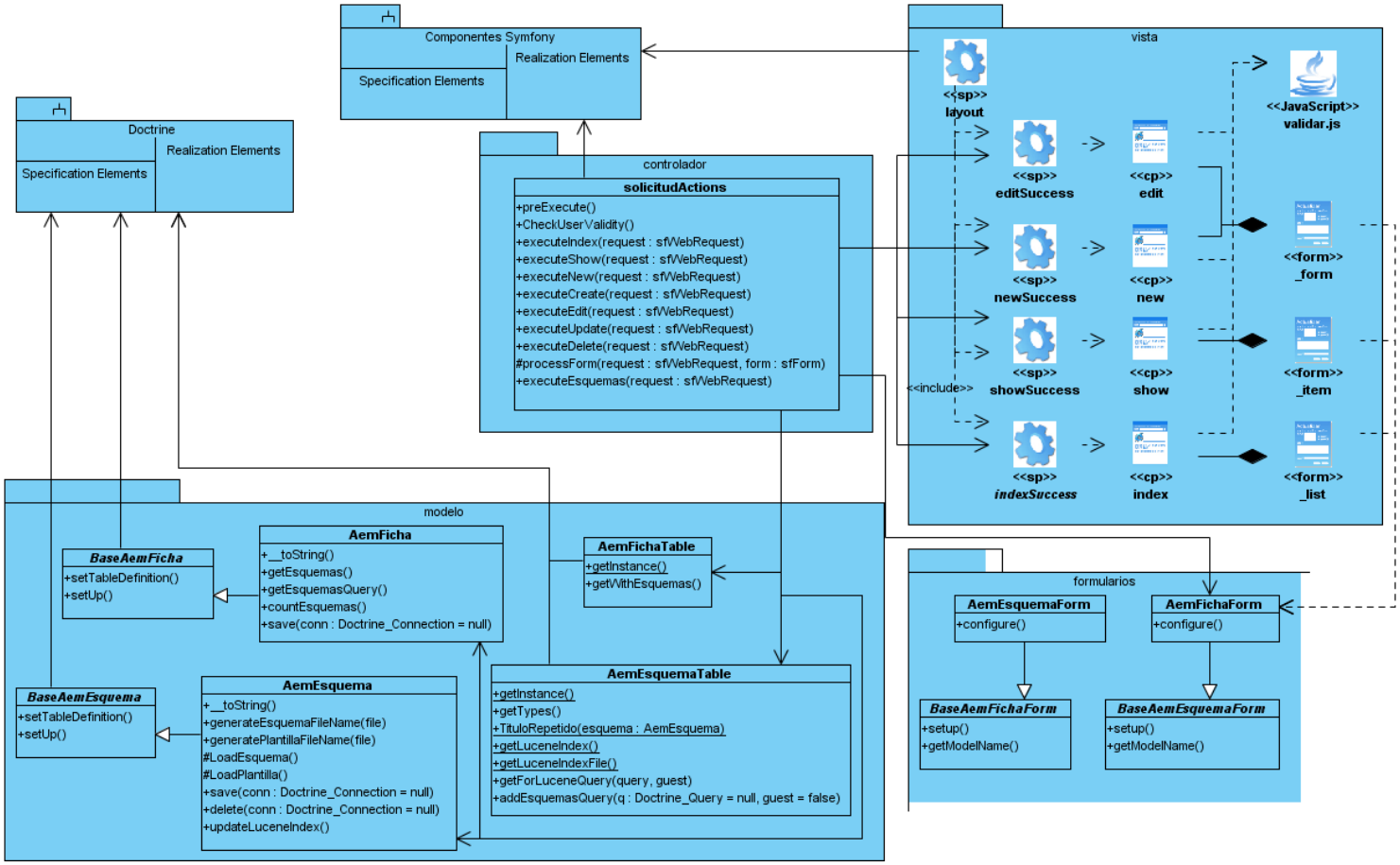

Figura 9: Diagrama de clases de diseño del módulo Solicitud.

Como se puede apreciar, existen tres paquetes fundamentales: controlador, vista y modelo.

- **Controlador:** En este paquete se incluyen las clases ***actions*** que como se explicó anteriormente en este capítulo, son el intermediario entre las clases entidad y las clases interfaz. Obtienen las solicitudes y los datos del cliente y se encargan de invocar la lógica del negocio a través de clases entidad.
- **Vista:** Las páginas web suelen contener elementos que se muestran de forma idéntica a lo largo de toda la aplicación: cabeceras de página, el layout genérico, el pie de página, entre otros. Muchas

veces sólo cambia el contenido de la página. Por este motivo, la vista se separa en un layout y en una plantilla (ej. ***newSuccess.php***). El layout es global en toda la aplicación y la mayoría de las páginas. La plantilla sólo se encarga de visualizar los elementos específicos, vincular determinadas acciones, incluir otras plantillas y construir el formulario (ej. ***AemFichaForm***).

- **Modelo:** Contiene las clases encargadas del acceso a los datos almacenados en el gestor de base de datos, que se realiza por el ORM (Object Relational Model) Doctrine. Para representar el modelo de la base de datos, se crea una serie de clases. A continuación se relacionan los tipos de clases generados en un proyecto Symfony+Doctrine.
  - **Clases Base:** Estas clases se generan a partir del esquema, y son clases entidad. Cada vez que se altera el esquema estas clases se vuelven a generar. Es por eso que la lógica incluida por el desarrollador no se debe introducir en este tipo de clases (ej. ***BaseAemEsquema***).
  - **Clases de Objetos:** Las clases de objetos propias que están en el directorio heredan de las clases con prefijo Base, por lo que también son clases entidad. Estas clases no se modifican cuando se cambia el esquema, por lo que en estas clases se añaden los métodos creados por el desarrollador. Esta clase hereda todos los métodos de la clase Base, pero no le afectan las modificaciones en el esquema gracias al bajo acoplamiento presente en el diseño. Este mecanismo de clases personalizadas que heredan de las clases base permite empezar a programar desde el primer momento, sin ni siquiera conocer el modelo relacional definitivo de la base de datos. La estructura de archivos creada permite personalizar y evolucionar el modelo (ej. ***AemEsquema***).
  - **Clases Table:** Son las clases que tienen métodos estáticos para trabajar con las tablas de la base de dato, por lo cual se les considera clases de modelo. Proporcionan los medios necesarios para obtener los registros de las tablas. Sus métodos devuelven normalmente un objeto o una colección de objetos de la clase entidad relacionada (ej. ***AemEsquemaTable***).

Como resultado de este sistema de representación, por cada tabla de la base de datos, se generan entonces tres clases.

3.3.2. Clases persistentes.

La persistencia es la capacidad de un objeto de mantener su valor en el espacio y en el tiempo. Las clases persistentes referencian directamente las entidades lógicas y sus atributos (Entorno Virtual de Aprendizaje, curso 2010 - 2011).

3.3.2.1. Diagrama de clases persistentes.

En este diagrama no se incluyen las clases relacionadas con la seguridad de la aplicación, provista por el plugin ***sfDoctrineGuard***.

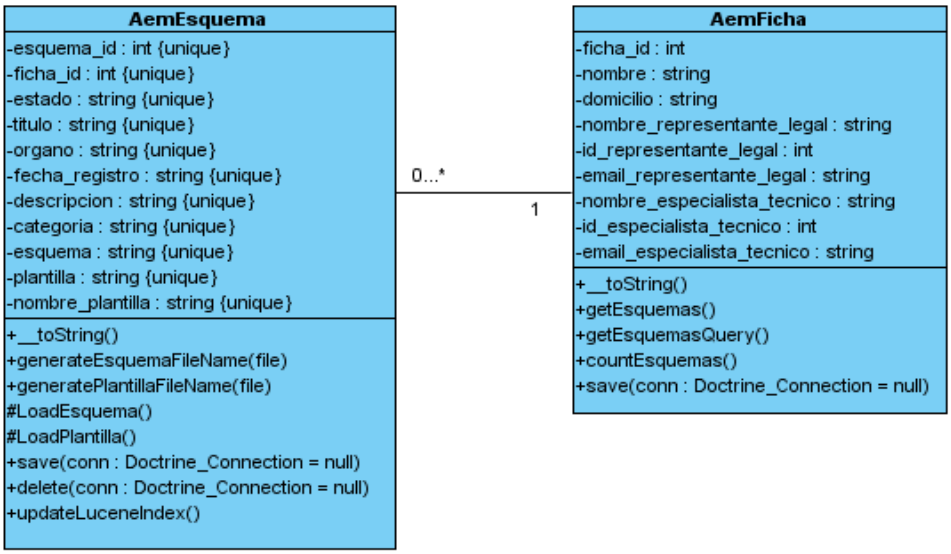

Figura 10: Diagrama de clases persistentes.

3.3.3. Modelo de datos.

El modelo de datos describe la representación lógica y física de la persistencia de los datos utilizados por la aplicación (Entorno Virtual de Aprendizaje, 2010 - 2011).

Al igual que en el diagrama anterior, se omitieron las tablas correspondientes al plugin de seguridad.

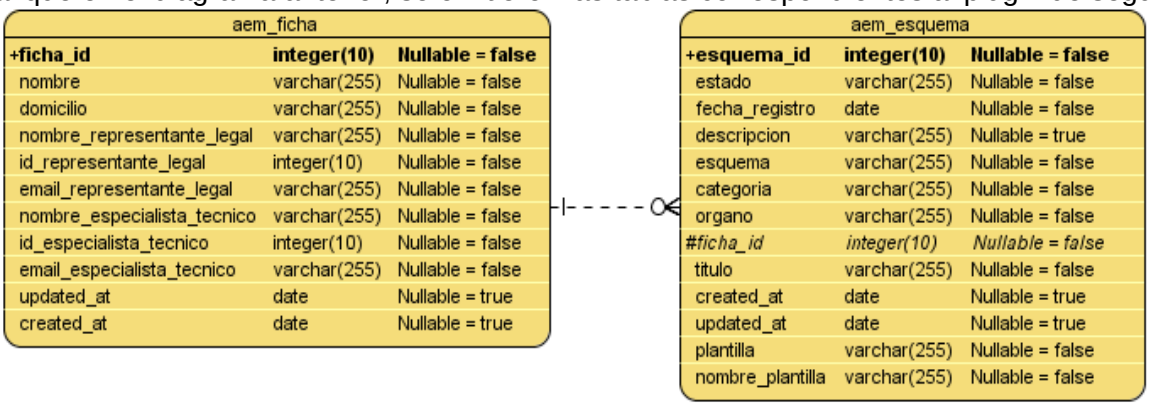

Figura 11: Modelo de Datos.

3.4. Métricas de diseño.

Las métricas del software permiten medir de forma cuantitativa la calidad de los atributos internos del producto, lo cual permite al desarrollador evaluar el desarrollo del sistema. Son varios los puntos de vista relacionados con este tema. En este epígrafe se evalúa el diseño propuesto para el Repositorio de Esquemas y Metadatos. Las métricas de diseño a nivel de componentes se concentran en las características internas de los componentes del software con medidas que pueden ayudar a juzgar la calidad del diseño a este nivel.

Un aspecto importante a tener en cuenta es la creación de métricas básicas inspiradas en el estudio de la calidad del diseño orientado a objetos. Los atributos de calidad que se miden son:

- **Responsabilidad**: Es la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- **Complejidad de implementación**: Es la complejidad de implementación que posee una estructura de diseño de clases.
- **Reutilización**: Es el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento**: Es el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.

- **Complejidad del mantenimiento**: Es el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
- **Cantidad de pruebas**: Es el número o el grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (componente, módulo, clase, conjunto de clases) diseñado.

Las métricas aplicadas al diseño de la solución propuesta son las siguientes:

- **Tamaño operacional de clase (TOC):** Está dado por el número de métodos asignados a una clase. Afecta los siguientes atributos:

Atributo de calidad	Modo en que lo afecta
Responsabilidad	El TOC es proporcional a la responsabilidad asignada a la clase.
Complejidad de implementación	El TOC es proporcional a la complejidad de implementación de la clase.
Reutilización	El TOC es inversamente proporcional del grado de reutilización de la clase.

Tabla 4: Relación entre la métrica TOC y los atributos que afecta.

Para la evaluación de los atributos de calidad anteriores se definieron los siguientes criterios y categorías:

Atributo	Categoría	Criterio
Responsabilidad	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Complejidad de implementación	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Reutilización	Baja	>2*Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	<=Promedio

Tabla 5: Criterios de evaluación de la métrica TOC.

- **Relaciones entre clases (RC):** Está dado por el número de relaciones de uso de una clase con otras. Afecta los atributos:

Atributo de calidad	Modo en que lo afecta
Acoplamiento	La RC es proporcional al Acoplamiento de la clase.
Complejidad de mantenimiento	La RC es proporcional a la complejidad del mantenimiento de la clase.
Reutilización	La RC es inversamente proporcional al grado de reutilización de la clase.
Cantidad de pruebas	La RC es proporcional a la Cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 6: Relación entre la RC y los parámetros que afecta.

Para la evaluación de los atributos de calidad anteriores se definieron los siguientes criterios y categorías:

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

Tabla 7: Criterios de evaluación de la métrica RC.

3.4.1. Resultado de la aplicación de las métricas.

Tras la aplicación de las métricas especificadas a 6 clases del diseño del sistema (2 de ellas controladoras y el resto modelo/entidad), se obtuvieron los siguientes resultados:

- **Métrica TOC.**

Al aplicársele TOC al diseño, se obtuvieron los valores que muestra la siguiente tabla:

Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
esquemaActions	14	Media	Media	Media
solicitudActions	11	Media	Media	Media
AemEsquema	8	Baja	Baja	Alta
AemEsquemaTable	8	Baja	Baja	Alta
AemFicha	6	Baja	Baja	Alta
AemFichaTable	2	Baja	Baja	Alta

Tabla 8: Evaluación de la métrica TOC.

Como se aprecia en la gráfica siguiente, el atributo Responsabilidad muestra resultados satisfactorios, pues el 67% de las clases tiene un grado de responsabilidad Bajo.

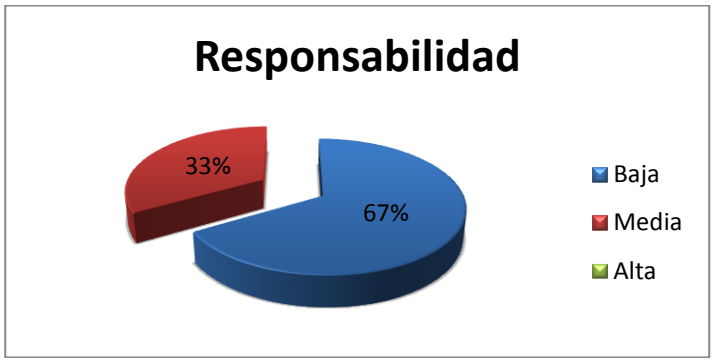

Figura 12: Responsabilidad de clases según la métrica TOC.

De igual manera, la gráfica del atributo Reutilización muestra resultados positivos, al poseer el 67% de las clases tiene un alto grado de reutilización.

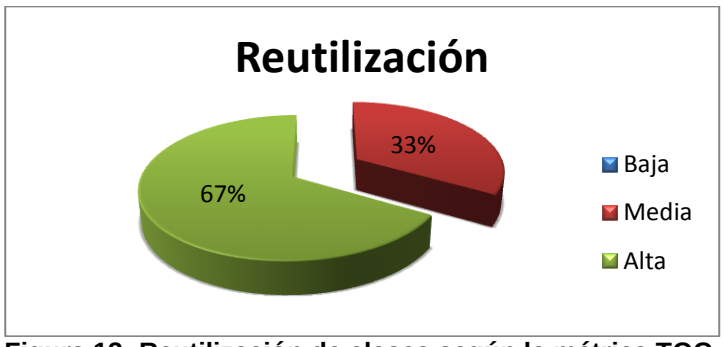

Figura 13: Reutilización de clases según la métrica TOC.

Por último, la Complejidad, según se muestra, es Baja en el 67% de las clases, siendo Media en el resto, lo cual también es un resultado satisfactorio.

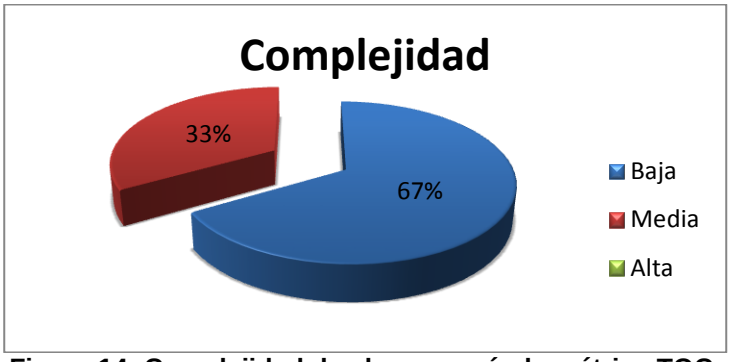

Figura 14: Complejidad de clases según la métrica TOC.

- **Métrica RC.**

Al aplicársele RC al diseño, se obtuvieron los valores que muestra la siguiente tabla:

Clase	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad de Mantenimiento	Reutilización	Cantidad de Pruebas
esquemaActions	3	Alto	Alta	Baja	Alta
solicitudActions	2	Medio	Media	Media	Media
AemEsquema	1	Bajo	Baja	Alta	Baja
AemEsquemaTable	1	Bajo	Baja	Alta	Baja
AemFicha	1	Bajo	Baja	Alta	Baja
AemFichaTable	2	Medio	Media	Media	Media

Tabla 9: Evaluación de la métrica RC.

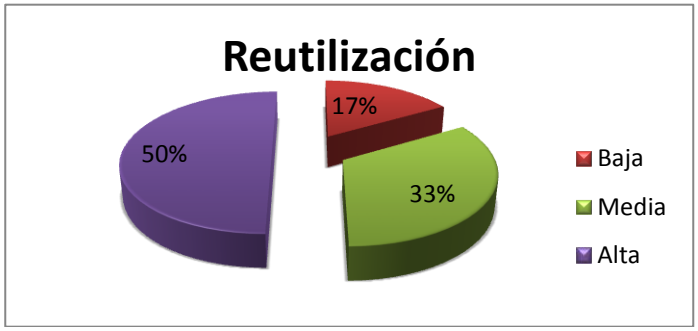

Figura 15: Reutilización de las clases según RC.

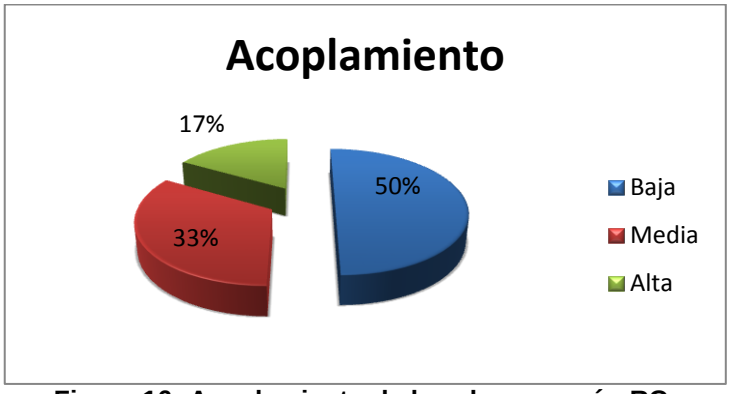

Figura 16: Acoplamiento de las clases según RC.

Figura 17: Complejidad de mantenimiento de las clases según RC.

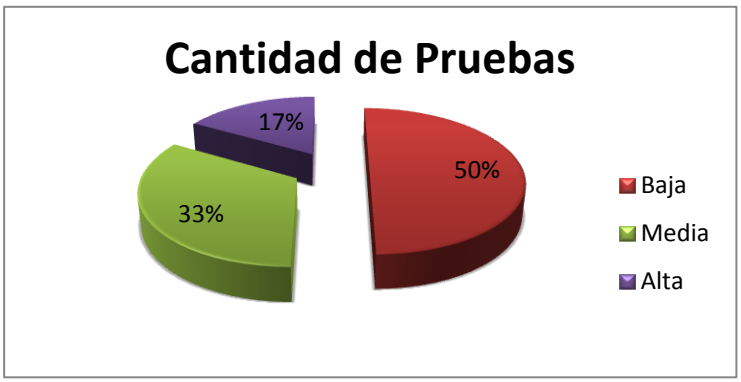

Figura 18: Cantidad de pruebas según RC.

3.4.1.1. Matriz de inferencia de indicadores de calidad

Con la matriz de inferencia de indicadores de calidad se representan los atributos de calidad y las métricas que se utilizaron para medir la calidad del diseño del sistema. La matriz muestra si los resultados de las relaciones entre atributos y métricas son positivos o negativos a partir de una escala numérica. En la

escala si los resultados son positivos se les asigna el valor de 1, si son negativos toman valor 0. Si no existe relación es considerada como nula y se representa con un guión simple "-". Luego se puede obtener un resultado general para cada atributo promediando todas sus relaciones no nulas.

Categoría	Rango de valor
Malo	<=0.4
Regular	>0.4 y <0.7
Bueno	>=0.7

Tabla 10: Rango de valores para la evaluación de la relación Atributo/Métrica.

Atributo/Métrica	TOC	RC	Promedio
Responsabilidad	1	-	1
Complejidad de Implementación	1	-	1
Reutilización	1	1	1
Acoplamiento	-	1	1
Complejidad de Mantenimiento	-	1	1
Cantidad de pruebas	-	1	1

Tabla 11: Resultados de la evaluación de la relación Atributo/Métrica.

Tras evaluarse los atributos de calidad se obtuvieron los siguientes resultados:

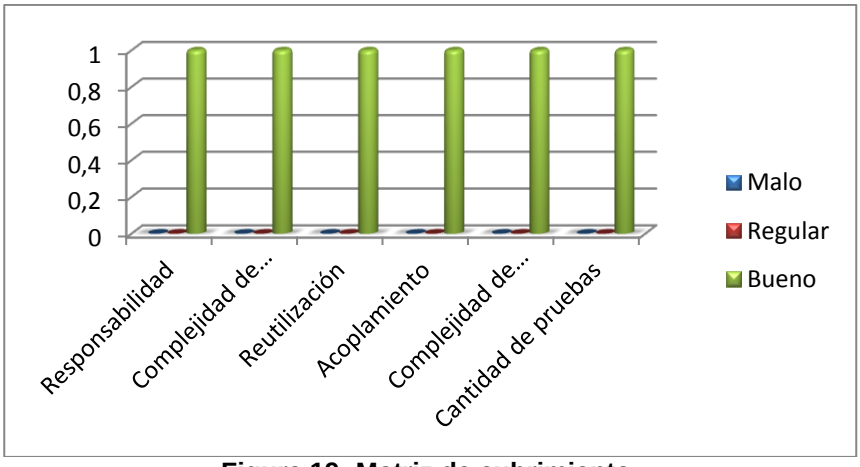

Figura 19: Matriz de cubrimiento.

3.5. Conclusiones del capítulo.

Después de transitar por todos los elementos de este capítulo, habiéndose realizado el modelo de clases de diseño, analizados los patrones de diseño que se tendrán en cuenta, y habiendo modelado las clases

persistentes, así como el modelo de datos, se está en condiciones de proceder a la implementación del sistema. La aplicación de las métricas de diseño mostró que el mismo es de buena calidad.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

4.1. Introducción.

Durante el presente capítulo se presentan los artefactos correspondientes a la implementación del sistema así como las pruebas que aseguren su validación. Se mencionan elementos del estilo de codificación, se muestra el diagrama de despliegue, y se realizan pruebas para asegurar la calidad del sistema.

4.2. Implementación.

En la implementación se comienza con el resultado del diseño y se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, *scripts*, ficheros de código binario, ejecutables y similares (Jacobson, y otros, 2000).

4.2.1. Modelo de implementación.

El modelo de implementación describe cómo los elementos del diseño, como las clases, se implementan en términos de componentes, como ficheros de código fuente, ejecutables (…) describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados (…) es la entrada principal de las etapas de prueba que siguen a la implementación (Jacobson, y otros, 2000).

4.2.1.1. Diagramas de componentes.

Un diagrama de componentes *"muestra un conjunto de componentes y sus relaciones; los diagramas de componentes muestran los componentes de un sistema desde un punto de vista estático"* (Jacobson, y otros, 2000)**.**

El sistema está formado por tres módulos: solicitud, esquema, y administración. Esta distribución se concibe para dotarle al sistema de una mayor cohesión y disminuir la dependencia entre los diferentes componentes.

- **Solicitud:** El objetivo de este módulo es separar del resto a los elementos correspondientes a la solicitud. En él se maneja lo referente a la creación, gestión y consulta de solicitudes.

- **Esquema:** El objetivo de este módulo es englobar los elementos específicos de los esquemas. Se maneja lo referente a la consulta y gestión de los esquemas.
- **Administración:** Este módulo responde a la necesidad de proveer funcionalidades para la gestión de usuarios y permisos para el sistema.

A pesar de ser módulos separados, Esquemas y Solicitud usan elementos comunes. Esto se realizó tratando de mantener al mínimo el acoplamiento entre ambos.

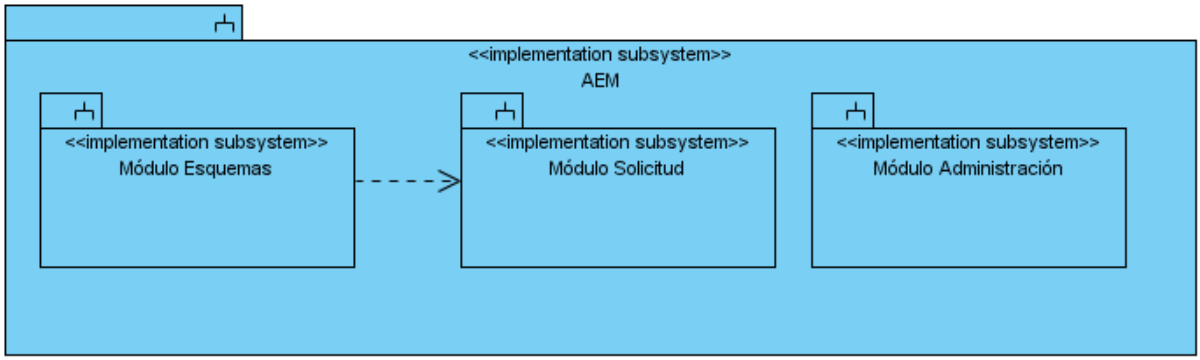

Figura 20: Diagrama de componentes de la aplicación.

El módulo solicitud está compuesto internamente por una vista, un modelo y un controlador, con la separación que se especificó anteriormente en el capítulo 3 de este informe:

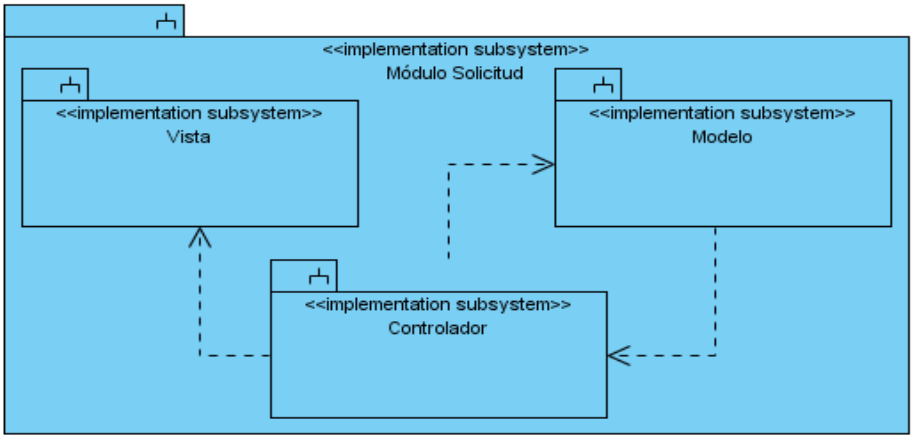

Figura 21: Diagrama de componentes del módulo solicitud.

Para ahondar aún más en el módulo, se muestran los diagramas correspondientes a cada subsistema (Vista, Controlador y Modelo):

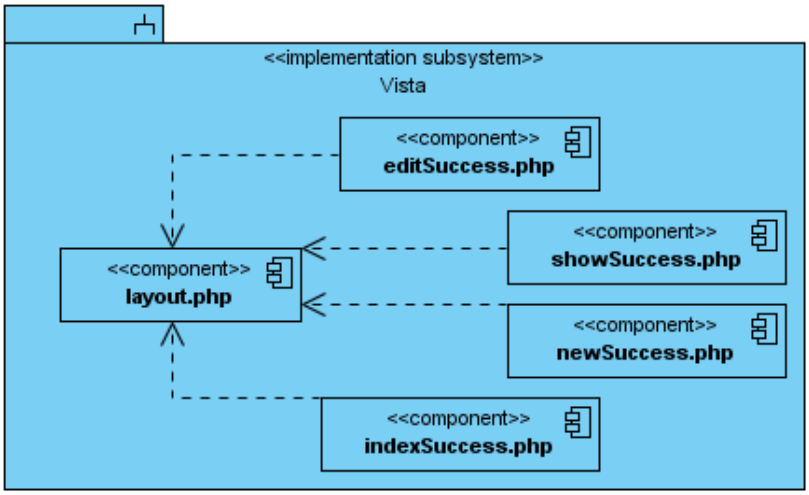

Figura 22: Diagrama de componentes de la vista del módulo Solicitud.

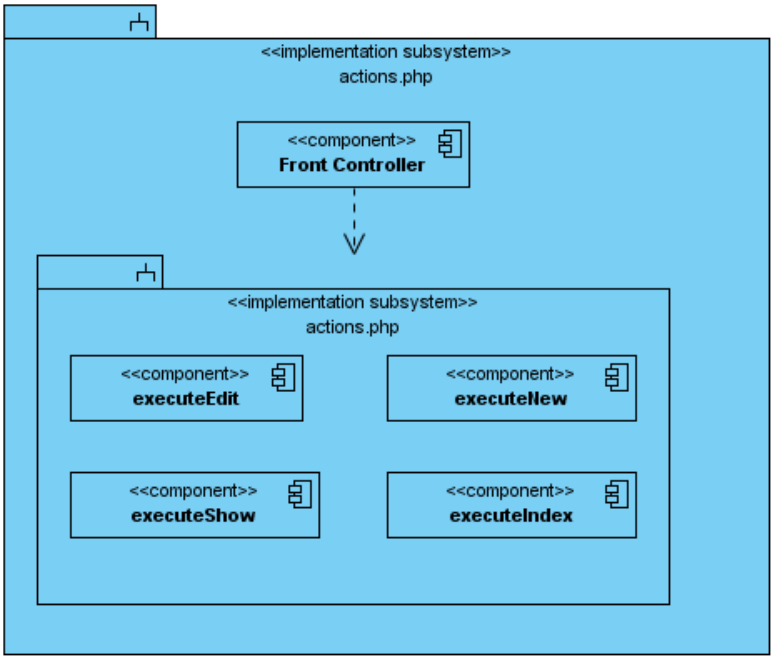

Figura 23: Diagrama de componentes de la vista del módulo Solicitud.

Por último, se muestra el diagrama correspondiente al modelo:

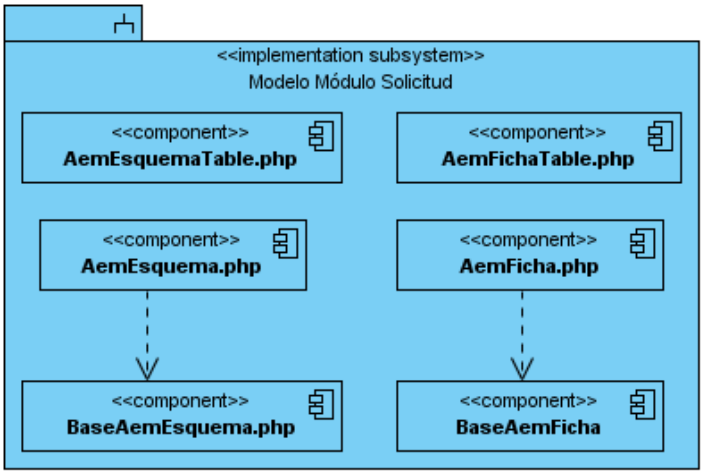

Figura 24: Diagrama del Modelo del módulo Solicitud.

4.2.1.2. Solución propuesta.

Se muestran algunas de las interfaces de usuario que presenta la aplicación. Han sido diseñadas con el objetivo de maximizar la funcionalidad, manteniendo características que las hagan agradables e intuitivas:

Figura 25: Interfaz de usuario principal del módulo Esquema.

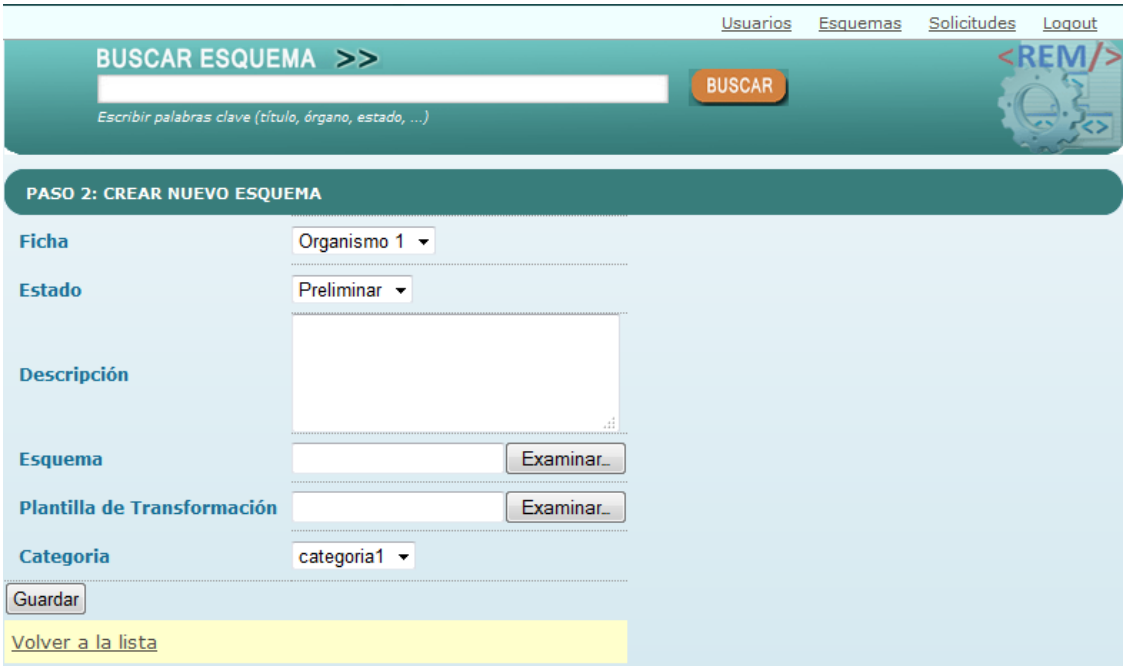

Figura 26: Interfaz de usuario correspondiente a la funcionalidad Crear Esquema.

4.2.2. Modelo de despliegue.

"El modelo de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo.

Se puede observar lo siguiente sobre el modelo de despliegue:

- Cada nodo representa un recurso de cómputo, normalmente un procesador o un dispositivo de hardware similar.
- Los nodos poseen relaciones que representan medios de comunicación entre ellos, tales como *Internet, intranet, bus y similares.*
- El modelo de despliegue puede describir diferentes configuraciones de red, incluidas las configuraciones para pruebas y para simulación" (Jacobson, y otros, 2000).

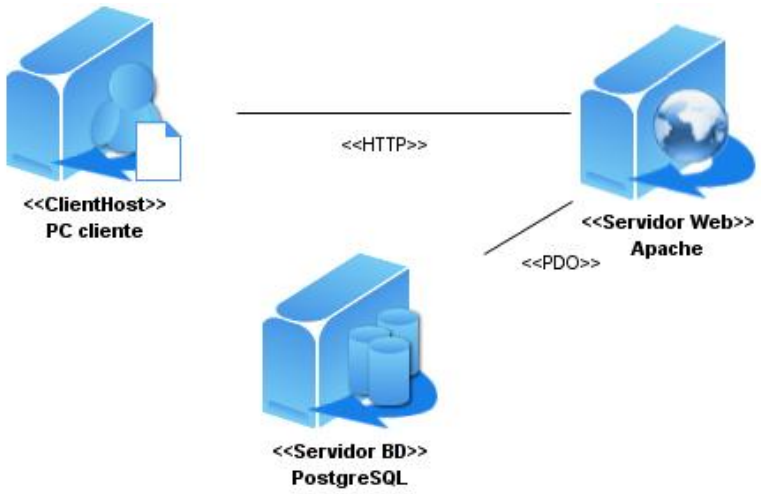

Figura 27: Diagrama de despliegue.

4.2.2.1. Descripción de los nodos físicos.

Un nodo es "un elemento físico que existe en tiempo de ejecución y que representa un recurso computacional, que en general tiene al menos una memoria y a menudo capacidad de procesamiento." (Jacobson, y otros, 2000)

- **PC Cliente:** Sistema Operativo Windows o Unix, y un navegador compatible con IE 6 o Mozilla Firefox 3, mediante los cuales los clientes tendrán acceso al sistema.
- **Servidor Web Apache:** En este servidor se ejecuta la aplicación.
- **Servidor BD PostgreSQL:** Este servidor posee la Base de Datos del sistema.

4.3. Estándares de codificación.

Los estándares de codificación son reglas o buenas prácticas que se establecen para dotar el código de una mayor legibilidad y por tanto mayor capacidad de mantenimiento. Mediante la buena aplicación de los estándares de codificación se puede lograr unidad, claridad y limpieza en el código.

Los estilos utilizados son los siguientes:

- Notación PascalCasing: En esta notación los identificadores y nombres de variables, métodos y funciones están compuestos por múltiples palabras juntas, iniciando cada palabra con letra mayúscula. El objetivo es que cada nombre aporte la mayor información posible sobre sus características.

- Notación CamelCasing: Similar al PascalCasing, pero a diferencia de este, la letra inicial del identificador siempre estará en mayúscula.

La forma en que se utilizarán los estilos son:

- **Para las clases:** se utilizará PascalCasing.
- **Para las funciones:** se utilizará CamelCasing.
- **Para las variables:** se utilizará CamelCasing.

4.4. Pruebas.

Habiéndose realizado la implementación del sistema, es preciso comprobar que este funcione correctamente.

4.4.1. Casos de pruebas.

Para comprobar la funcionalidad del sistema a través del método de caja negra, se elaboró una serie de casos de prueba.

Diseño del **Caso de prueba 1: CU** CRUD Esquema.

Descripción General:

Inicia cuando la comisión de registro realiza una de las siguientes acciones: modificar o eliminar esquema. Finaliza cuando el sistema notifica el éxito de las operaciones realizadas.

Condiciones de ejecución:

- El usuario debe estar autenticado y tener el rol de Revisor Técnico.
- El usuario debe seleccionar el esquema y pulsar "***Editar***" y seleccionar el nuevo estado del esquema.
- El usuario debe pulsar el botón "***Guardar***".

Nombre de la Sección	Escenarios de la sección	Descripción de la funcionalidad
SC 1: Actualizar Esquema	EC 1.1: Actualizar el esquema	1.1 El sistema muestra al usuario los parámetros a modificar del esquema: **Ficha.** **Estado.** **Descripción.** **Esquema a adjuntar** **Plantilla a adjuntar.** **Categoría del esquema.** Se muestra además el botón: **Guardar.**
	EC 1.2 Datos incorrectos	El sistema muestra un mensaje de error al querer subir un archivo no valido en los campos: **Esquema a adjuntar** **Plantilla a adjuntar.**
	EC 1.2 Campos vacíos	El sistema vuelve a la página anterior y muestra un mensaje de error, señalando el campo vacío como requerido.

Tabla 12: Secciones a probar en el CU Actualizar Esquema.

No	Nombre del campo	Clasificación	Valor nulo	Descripción
1	Ficha	Lista desplegable	no	El campo solo permite introducir letras.
2	Estado	Lista desplegable	no	El campo puede contener letras.
3	Descripción	Campo de texto	si	El campo permite la entrada de números, letras y caracteres especiales.
4	Esquema	Control de subir archivo	no	El control sólo permite subir archivos XML schema (xsd)
5	Plantilla de transformación	Control de selección	no	El control sólo permite subir archivos (xsl)
6	Categoría	Lista desplegable	no	El campo permite introducir números, letras y caracteres inválidos.

Tabla 13: Descripción de las variables.

Escenario	Respuesta esperada	Ficha	Estado	Descripción	Esquema	Plantilla de transformación	Categoría	Resultado de la prueba
EC 1.1: Actualizar el esquema	Se actualiza el esquema con los datos nuevos introducidos	V	V	V	V	V	V	Satisfactorio
EC 1.2: Datos incorrectos.	El sistema muestra un mensaje de error.	N/A	N/A	N/A	I	I	N/A	Satisfactorio
EC 1.2 Campos vacíos	El sistema muestra un mensaje de error y señala el campo requerido.	N/A	N/A	N/A	N/A	N/A	I	Satisfactorio

Tabla 14: Matriz de Datos SC 1. Agregar ficha.

Las celdas de la tabla anterior contienen V, I, o N/A. V indica válido, I indica inválido, y N/A que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante.

Elemento	No	No Conformidad	Aspecto Correspondiente	Etapa de Detección	Significativa	No Significativa	Recomendación	Estado NC	Respuesta del Equipo de Desarrollo
–	–	–	–	–	–	–	–	–	–

Tabla 15: Registro de defectos y dificultades detectados.

En total se elaboró un caso de prueba por cada caso de uso en una primera iteración. De siete pruebas realizadas, fueron detectadas seis no conformidades. Estas no conformidades estuvieron relacionadas fundamentalmente con el diseño del sistema, y la forma en que el mismo cumplía con las funcionalidades especificadas. En una segunda iteración, para la misma cantidad de pruebas, no fueron detectadas no conformidades.

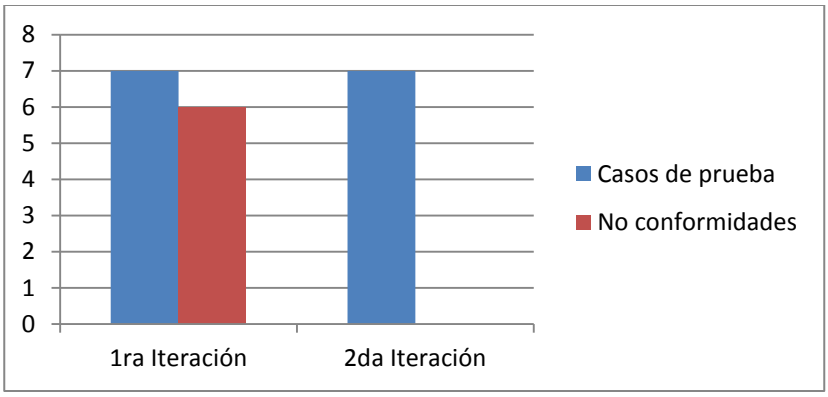

Figura 28: Casos de prueba realizados y no conformidades detectadas.

4.3. Conclusiones del capítulo.

En este capítulo se han mostrado los artefactos generados en el flujo de trabajo de Implementación, lo cual permite conocer la forma en que se realizó esta actividad. Además se han mostrado a través del modelo de prueba las actividades que permitieron asegurar que el sistema cumple con lo requerido.

CONCLUSIONES

Al finalizar el desarrollo de los objetivos se ha podido constatar que:

- Habiendo realizado un estudio sobre las experiencias a nivel global y más específicamente a nivel regional sobre el tema que se aborda, se pudieron tener en cuenta elementos útiles que permitieron elevarle la calidad a la solución propuesta, y a la vez adecuarla más al entorno nacional. El proceso de selección de las herramientas a utilizar contribuyó a obtener una mejor infraestructura para el desarrollo.
- Las actividades de modelado de negocio permitieron un mayor acercamiento entre la perspectiva del desarrollador y la necesidad del cliente, garantizándose así la conformidad por parte de este último con lo que se pretendía desarrollar.
- Mediante las actividades de diseño del sistema se logró tomar lo definido en el modelo del negocio y transformarlo en una guía para la implementación, facilitando de igual manera este proceso.
- Las pruebas realizadas verificaron la calidad del producto Repositorio de Esquemas y Metadatos, por lo que se cumplió con éxito el principal objetivo del trabajo.

RECOMENDACIONES

Al concluir este trabajo, se recomienda:

- Continuar el desarrollo de la aplicación para extender su funcionalidad.
- Implementarle al sistema en versiones posteriores un mecanismo para la notificación vía correo electrónico del cambio del estado de los esquemas a los usuarios interesados.
- Integrarle al sistema, en la medida de lo posible, y sin afectar el rendimiento de este, elementos de JavaScript asíncrono que permita al usuario utilizar la aplicación desde un entorno más amigable.

BIBLIOGRAFÍA

Administración del SGBD PostgreSQL. **Alarcón, José Manuel. 2006.** 2006.
Arcia Carrazana, Maikel y Piñero, Pedro Y. 2010. *INFORME TECNOLÓGICO DE LA PRODUCCIÓN.* s.l. : UCI, 2010.
Bloomberg, Jason y Schmelzer, Ronald. 2006. *Service Orient or Be Doomed!* 2006.
CEPAL. 2007. *Libro blanco de interoperabilidad de gobierno electrónico para América Latina y el Caribe.* s.l. : CEPAL, 2007.
DESA. 2005. *UN Global E-government Readiness Report 2005.* s.l. : Naciones Unidas, 2005.
EICTA. 2007. EICTA Interoperability White Paper. [En línea] 2007. http://www.eicta.org/fileadmin/user_upload/document/document1166548285.pdf.
Entorno Virtual de Aprendizaje. curso 2010 - 2011. Ingeniería de Software II. *Conferencia # 5 "Culminación del flujo de trabajo Análisis y Diseño. Diseño de la Base de Datos".* [En línea] curso 2010 - 2011. http://eva.uci.cu/mod/resource/view.php?id=29333.
—. 2010 - 2011. Ingeniería de Software II. *Conferencia # 5 "Culminación del flujo de trabajo Análisis y Diseño. Diseño de la Base de Datos".* [En línea] 2010 - 2011. http://eva.uci.cu/mod/resource/view.php?id=29333.
European Commission. 2010. Annex II - EIF (European Interoperability Framework). *Communication "Towards interoperability for European public services".* 2010.
Flanagan, David. 2001. *JavaScript: The Definitive Guide, Fourth Edition.* s.l. : O'Reilly Media, 2001.
Freeman, R. E. 1984. *Strategic Management: A Stakeholder Approach.* 1984.
González, Ing. Jorge Luis Valdés. 2010. *PROPUESTA DE NORMA TÉCNICA PARA EL REGISTRO DE ESQUEMAS Y METADATOS DE DOCUMENTOS JURÍDICOS. APOYO A LA INTEROPERABILIDAD TÉCNICA.* La Habana, Cuba : Universidad de las Ciencias Informáticas, 2010.
IBM Corporation. 2007. Rational Method Composer Versión 7.2.0. *Classic RUP for SOMA.* [En línea] IBM Corporation, 2007.
IEEE. *610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology.* s.l. : IEEE.
Interoperability frameworks and enterprise architectures in e-government initiatives in Europe and the United States. **Guijarro, Luis. 2007.** 1, s.l. : Government Information Quarterly, 2007, Vol. 24.
Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000. *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison - Wesley, 2000.
Juliá, Miguel Angel Abellán. 2004. *LA COLABORACIÓN, UNA REALIDAD GRACIAS A LA ARQUITECTURA TECNOLÓGICA "HP eGOVERNMENT FRAMEWORK".* s.l. : Hewlett-Packard Española, SL, 2004.
LA REALIDAD LATINOAMERICANA EN GESTIÓN DE DOCUMENTOS ELECTRÓNICOS. **Mendoza Navarro, Aida Luz. 2010.** 2010.
Lamb, David Alex. 1998. *What's a CASE Tool?* s.l. : Advameg, Inc., 1998.
Marqués Andrés, María Mercedes. Sistemas de bases de datos. [En línea] [Citado el: 10 de marzo de 2010.] http://www3.uji.es/~mmarques/f47/apun/node4.html.
Mesa, Francisco Aragón. 2005. *Introducción a la programación orientada a objetos.* 2005.
Ministerio de Economía y Fomento. 2009. Administrador de Esquemas y Metadatos. AEM. [En línea] 2009. http://www.aem.gob.cl/archivos/AEM-PRES-Como_funciona-052009.pdf.
Ministerio de Tecnologías de la Información y las Comunicaciones, Colombia. 2011. Portal del lenguaje común de intercambio de información. [En línea] 2011. http://lenguaje.intranet.gov.co/web/gelxml/inicio.

Bibliografía

Moreno Escobar, Hernán, Sin Triana, Hugo y Caino Silveira Netto, Sérgio. 2007. *Conceptualización de arquitectura de gobierno electrónico y plataforma de interoperabilidad para América Latina y el Caribe.* s.l. : Naciones Unidas, 2007.
Oracle Corporation. 2011. NetBeans. [En línea] 2011. http://netbeans.org/index.html.
Organización Internacional de Estandarización (ISO). 2001. *ISO-15489-1 Information and Documentations. Records Management. Part 1. General.* s.l. : ISO, 2001.
—. 2006. *ISO-23081-1 Información y Documentación - Procesos de gestión de documentos. Metadatos para la gestión de documentos.* s.l. : ISO, 2006.
Övergaard, Gunnar y Palmkvist, Karin. 2004. *Use Cases Patterns and Blueprints.* s.l. : Addison Wesley Professional, 2004. ISBN 0-13-145134-0.
Perens, Bruce. 2007. Open Standards: Principles and Practice. [En línea] 2007. http://www.perens.com/OpenStandards/Definition.html.
PostgreSQL Global Development Group . 2011. PostgreSQL. [En línea] 2011. http://www.postgresql.org/.
Potencier, Fabien. 2010. *A Gentle Introduction to symfony.* s.l. : Sensio Labs, 2010.
Potencier, Fabien y Zaninotto, François. 2007. *Symfony, la guía definitiva.* www.librosweb.es : s.n., 2007.
Pressman, Roger S. 2002. *Ingeniería de Software, un enfoque práctico.* 2002.
Rumbaugh, James, Jacobson, Ivar y Booch, Grady. 1998. *The Unified Modeling Language. Reference Manual.* Massachusetts : Addison Wesley, 1998.
SPARX Systems. 2011. [En línea] 2011. http://sparxsystems.com.ar/products/ea.html.
Study on Interoperability at Local and Regional Level. 2007. *Interoperability Study.* s.l. : European Commission, 2007.
Sutor, Bob. 2007. Interoperability More or Less. [En línea] 2007. http://www.sutor.com/newsite/blog-open/?p=1372.
The PHP Group. 2011. PHP: Hypertext Preprocessor. [En línea] 2011. http://www.php.net/.
UNDESA. 2010. *United Nations E-Government Survey 2010. Leveraging e-government at a time of financial and economic crisis. 2010. pág. 1-140.* s.l. : United Nations Department of Economic and Social Affairs, 2010.
—. 2010. *United Nations E-Government Survey 2010. Leveraging e-government at a time of financial and economic crisis. 2010. pág. 140.* s.l. : United Nations Department of Economic and Social Affairs, 2010.
Vargas Del Valle, Ricardo J. 2007. Programación en Capas. [En línea] 2007. [Citado el: 1 de marzo de 2010.] http://www.di-mare.com/adolfo/cursos/2007-2/pp-3capas.pdf.
Vaswani, Vikram. 2008. *PHP. A Begginer's Guide.* New York : Mc Graw Hill, 2008.
Visual Paradigm International. 2011. Visual Paradigm. [En línea] 2011. http://www.visual-paradigm.com/.
W3C. XML Technology - W3C. [En línea] Se encuentra disponible en http://www.w3.org/TR/2008/WD-timesheets-20080110/. http://www.w3.org/standards/xml/.
World Wide Web Consortium. 2011. HTML & CSS - W3C. [En línea] 2011. http://www.w3.org/standards/webdesign/htmlcss.

GLOSARIO

C

Computer Assisted Software Engineering (CASE): Herramienta de diseño asistido por computadora especializada en Ingeniería de software.

D

Débilmente tipado: En el contexto de desarrollo de software, así se le llama al lenguaje de programación que no controla los tipos de las variables que se declaran.

E

Esquema XML: Lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa.

Estado del arte: Nivel más alto de desarrollo conseguido en un momento determinado sobre cualquier aparato, técnica o campo científico.

F

Framework: Estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado.

G

Gobierno electrónico: Consiste en el uso de las tecnologías de la información y el conocimiento en los procesos internos de gobierno y en la entrega de los productos y servicios del Estado tanto a los ciudadanos como a la industria.

Gang of Four (GoF): Grupo de cuatro autores que definen una serie de patrones, a los cuales se le conoce como patrones GoF.

General Responsibility Assignment Software Patterns (GRASP): Patrones de software de asignación de responsabilidades generales.

H

Hypertext Transfer Protocol (HTTP): Protocolo de transferencia de hipertexto, es el protocolo usado en cada transacción de Internet.

I

Integrated Development Environment (IDE): Entorno integrado de desarrollo, es un programa informático compuesto por un conjunto de herramientas de programación.

L

Layout: Herramienta de diseño web que define atributos comunes a varias plantillas.

M

Metadatos: Datos que describen otros datos.

O

Object Relational Mapping (ORM): Técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional.

P

Patrón de diseño: Es la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

PHP Data Objects (PDO): Extensión que provee una capa de abstracción de acceso a datos para PHP 5, con lo cual se consigue hacer uso de las mismas funciones para hacer consultas y obtener datos de distintos controladores de bases de datos.

Plugin: Un plugin o complemento es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica.

R

Repositorio: Sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.

S

Sistema gestor de bases de datos (SGBD): Tipo de software dedicado a servir de interfaz entre la base de datos, y el usuario o las aplicaciones que la utilizan.

Stakeholder: "quienes pueden afectar o son afectados por las actividades de una empresa" (Freeman, 1984)

T

Tecnologías de la Información y las comunicaciones (TICs): Agrupan los elementos y las técnicas utilizadas en el tratamiento y la transmisión de las informaciones, principalmente de informática, internet y telecomunicaciones.