UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 1

CENTRO DE INFORMATIZACIÓN UNIVERSITARIA (CENIA)

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

Propuesta de modelado para sistemas con arquitecturas basadas en Transferencia de Estado Representativo

Autor (es): Yurielkis Guzmán Matos

Tutor(es): Ing. Pedro Rodriguez Samon

Co-Tutor(es): Ing. Michel David Suárez

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al Centro de Informatización Universitaria (CENIA) de la Universidad de las Ciencias Informáticas; así como a dicha institución para que hagan el uso que estimen pertinente con este trabajo.

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

_____
Yurielkis Guzmán Matos

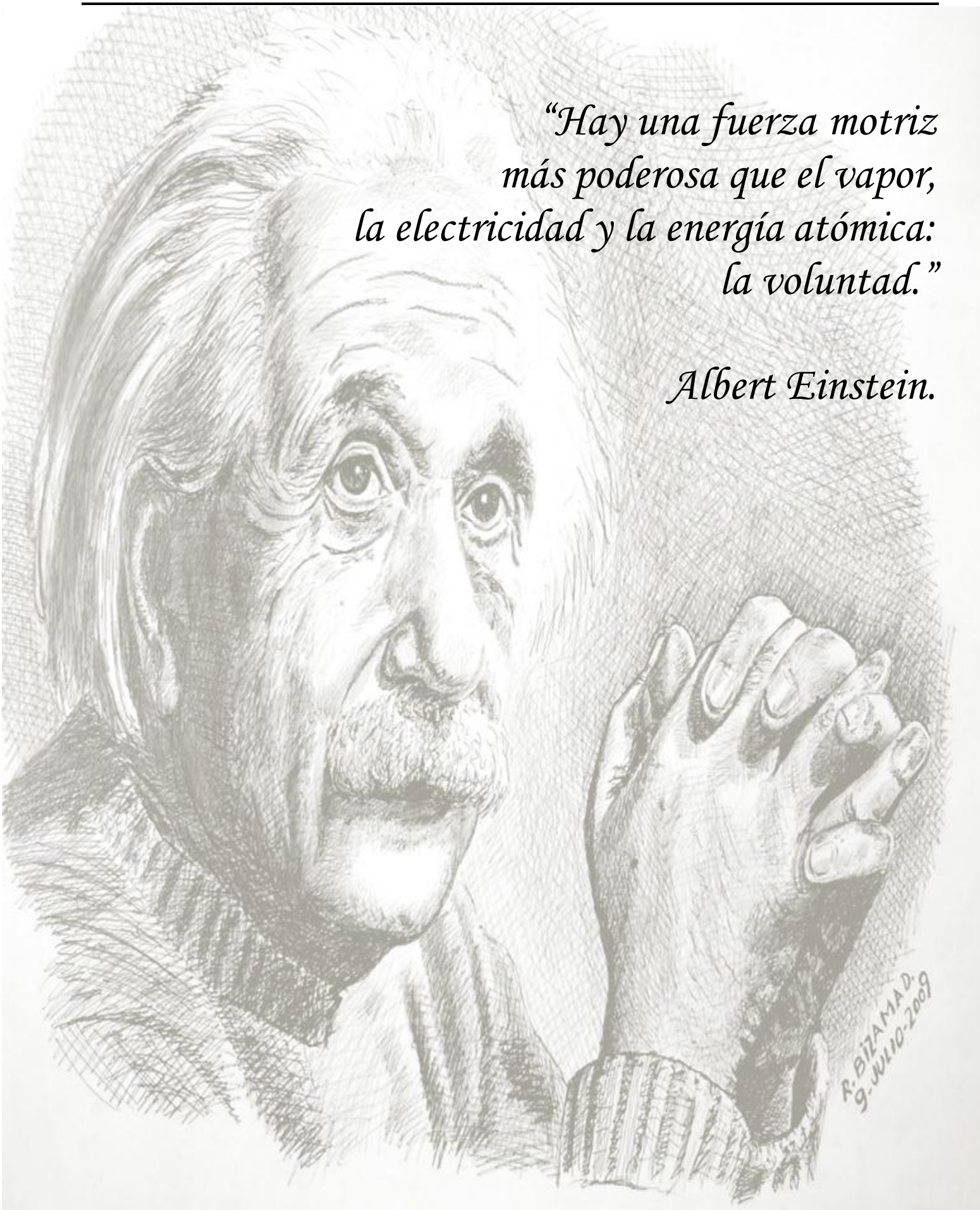
Autor

_____
Ing. Pedro Rodríguez Samon

Tutor

_____
Ing. Michel David Suárez

Tutor



"Hay una fuerza motriz
más poderosa que el vapor,
la electricidad y la energía atómica:
la voluntad."

Albert Einstein.

Una hoja no bastaría para intentar agradecerles a todas las personas que de una forma u otra me han apoyado en cada paso y etapa de mi vida, para llegar a este momento.

Tratando de no quedar en deuda con ninguna de ellas, quiero agradecer a:

A mis padres, por ser todo en mi vida,
mi faro, mi guía y apoyo en todo momento, por los concejos, regaños
y por inculcarme los valores que hoy hacen de mi la persona que soy.

A mis abuelos, por malcriarme a su gusto y al mío también,
por todo el amor incondicional y la educación que siempre me han dado.

A mi herma, por ser hermana, amiga, compañera
y verdugo cuando era niño jeje, por todo su apoyo
y amor incondicional y representar para mí un ejemplo a seguir.

A mi tetekeee por todo el amor, cariño,
y la paciencia que me ha tenido jeje,
por ser compañera, amiga, pero sobre todo la persona a la que amo.

A toda mi familia que siempre ha estado a mi lado, dándome apoyo, concejos y si hoy soy alguien en la vida se lo debo a todos ustedes

A mis tutores, Michel y Pedro, por su apoyo incondicional

A todos mis amigos y compañeros de batalla en estos 6 años,
Ernesto, el Yasma, el Yole,
a los compañeros de aula, de estudios, de malanoche, de fiestas, de locuras.

A todos los que de una forma u otra contribuyeron a hacer realidad el mayor de mis sueños. A todos GRACIAS...!!!

A mi papá y mi mamá por el amor y apoyo incondicional durante todos mis estudios y porque gracias a ellos soy la persona que soy.

A mi herma, mis abuelos, a toda mi familia por todo el amor brindado.

A mis tutores por el apoyo incondicional durante el desarrollo de la tesis.

A mis amistades, a los que están y los que no.

El estilo arquitectónico *Transferencia de Estado Representativo* (*REST*, por sus siglas en inglés) es una tendencia marcada en la actualidad para el desarrollo de sistemas que usan servicios web. Este trabajo de diploma surge de la necesidad de definir un mecanismo para el modelado de las aplicaciones *RESTful* en el Proceso Unificado de Desarrollo de Software (*RUP*, por sus siglas en inglés). La investigación parte del estudio valorativo del estado del arte, en el cual se abordan conceptos importantes relacionados con el ámbito de: el proceso de modelado de software; la metodología de desarrollo *RUP* y el proceso de modelado en la misma; y por último el estilo arquitectónico *REST,* así como el modelado de aplicaciones desarrolladas sobre él. Como solución se propone una extensión al Lenguaje Unificado de Modelado (*UML*, por sus siglas en inglés) con el fin de lograr una notación estándar para el modelado de aplicaciones *RESTful*. Permitiendo que cualquier usuario que desee desarrollar aplicaciones de este tipo, haga uso de la propuesta sin dificultad y pueda contar con una documentación legible y normalizada de todo el proceso de desarrollo de la aplicación.

Palabras clave:
modelado de software, *RESTful*, servicios web.

ÍNDICE

INTRODUCCIÓN

En los primeros años de la informática la programación se consideraba un arte y se desarrollaba como tal. Con el fin de reducir la complejidad que implicaba para el hombre comprender el lenguaje de los ordenadores, se desarrollaron mecanismos que garantizaron dar a las computadoras órdenes cada vez más cercanas al razonamiento humano y al lenguaje natural, fortaleciendo así, el desarrollo de *‹‹software››*[1].

Aunque el proceso no resulte sencillo, se puede decir que desarrollar ‹‹*software*›› no es más que construirlo simplemente mediante su descripción. Desde sus inicios, los programadores han encapsulado sus conceptos en diversos tipos de dibujos o más ampliamente de modelos. En la actualidad, la comunidad del ‹‹*software*›› precisa comunicar sus modelos, no sólo entre los miembros de un proyecto, sino a todas las personas involucradas en él y con el paso del tiempo, a los desarrolladores de futuras generaciones. Por esta razón se necesita un lenguaje no sólo para comunicarse con otros, sino para proporcionar un marco en el que desarrolladores individuales puedan pensar y analizar, además, éstos no pueden retener todo el contenido en sus cabezas durante meses o años, tienen que registrarlo sobre papel o electrónicamente.

Esta es una buena razón para considerar el proceso de modelado de ‹‹*software*›› como una acción fundamental dentro del desarrollo del mismo, ya que al especificar, visualizar y documentar los elementos de los sistemas informáticos como un plano para su construcción, así como describir tanto aspectos conceptuales como concretos de la aplicación, simplifica el complejo proceso de análisis y diseño del *‹‹software››* (1).

El Lenguaje Unificado de Modelado (*UML*, por sus siglas en inglés) permite a los programadores visualizar los resultados de su trabajo en esquemas o diagramas estandarizados. El mismo está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios, proporcionando ventajas en la representación del ciclo de vida de la aplicación y de los artefactos específicos del proceso de desarrollo. *UML* permite una comunicación sencilla y rápida entre implementadores y clientes, por lo que no se necesitan conocimientos profundos de Ingeniería del Software[2] para que los clientes comprendan lo que los programadores muestran (1).

[1] En esta investigación resulta esencial precisar el significado de ciertos términos. En la medida de lo posible el significado se

[2] Según Pressman, Roger S. en Ingeniería del Software. Vol (I), define Ingeniería del Software como: Es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora y la documentación asociada requerida para desarrollar, operar (funcionar) y mantenerlos. Se conoce también como desarrollo de software o producción de software.

En el Centro de Informatización Universitaria (CENIA) perteneciente a la Universidad de las Ciencias Informáticas (UCI), se desarrolla el Gestor de Documentos Administrativos (GDA) eXcriba. Este sistema está desarrollado sobre el estilo arquitectónico Transferencia de Estado Representativo[3] (*REST*, por sus siglas en inglés) y se guía por el Proceso Unificado de Desarrollo de Software (*RUP*, por sus siglas en inglés). Sin embargo, durante el flujo de diseño, el modelado de los servicios del GDA eXcriba no fue posible, ya que el lenguaje *UML* no cuenta con una notación estándar que permita modelar los servicios *REST*. Como consecuencia de tal limitación tanto los modelos de diseño y de la arquitectura del sistema, resultan afectados en cuanto a su visualización, especificación, construcción y documentación. Además, esto imposibilita al equipo de desarrollo obtener una clara visión de lo que se quiere lograr, retrasando así el desarrollo y futuros mantenimientos del software. Todo esto acarrea insuficiencias y riesgos durante la Gestión de Proyecto[4], ya que las faltas mencionadas influyen sobre la estimación de tiempo para el desarrollo del sistema, afectando todo el proceso. La problemática antes planteada es consecuencia de la insuficiencia en la expresividad de *UML* para cubrir todo tipo de situaciones.

Ante esta problemática se hace inminente un estudio enmarcado en la aceptación de mecanismos y técnicas de modelado, que den paso a la calidad en el desarrollo de servicios o aplicaciones web orientadas a recursos. De esta forma, las aplicaciones *‹‹RESTful››* podrán contar con una documentación legible durante todo el proceso de desarrollo, siguiendo así las políticas y estándares para el correcto desarrollo del ‹‹*software*››. Partiendo de los acontecimientos anteriores se puede plantear el siguiente **problema científico**:

¿Cómo modelar aplicaciones *‹‹RESTful››* en el Proceso Unificado de Desarrollo de Software?

Constituye el **objeto de estudio** de la investigación el método de modelado en el Proceso Unificado de Desarrollo de Software y el **campo de acción** es el método de modelado en el Proceso Unificado de Desarrollo de Software de los sistemas *‹‹RESTful››.*

Como propuesta de solución al problema planteado anteriormente se define el siguiente **objetivo general:**

Proponer un mecanismo de modelado para los sistemas *‹‹RESTful››.*

[3] N. del T. En el contexto de esta investigación se toma como traducción al termino *Representational* del argumento *Representational State Transfer (REST)* la palabra Representativo, haciendo alusión a la cualidad que tenga un determinado objeto de ser representado mediante una figura, imagen o idea.

[4] La Gestión de Proyectos es una disciplina que permite la planificación y dirección de proyectos. La misma debe garantizar que los proyectos se desarrollen convenientemente y se obtengan óptimos resultados. Debe permitir el control de la evolución del proyecto y poder explicarlo de forma satisfactoria al equipo de trabajo y al cliente.

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

- definir un mecanismo en la propuesta que permita modelar las aplicaciones *‹‹RESTful››* en el Proceso Unificado de Desarrollo de Software,
- aplicar el resultado de la investigación en la documentación de las aplicaciones *‹‹RESTful››,*
- definir una guía descriptiva por la cual poder modelar las aplicaciones *‹‹RESTful››,*
- validar la solución propuesta.

La investigación se sustenta en la siguiente **idea a defender:**

La propuesta de un mecanismo de modelado para los sistemas *‹‹RESTful››,* mejorará la visualización, especificación, construcción y documentación de estos sistemas durante todo el Proceso Unificado de Desarrollo de Software.

Para la investigación fueron empleados los **métodos científicos** siguientes:

Métodos teóricos:

- El método histórico-lógico permitió realizar un estudio de sistemas homólogos en el ámbito internacional y nacional, para profundizar en el tema, sus características e importancia. Posibilitó un mejor análisis histórico de los procesos de modelado en sistemas *‹‹RESTful››.*

Métodos empíricos:

- La entrevista permitió conocer de mano del equipo de desarrollo del GDA eXcriba, las especificidades sobre las necesidades existentes en el modelado de aplicaciones *‹‹RESTful››.*
- El método centrado en la observación con carácter cualitativo permitió conocer en el equipo de desarrollo del GDA eXcriba, cuáles son las mayores dificultades existentes relacionadas con el modelado de sistemas *‹‹RESTful››.*

Posibles resultados:

La propuesta de un mecanismo de modelado para aplicaciones *‹‹RESTful››*, le permitirá a estos sistemas contar con una documentación legible y normalizada del proyecto.

Estructura del trabajo por capítulos

Para un mejor desarrollo, la investigación se compondrá de capítulos, los mismos estarán enfocados en los principales epígrafes a tener en cuenta durante el estudio. Los capítulos se componen como sigue:

Capítulo 1: Fundamentación teórica: se presentan los elementos teóricos que sirven de base a la investigación del problema planteado, analizando los principales conceptos relacionados con el objeto de estudio y campo de acción. Se realiza un estudio del estado del arte de las herramientas y tecnologías que se utilizarán en el desarrollo de la investigación.

Capítulo 2: Propuesta solución: se desarrolla la propuesta de solución al problema planteado, donde se describen cada una de las actividades que componen dicho proceso, se desarrolla una guía como notación estándar para el modelado de aplicaciones *‹‹RESTful››.*

Capítulo 3: Validación de la solución: se realiza una validación de la solución propuesta, usando para ello la variante del método experto *Delphi*.

En el presente capítulo se presentan los elementos teóricos que sirven de base a la investigación del problema planteado. Durante el desarrollo del mismo, se definen conceptos clave que darán paso a una mejor comprensión de los temas tratados en toda la investigación. Se profundizará en temas necesarios a tener en cuenta para la solución de la investigación, cubriendo de esta manera los aspectos en los que se centra el objeto de estudio así como el campo de acción del presente trabajo de diploma.

Para iniciar un enfoque teórico sobre el proceso de modelado, se hace necesario comenzar desde la base que dio surgimiento a este proceso y cuáles fueron las condiciones propicias para su desarrollo. Tomando lo antes planteado como punto de partida, se puede decir que la necesidad en el hombre de representar formalmente una realidad deseada, propició el impulso de técnicas normalizadas buscando simplicidad y eficiencia a la hora de comunicar aspectos muy difíciles de transmitir. A lo largo de la historia, el hombre ha creado diversas formas de comunicación, dígase a través de sonidos, señales, gestos, gritos o dibujos. Estas señales, símbolos y signos, se empezaron a plasmar por medios gráficos en piedras o paredes, surgiendo de esta forma la técnica de modelado.

1.1. Proceso de modelado

Según el Dpto. de Informática de la Universidad Carlo III de Madrid, un modelo no es más que una representación simplificada de la realidad, que recogiendo aspectos de interés, promueve entendimiento para comprender, describir, predecir y responder preguntas. Obteniendo como resultado final, una representación gráfica de un conjunto de elementos interconectados (2).

La comunicación a través del modelado, constituye un mecanismo sencillo y rápido para lograr la representación de un problema en una escala estándar, resultando así comprensible por todo el personal. Debe ser capaz de expresar de manera clara y concreta lo que se quiere obtener, sin necesidad de sobrecargar de información el modelo.

Tipos de modelado

Según la complejidad del elemento a modelar, se tiene en cuenta los diferentes tipos de modelos (2):

- **Abstracto:** simplificación de la realidad: divide y vencerás.
- **Comprensible**: expresado de tal forma que se pueda entender fácilmente.
- **Preciso**: representa fielmente el sistema modelado.
- **Predictivo**: se puede utilizar para obtener conclusiones correctas sobre el sistema.

- **Barato**: más económico que construir y estudiar el propio sistema.

Cada modelo se emplea para proponer, plantear y enunciar los elementos a modelar con el nivel de detalle deseado. Sirve también para aplicar las reglas que definen el proceso a realizar y para comprender mejor el problema representado. El modelado de ‹‹*software*›› se ha considerado por mucho tiempo como una de las mejores formas de construir un sistema, ya que incluye aspectos conceptuales tales como; proceso de negocio y funciones de la aplicación. En el modelo se definen aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de ‹‹*software*›› reutilizables.

1.1.1 Modelado de software

El modelado de ‹‹*software*›› es el diseño estático o dinámico de una aplicación, el mismo representa cómo los componentes se relacionan entre sí antes de codificarla. Según el Dpto. de Informática de la Universidad Carlo III de Madrid, el modelado de ‹‹*software*›› resulta una abstracción de un sistema, es decir, una simplificación completa y consistente del sistema real que sirve para comprenderlo mejor. Puede resultar formal o informal dependiendo de la importancia del contexto expuesto (2).

Principios del modelado de software

Se deben tener en cuenta cuatros principios para un correcto modelado (2):

- elegir los modelos a utilizar que sirvan al propósito deseado,
- los modelos pueden ser expresados en distintos niveles de precisión,
- mientras más coherente sea un modelo con la realidad mejor será el resultado,
- cualquier sistema no trivial se aborda mejor con varios modelos casi independientes.

Es importante destacar la importancia del proceso de modelado durante el desarrollo de ‹‹*software*››, ya que el mismo permite detectar errores u omisiones en el diseño antes de comprometer recursos para la implementación. A través del modelado se puede analizar y experimentar para mitigar riesgos durante el proceso de desarrollo de ‹‹*software*››. Además, permite la comunicación con el personal de conocimientos ajenos a dicho proceso y sirve como guía para la implementación en el mismo.

1.1.2 Lenguajes de modelado

El desarrollo de *‹‹software››* resulta un proceso complejo, por lo que el equipo de desarrollo debe integrarse en forma conjunta y encaminarse en un objetivo común. Para lograr este objetivo, es necesario el apoyo en lenguajes y herramientas de modelado de ‹‹*software*››, como idioma común para la comunicación entre programadores y analistas del equipo, logrando la cohesión necesaria para una

lógica comprensión del sistema.

La investigación se enfocará en lenguajes de modelado como *UML* y *BPMN*, por ser dos de los más usados en la construcción de aplicaciones que desarrollan una lógica de negocio.

Lenguaje Unificado de Modelado (UML)

Según afirma *Jacobson* en el libro "*El Proceso Unificado de Desarrollo de Software"*, *UML* es un lenguaje muy utilizado en la industria del ‹‹*software*›› para su modelado. El mismo ayuda a los profesionales a visualizar, comunicar y aplicar sus diseños (1).

UML sirve para especificar, visualizar y documentar esquemas de sistemas de *‹‹software››* orientado a objetos, proporcionando a los desarrolladores un vocabulario que incluye tres categorías (1):

- **Elementos:** existen cuatro tipos de elementos: estructurales, de comportamiento, de agrupación y de anotación.
- **Relaciones:** existen tres tipos de relaciones: de dependencia, asociación y de generalización.
- **Diagramas:** *UML* proporciona nueve tipos de diagramas: diagrama de casos de uso, de clase, de objeto, de secuencia, de colaboración, de estados, de actividad, de componentes y de despliegue.

UML puede ajustarse a un sistema, proyecto o proceso de desarrollo específico si es necesario, para esto, proporciona mecanismos de extensibilidad, los cuales permiten a los usuarios refinar su sintaxis y semántica (3), incluyendo:

- **Estereotipos:** proporcionan una forma de definir nuevos elementos extendidos.
- **Valores etiquetados:** proporcionan una forma de definir nuevas propiedades de elementos ya existentes.
- **Restricciones:** proporcionan una forma de imponer reglas sobre elementos y sus propiedades.

Una extensión de *UML* comienza con una descripción breve, luego lista y describe todos los estereotipos, valores etiquetados, y restricciones de la extensión. Además de estos elementos, contiene un juego de reglas de "buenas formas" que se usan para determinar si un modo es consistente con sí mismo.

Notación para el Modelado de Proceso de Negocios (BPMN)

BPMN es un estándar cuyo principal objetivo según la Iniciativa de Administración para el Proceso de Negocio (*BPMI*, por sus siglas en inglés) es: *"proporcionar una notación fácilmente comprensible por todos los usuarios del negocio, desde los analistas, los programadores, hasta aquellos que gestionarán los procesos"* (4). Otros objetivos importantes que plantea esta especificación son:

- crear puentes entre el diseño de los procesos de negocio y la implementación del proceso,
- que los lenguajes basados en *‹‹XML››* para describir procesos tengan una notación gráfica.

Esta notación fue diseñada para representar procesos de negocio y permite una especificación mucho más profunda y consistente que el resto de las notaciones. Es la notación que más patrones de flujo de trabajo soporta y se considera la notación a utilizar en proyectos de tipo *BPM/SOA* (4). Es importante tener en cuenta para la investigación que *BPMN* abarca únicamente los procesos de negocio, lo que significa que otro tipo de modelos relacionados como: estructura de la organización, recursos, modelos de datos, estrategias, reglas de negocio, entre otros, quedan fuera de la especificación.

Es evidente la aceptación y comprensión de *BPMN* por su especificación consistente para el análisis de proceso de negocio, pero no es solo en el modelado de dicho proceso en el que se enfoca la investigación. *BPMN* está orientado a analistas de negocio mientras que *UML* se enfoca en el desarrollo de ‹‹*software*››, cubriendo el modelado de negocio con sus diagramas de casos de uso del sistema y el diagrama de actividades. Según lo antes planteado y teniendo en cuenta que *UML* constituye el lenguaje que propone el Proceso Unificado de Desarrollo de Software para el modelado, siendo este el objeto de estudio de la presente investigación, se recomienda usar *UML* como lenguaje de modelado para la propuesta de solución, ya que sus diagramas abarcan todo el flujo de trabajo durante el desarrollo de ‹‹*software*››.

1.1.3 Herramientas para el modelado de software

En la actualidad es casi imposible el desarrollo de ‹‹*software*›› sin utilizar un proceso soportado por herramientas, ya que constituyen un elemento esencial a la hora de automatizar procesos repetitivos, mantener los elementos estructurados, gestionar grandes cantidades de información y sobre todo para guiar el desarrollo. Con poco soporte debido a la escasez de herramientas, el proceso debe sostenerse sobre gran cantidad de trabajo manual y será por tanto menos formal, resultando difícil mantener actualizados los modelos y la implementación. El epígrafe se centrará en el análisis de las principales herramientas de apoyo que soporten los lenguajes estudiados durante la investigación.

Rational Rose

Es una de las herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos. Su funcionalidad consiste en modelar un sistema para luego comenzar con la etapa de construcción. Abarca todo el ciclo de vida de un proyecto: concepción del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases. Los diseñadores pueden tomar ventajas de esta herramienta, porque permite que la salida de una fase sea la entrada de la

próxima que está por venir. Puede generar código en *Java*, *C++*, *Visual Basic. Ada, Corba* y *Oracle* (5). Esta herramienta permite que el equipo de desarrollo entienda mejor el problema, que identifique las necesidades del cliente en forma más efectiva y comunique la solución propuesta de forma clara. También permite completar una gran parte de los flujos de *RUP* como son: captura de requisitos, análisis y diseño, implementación, entre otros.

Visual Paradigm

Es una herramienta profesional que soporta completamente el ciclo de vida del desarrollo de *‹‹software››* como son: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Soporta el lenguaje de modelado *UML* facilitando una rápida construcción de aplicaciones de calidad y mejoras. Permite realizar todo tipo de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta *UML CASE* proporciona además, una guía, demostraciones interactivas y proyectos. Permite al grupo de desarrollo lograr un modelo exclusivo, así como la formación y despliegue de un proceso de desarrollo. Es capaz de resistir varios lenguajes tanto de código como de ingeniería inversa tales como: *Java, C + +, CORBA IDL, PHP, ‹‹XML›› Schema, Ada y Python*. Además, para la generación de código como: *C #, VB. NET, Object Definition Language (ODL), Flash ActionScript, Delphy, Perl, Objective-C,* y *Ruby* (6).

Las herramientas de modelado se convierten en partes fundamentales dentro del desarrollo de todo sistema informático, ya que son las encargadas de modelar los requerimientos del sistema, la funcionalidad y la interacción con otros módulos. Se emplean para automatizar actividades, de manera completa o parcial, para incrementar la productividad y la calidad así como para reducir el tiempo de desarrollo. Al elegir la herramienta adecuada para la investigación que apoye el modelado de aplicaciones *‹‹RESTful››*, surge una pregunta: ¿Qué se debe tener en cuenta? Como respuesta, el estudio se sustenta en optar por el modelo de desarrollo y distribución libre, esto no solo implica el uso libre de dichas herramientas, sino también su modificación para adaptarlas a casos particulares, así como la exploración de sus mecanismos de funcionamiento para utilizarlos en futuros desarrollos. De esta forma, se obtiene una ventaja significativa frente a aquellas empresas que al basar su negocio en el modelo propietario o cerrado, se privan de esta base de herramientas y conocimiento. Teniendo en cuenta lo antes planteado, la investigación se apoyara en la herramienta *Visual Paradigm* para el modelado, por ser una herramienta que permite hacer diagramas *UML*. La misma da la posibilidad de integrarse con entornos de desarrollo integrado como *NetBeans* y *Eclipse*, facilitando la generación automática de código así como la

sincronización entre modelos y código. Además posee una licencia comercial, con versión *free* para la comunidad y versiones de evaluación del resto (7).

1.1.4 Estándares para el modelado de software.

Arquitectura dirigida por modelos (por sus siglas en inglés, MDA)

Es un conjunto de estándares del Grupo de Gestión de Objetos (*OMG, por sus siglas en inglés*) que posibilitan la especificación de modelos y sus transformaciones en otros modelos y sistemas completos. *MDA* separa los problemas, de forma tal que los modelos orientados a aplicaciones sean independientemente utilizados a través de múltiples programaciones y viceversa (8).

La arquitectura dirigida por modelos no es más que un acercamiento al diseño de *‹‹software››*. La misma se ha concebido para dar soporte a la ingeniería dirigida a modelos, ya que proporciona un conjunto de guías para estructurar especificaciones expresadas como modelos.

Meta-Object Facilty (por sus siglas en inglés, MOF)

Es el estándar *OMG* para *‹‹metadatos››* y manejo de modelos, encontrándose en el corazón de *MDA*. Especifica cómo definir *‹‹metamodelos››* utilizando un sub-conjunto *UML* y genera esquemas *‹‹XML››* para intercambio e interfaces de programación de aplicaciones para manipular los modelos actuales (ej., diseños *UML*). Ha sido extendido también para especificar servicios para manejo de modelos y sus elementos, como identidad, ciclo de vida, versiones, vistas y transformaciones (9).

Es válido destacar que el desarrollo guiado por modelo no es más que una aproximación al desarrollo de *‹‹software››* en el que el enfoque y los artefactos principales son los modelos y no los programas. Esto implica la generación automática de programas a partir de dichos modelos, utilizando los lenguajes de modelado como herramientas para la implementación. Trayendo como ventaja que los conceptos de modelado están menos ligados a las tecnologías de implementación y más cerca del dominio del problema.

1.2. Proceso Unificado de Desarrollo de Software (RUP)

RUP es una metodología pesada de desarrollo de *‹‹software››,* cuyo propósito es la producción eficaz de un sistema informático que cumpla con los requisitos del cliente. Junto con *UML*, constituye la técnica estándar más utilizada para el análisis, desarrollo y documentación de sistemas orientados a objetos (1).

Esta metodología se encuentra preparada para el desarrollo de grandes y complejos proyectos, se puede especializar para gran variedad de sistemas de *‹‹software››*, distintas áreas de aplicación, tipos

de organizaciones, niveles de actitud y tamaños de proyecto. Se caracteriza por ser un proceso iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso (1). *RUP* incluye artefactos que son los productos tangibles del proceso como por ejemplo, el modelo de casos de uso, el código fuente, entre otros, y roles que son el papel que desempeña una persona en un determinado momento. Una persona puede desempeñar distintos roles a lo largo del proceso. Cada fase concluye con un hito bien definido, en cada uno de estos se deben tomar acuerdos y decisiones, garantizando el cumplimiento de los objetivos y metas antes de la transición a la nueva fase.

1.3. El modelado dentro del Proceso Unificado de Desarrollo de Software

RUP proporciona un conjunto de modelos haciendo más claro el sistema para todos los trabajadores, incluyendo clientes, usuarios y jefes de proyectos, satisfaciendo así la necesidad de información de los trabajadores. En esta metodología el modelado del sistema a construir debe describir las interacciones entre la aplicación y los que la rodean, por tanto, aparte del *‹‹software››* que se está modelando, se deben incluir también elementos que describan las partes relevantes de su entorno (ej., actores) (1).

La mayoría de los modelos en la metodología son un subconjunto de *UML*, por ejemplo, el modelo de casos de uso comprende a los casos de uso y los actores. Por otra parte, el modelo de diseño describe los subsistemas y las clases del sistema, así cómo interactúan para llevar a cabo la realización de los casos de uso. El modelo de casos de uso es una vista externa del sistema, que captura los usos de la aplicación, mientras que el modelo de diseño es una vista interna que representa la construcción de la misma.

RUP utiliza el Lenguaje Unificado de Modelado para la visualización, especificación, construcción y documentación de sistemas de *‹‹software››*. Según *Jacobson* en el libro "*El Proceso Unificado de Desarrollo de Software"* los artefactos más interesantes utilizados en *RUP* son los modelos (1). Sin embargo existen dependencias entre algunos de ellos. La Figura 1 ilustra las dependencias entre el modelo de casos de uso con el resto de los modelos.

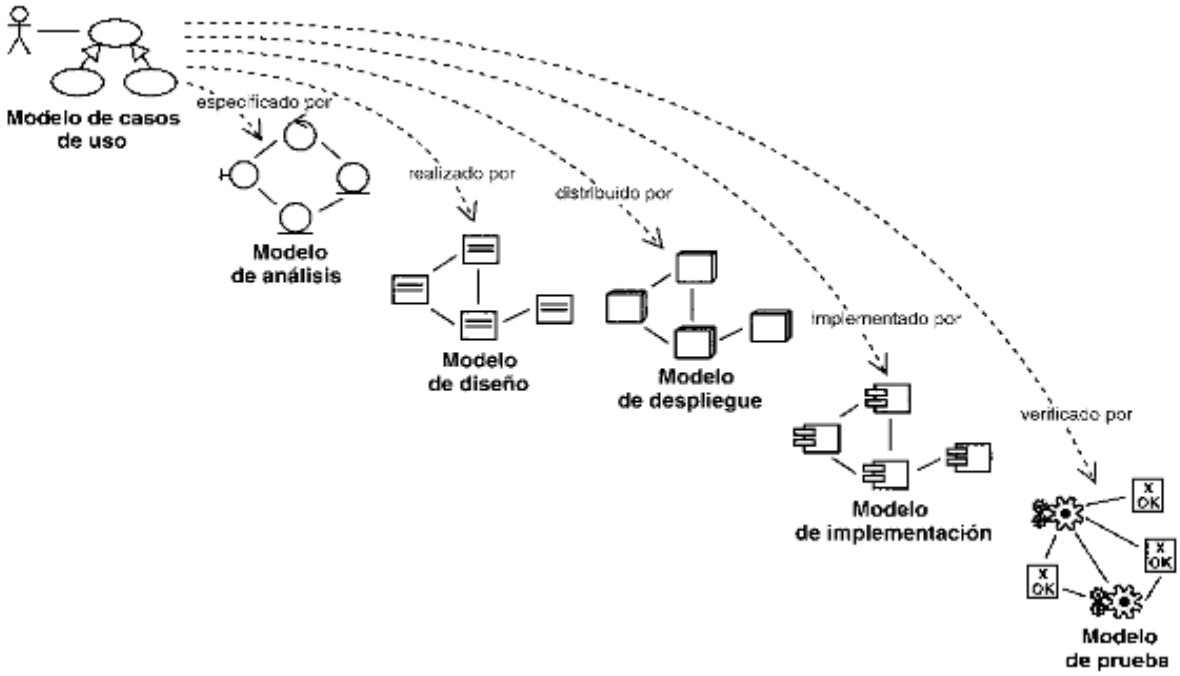


***Figura 1: Modelo del Proceso Unificado de Desarrollo de Software* (1).**

Cada modelo dentro de *RUP* define aspectos que influyen en la construcción del sistema, dígase:

Modelo de casos de uso: representa todos los casos de uso del sistema, así como su relación con los usuarios.

Modelo de análisis: refina los casos de uso con más detalles y establece la asignación inicial de funcionalidad del sistema a un conjunto de objeto que proporcionan el comportamiento.

Modelo de diseño: representa la estructura estática del sistema en la forma de subsistemas, clases e interfaces, así como los casos de usos reflejados como colaboraciones entre los subsistemas, clases, e interfaces.

Modelo de implementación: representa los componentes –que representan el código fuente– y la correspondencia de sus clases.

Modelo de despliegue: define los nodos físicos –ordenadores– y la correspondencia de los componentes con dichos nodos.

Modelo de prueba: describe los casos de prueba que verifican los casos de uso.

Todos estos modelos están relacionados y en su conjunto representan el sistema como un todo. La aplicación también debe tener una representación de la arquitectura así como un modelo del dominio o modelo de negocio que describa el contexto del negocio en el que se halla el sistema. El desarrollo de *‹‹software››* es un proceso de construcción de modelos, utilizando distintos tipos para describir todas

las perspectivas diferentes del sistema durante cada flujo de trabajo dentro de *RUP*. Dicha metodología define cinco flujos de trabajos fundamentales (1):

- Requisitos,
- Análisis,
- Diseño,
- Implementación,
- Prueba.

Requisitos

Según *Jacobson,* la captura de Requisitos es el proceso de averiguar en circunstancias difíciles lo que se debe construir (1). Aunque la principal técnica es usar el lenguaje del cliente para la captura de requisitos el proceso resulta complicado, debido a que los clientes constituyen una fuente imperfecta de información. Un elevado número de los mismos no saben qué partes de su trabajo pueden transformarse en ‹‹*software*››, por lo que con frecuencia no conocen cuales son los requisitos ni tampoco cómo especificarlos de una manera precisa. El principal objetivo de este flujo de trabajo es guiar el desarrollo hacia el sistema correcto (1). Esto se logra mediante una descripción de los requisitos del sistema –las condiciones o capacidades que el sistema debe cumplir– suficientemente buena para llegar a un acuerdo entre el cliente y los desarrolladores sobre lo que el sistema debe hacer y lo que no. Para la captura de los requisitos de manera eficaz, los analistas se apoyan en un conjunto de técnicas y artefactos que permiten obtener una visión del sistema, posibilitando avanzar a los flujos de trabajo siguientes.

El artefacto fundamental utilizado durante la captura de requisitos es el modelo de casos de uso, el mismo incluye los casos de uso y los actores. También puede haber otros tipos de artefactos, como el prototipo de interfaz de usuario. La Figura 2 muestra los trabajadores y artefactos pertenecientes al flujo de trabajo de Requisitos y la Figura 3 representa el comportamiento dinámico de dicho flujo.

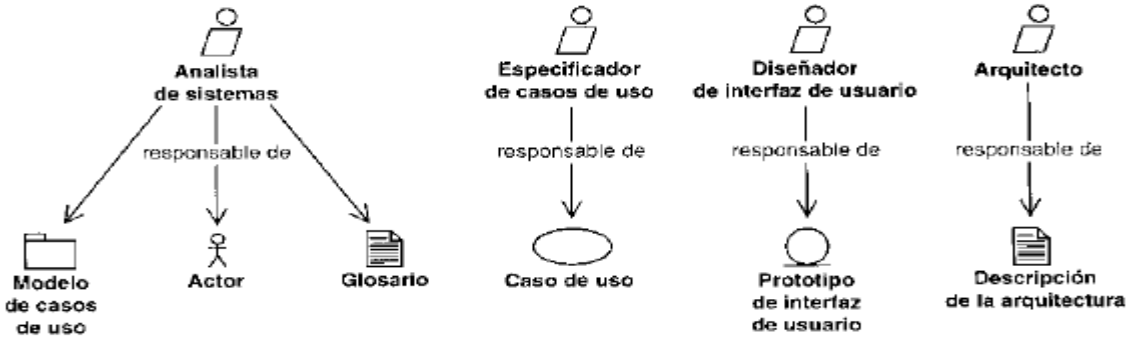


***Figura 2: Trabajadores y artefactos implicados en la captura de requisitos mediante casos de uso* (1).**

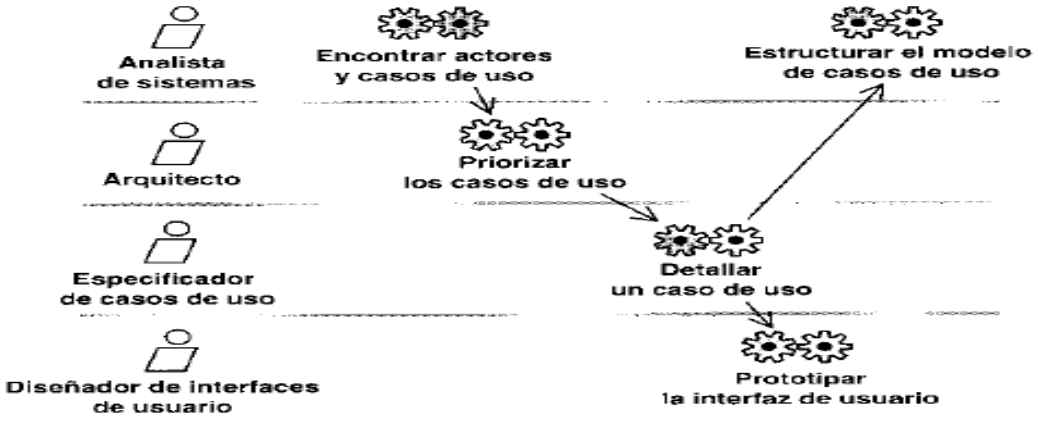


***Figura 3: Trabajadores y sus actividades durante la captura de requisitos mediante casos de uso* (1).**

En las distintas fases e iteraciones de *RUP,* la captura de requisitos así como los artefactos resultantes en dicho flujo de trabajo adquieren diferentes formas: durante la fase de inicio, los analistas identifican la mayoría de los casos de uso detallando los más importantes, limitando el sistema y el alcance del proyecto; durante la fase de elaboración, los analistas capturan la mayoría de los requisitos restantes para que los desarrolladores puedan estimar el esfuerzo de desarrollo requerido; los requisitos restantes se capturan e implementan durante la fase de construcción; en la fase de transición casi no hay captura de requisitos a menos que exista algunos cambios en los mismos.

Durante el estudio del flujo de trabajo Requisitos*,* en las actividades, no se evidencian afectaciones que guarden relación con el problema de la investigación, e interrumpan un correcto desarrollo de aplicaciones *‹‹RESTful››.*

Análisis

En este flujo de trabajo se analizan los requisitos descritos durante el flujo de trabajo Requisitos. Los mismos se refinan y se estructuran para conseguir una comprensión y descripción más precisa, la cual sea fácil de mantener y que ayude a estructurar el sistema entero incluyendo su arquitectura. El propósito fundamental de este flujo de trabajo es analizar los requisitos con mayor profundidad, con la diferencia de que se puede utilizar el lenguaje de los desarrolladores para describir los resultados. Esto posibilita un mayor razonamiento sobre los aspectos internos del sistema. El artefacto principal del flujo es el modelo de análisis, el mismo puede considerarse una primera aproximación al modelo de diseño y posibilita (1):

- una especificación más precisa de los requisitos que la obtenida por el flujo de trabajo anterior,

- un mayor formalismo y ser utilizado para razones sobre los funcionamientos internos del sistema,
- estructurar los requisitos de modo que facilita su comprensión, preparación y modificación, así como su mantenimiento en general.

***Figura 4: Trabajadores y artefactos implicados en el flujo de trabajo Análisis* (1).**

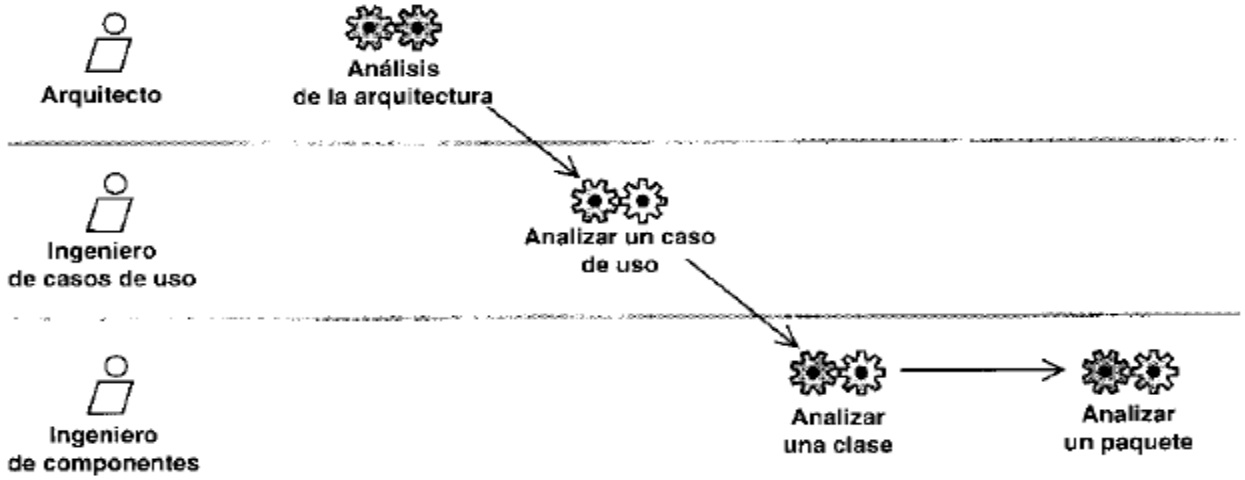


***Figura 5: Trabajadores y actividades implicadas en el flujo de trabajo Análisis* (1).**

Actividad: análisis de la arquitectura:

El propósito de esta actividad es esbozar el modelo de análisis y arquitectura mediante la identificación de paquetes del análisis, así como las clases evidentes del análisis y requisitos especiales comunes.

Actividad: analizar un caso de uso:

Se analiza un caso de uso para (1):

- identificar las clases del análisis cuyos objetos son necesarios para llevar a cabo el flujo de sucesos del caso de uso,
- distribuir el comportamiento del caso de uso entre los objetos del análisis que interactúan,
- capturar requisitos especiales sobre la realización del caso de uso.

Actividad: analizar una clase (1):

Se analiza una clase con el objetivo de:

- identificar y mantener las responsabilidades de una clase del análisis, basada en su papel durante la realización del caso de uso,
- identificar y mantener los atributos y relaciones de la clase del análisis,
- capturar requisitos especiales sobre la realización de la clase del análisis.

Actividad: analizar un paquete:

Se analiza un paquete con el objetivo de (1):

- garantizar que el paquete del análisis es tan independiente de otros paquetes como sea posible,
- garantizar que el paquete del análisis cumple su objetivo de realizar algunas clases del dominio o casos de uso,
- describir las dependencias de forma que pueda estimarse el efecto de los cambios futuros.

Durante el estudio del flujo de trabajo Análisis, así como las actividades realizadas durante el mismo, no se reflejan afectaciones relacionadas con el problema planteado en la investigación, por lo que no se evidencian inconvenientes para el desarrollo de aplicaciones *‹‹RESTful››.*

Diseño

En el diseño se modela el sistema en toda su forma, incluida la arquitectura, de manera que soporte todos los requisitos incluyendo los no funcionales y otras restricciones. La principal entrada en el diseño lo constituye el modelo de análisis, el mismo proporciona una comprensión detallada de los requisitos. El principal propósito del diseño consiste en adquirir una comprensión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia, de interfaz de usuario, así como de gestión de transacciones, entre otras (1).

El diseño pretende crear una entrada apropiada y un punto de partida para actividades de implementación en la siguiente fase, capturando los requisitos o subsistemas individuales, interfaces y clases. Debe ser capaz de descomponer los trabajos de implementación en partes manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo. El modelo de diseño es el artefacto más relevante dentro del presente flujo de trabajo estudiado, el mismo sirve de abstracción y de entrada para la implementación del sistema en el flujo siguiente.

***Figura 6 : Trabajadores y artefactos implicados en el flujo de trabajo Diseño* (1).**

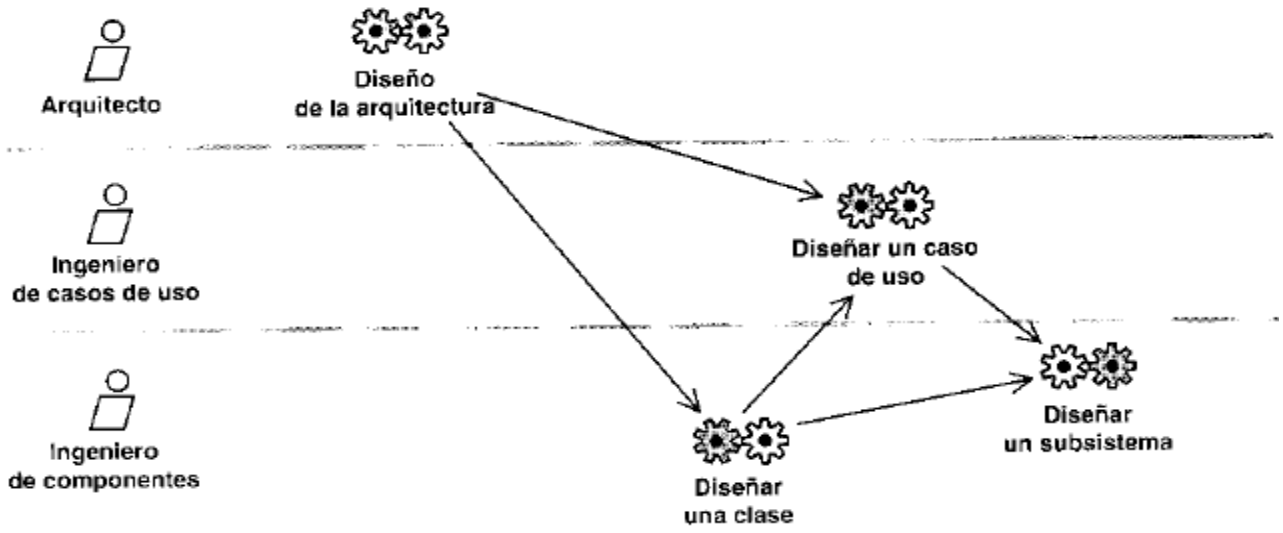


***Figura 7: Trabajadores y actividades implicadas en el flujo de trabajo Diseño* (1).**

Actividad: diseño de la arquitectura:

El objetivo de esta actividad es esbozar los modelos de diseño y despliegue, así como la arquitectura mediante la identificación de los siguientes elementos (1):

- nodos y sus configuraciones de red,
- subsistemas y sus interfaces,
- clases significativas del diseño para la arquitectura, como las clases activas,
- mecanismos de diseños genéricos que tratan requisitos comunes, como los requisitos especiales sobre persistencia, distribución, rendimiento y demás.

En esta actividad los arquitectos consideran distintas posibilidades de reutilización, como la de partes de sistemas parecidos o de productos ‹‹*software*›› generales. El arquitecto en esta actividad también

mantiene, refina y actualiza la descripción de la arquitectura, así como las vistas arquitectónicas de los modelos de diseño y despliegue.

Actividad: diseño de un caso de uso:

El diseño de un caso de uso cumple con los objetivos de (1):

- identificar las clases del diseño y/o los subsistemas cuyas instancias son necesarias para llevar a cabo el flujo de sucesos del caso de uso,
- distribuir el comportamiento del caso de uso entre los objetos del diseño que interactúan entre los subsistemas participantes,
- definir los requisitos sobre las operaciones de las clases del diseño y/o sobre los subsistemas y sus interfaces,
- capturar los requisitos de implementación del caso de uso.

En el estudio de esta actividad como parte del objeto de estudio en el que se enmarca el trabajo de diploma a desarrollar, se evidencia un inconveniente para el desarrollo de aplicaciones *‹‹RESTful››.* Durante la identificación de las clases necesarias para llevar a cabo el flujo de sucesos de los casos de usos en el diseño, la investigación arroja que no se cuenta con las especificaciones necesarias, dígase estereotipos propios del lenguaje *UML*, para cubrir el modelado de los recursos, así como las restricciones impuestas del estilo *REST* para el desarrollo de aplicaciones orientadas a recursos. Esto dificulta la correcta realización del diagrama de clases del diseño, resultando insuficiente para cubrir los aspectos necesarios para el modelado de aplicaciones *‹‹RESTful››.* Afectando también la representación del comportamiento de los casos de uso con los restantes objetos del diseño. Resultando obstaculizada la actividad y el flujo de diseño en general.

Actividad: diseño de una clase:

El objetivo es crear una clase del diseño que cumpla su papel en las realizaciones de los casos de uso y los requisitos no funcionales que se aplican a estos, incluyendo el mantenimiento del diseño de clases en sí mismo y los siguientes aspectos de este (1):

- sus operaciones,
- sus atributos,
- las relaciones en las que participa,
- sus métodos,
- los estados impuestos,
- sus dependencias con cualquier mecanismo de diseño genérico,
- los requisitos relevantes a su implementación,

- la correcta realización de cualquier interfaz requerida.

En la presente actividad se encuentra el mismo inconveniente planteado anteriormente, la falta de estereotipos para modelar los recursos, así como todas las restricciones a tener en cuenta a la hora del desarrollo de aplicaciones *‹‹RESTful››*. Todo esto imposibilita el buen desarrollo de la actividad, haciendo defectuoso el flujo de diseño.

Actividad: diseño de un subsistema:

Los objetivos del diseño de un subsistema consisten en (1):

- garantizar que el subsistema resulte tan independiente como sea posible de otros subsistemas y/o de sus interfaces,
- garantizar que el subsistema proporcione las interfaces correctas,
- garantizar que el subsistema cumpla su propósito de ofrecer una realización correcta de las operaciones tal y como se define en las interfaces que proporciona.

Como ya se había mencionado en párrafos anteriores el principal resultado de este flujo de trabajo es el modelo de diseño. Pero el mismo se ve afectado por la carencia de elementos del lenguaje *UML* que permitan el modelado de aplicaciones *‹‹RESTful››*. Empezando por la carencia de estereotipos para modelar los recursos y la gestión de los mismos, se desata todo un conjunto de deficiencias: el diagrama de clases del diseño resulta insuficiente al no cubrir todos los aspectos necesarios para la gestión de recursos, por lo que el resto de las actividades se ven afectadas, imposibilitando la correcta realización del caso de uso y consigo todo el flujo de diseño en general. Como resultado final se obtiene una falla a partir del presente flujo de trabajo, impidiendo la continuidad al siguiente por resultar afectados los artefactos de entradas necesarios para la continuidad del proceso de desarrollo.

Implementación

El flujo inicia con los resultados del diseño y se implementa el sistema en términos de componentes. La mayor parte de la arquitectura es capturada durante el diseño, siendo el propósito de la implementación desarrollar la arquitectura y el sistema como un todo. El flujo de trabajo produce como resultado un refinamiento de la vista de la arquitectura del modelo de despliegue, donde los componentes ejecutables son asignados a nodos. Como principales objetivos de la implementación se tiene (1):

- planificar las integraciones de sistemas necesarias en cada iteración,
- distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue,
- implementar las clases y subsistemas encontrados durante el diseño. Las clases se implementan como componentes de fichero que contienen código fuente,

- probar los componentes individuales e integrarlos compilándolos y enlazándolos en uno o más ejecutables, antes de ser enviados para ser integrados y llevar a cabo las comprobaciones de sistema.

El resultado principal de la implementación es el modelo de implementación.

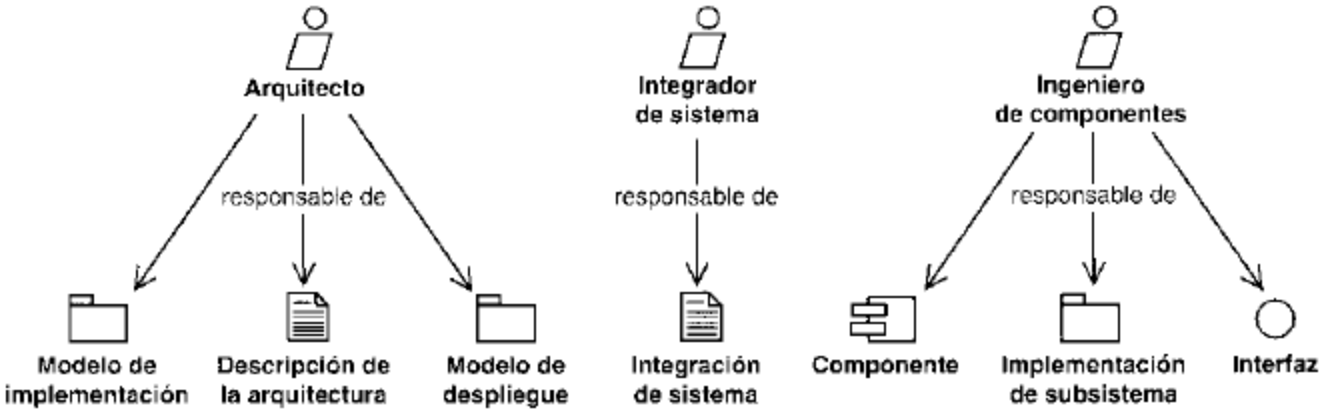


***Figura 8: Trabajadores y artefactos implicados en el flujo de trabajo Implementación* (1).**

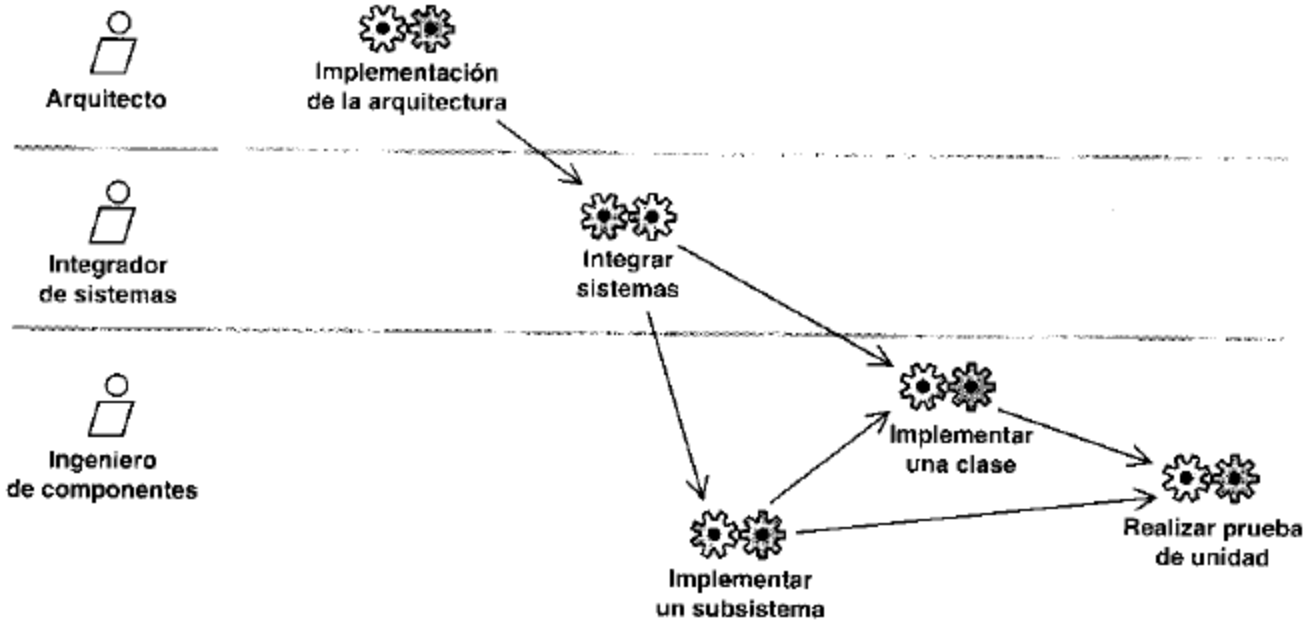


***Figura 9: Trabajadores y actividades implicadas en el flujo de trabajo Implementación* (1).**

Actividad: implementación de la arquitectura:

El principal objetivo es esbozar el modelo de implementación y su arquitectura mediante:

- la identificación de componentes significativos arquitectónicamente, tales como componentes ejecutables,
- la asignación de componentes a los nodos en las configuraciones de redes relevantes.

Actividad: integrar el sistema:

Los objetivos de esta actividad consisten en (1):

- crear un plan de integración que describa las construcciones necesarias en una interacción y los requisitos de cada construcción,
- integrar cada construcción antes de que sea sometida a pruebas de integración.

Actividad: implementar un subsistema:

Con la implementación de un subsistema se asegura que el mismo cumpla su papel en la construcción, tal y como se especifica en el plan de integración de la construcción. Se asegura que los requisitos implementados en la construcción y aquellos que afectan el subsistema sean implementados correctamente por componentes o por otros subsistemas.

Actividad: implementar una clase:

La implementación de una clase incluye (1):

- esbozo de un componente fichero que contendrá el código fuente,
- generación de código fuente a partir de la clase de diseño y de las relaciones en que participa,
- Implementación de las operaciones de las clases de diseño en forma de métodos,
- comprobación de que el componente proporciona las mismas interfaces que la clase de diseño.

El modelo de implementación es la entrada principal del flujo de trabajo de prueba que sigue al de implementación, pero dicho modelo no cuenta con la calidad requerida ya que el flujo se ve afectado por las deficiencias encontradas en su antecesor, dificultando la calidad del proceso e imposibilitando un avance en el desarrollo de aplicaciones *‹‹RESTful››*. La continuidad del proceso de desarrollo durante las siguientes etapas y flujos de trabajo se ve afectada, ya que no se cuenta con la calidad, elementos y procedimientos necesarios para lograr la disposición y aceptación requerida por el proyecto. Por lo antes planteado, se hace necesario centrar la presente investigación en el estudio del estilo arquitectónico *REST* en busca de una solución que permita la continuidad del proceso de desarrollo de aplicaciones *‹‹RESTful››* y que selle la brecha encontrada entre los flujos de trabajo de *RUP* estudiados en este epígrafe.

1.4. Estilo Arquitectónico Orientado a Recurso (ROA)

El uso de estilos arquitectónicos para el desarrollo de sistemas informáticos se ha convertido en un

mecanismo para acoplar la complejidad de dicho proceso. El presente epígrafe se enmarcará en el estudio de un estilo de arquitectura de *‹‹software››* orientado a recursos, el mismo se ha popularizado en los últimos años y es conocido como Transferencia de Estado Representativo, centrándose así la investigación en su campo de acción.

1.4.1. Transferencia de Estado Representativo (REST)

Según *Roy T Fielding*, *REST* es un estilo de arquitectura de *‹‹software››* para *‹‹sistemas hipermedias distribuidos››* como la web, derivado de varios de los estilos arquitectónicos basados en la red y en combinación con las restricciones adicionales que definen a un conector de interfaz uniforme (10).

Recursos en REST

Uno de los elementos clave en la arquitectura *REST* es el elemento recurso, el cual se define como cualquier información que pueda ser nombrada. De acuerdo con esta definición dada por *Fielding* (10), se puede tomar como ejemplo un documento o una imagen almacenada en uno o varios servidores sede, que será consumida posteriormente por los clientes. Un recurso tiene un identificador, un ‹‹*URI*›› asociado a él y puede tener también un ‹‹*metadato*›› que resulte útil para el consumo del mismo, (ej., la fecha de modificación).

Estándares de REST

El estilo hace uso de varios estándares:

- ‹‹*HTTP*››,
- ‹‹*URL*››,
- representación de los recursos: *‹‹XML››/‹‹HTML›› /GIF/JPEG,* entre otros,
- tipos ‹‹*MIME*››: *text/‹‹XML››, text/‹‹HTML››,* entre otros.

Principios de REST

Un servicio web *REST* sigue cuatro principios de diseño fundamentales:

- utiliza los métodos *‹‹HTTP››* de manera explícita (*GET, POST, DELETE, PUT)*,
- no mantiene estado,
- expone *‹‹URI››* con forma de directorios,
- transfiere *‹‹XML››*, *‹‹JavaScript Object Notation››* (*JSON*), entre otros.

1.4.2. Restricciones de REST

El estilo define un conjunto de restricciones o características que deben cumplir las arquitecturas de servicios web:

- cliente servidor,
- sin estado,
- caché,
- sistema de capas,
- interfaz uniforme,
- código bajo demanda.

Cliente-Servidor

Esta restricción separa lo que es competencia del cliente y el servidor, distinguiendo lo que concierne a la interfaz de usuario del almacenamiento de datos, así se puede acceder a esta desde múltiples plataformas (11). Mejora el rendimiento, pues simplifica los componentes del servidor al no tener que implementar las funcionalidades que van asociadas a la interfaz del usuario, permitiendo componentes independientes.

Sin estado

Esta restricción define que cada petición del cliente debe contener toda la información necesaria para que el servidor la comprenda sin necesidad de ningún dato almacenado previamente sobre el contexto de la comunicación (12). De esta manera, el estado de la sesión se guarda íntegramente en el cliente, mejorando así la visibilidad, eficiencia y rendimiento. La visibilidad mejora porque el servidor no tiene que apoyarse en otros sitios ni realizar más operaciones para comprender la naturaleza de una simple petición; la eficiencia aumenta porque se hace más fácil recuperarse de errores parciales; el rendimiento se ve también afectado porque al no necesitar almacenar los estados entre las peticiones, los componentes pueden liberar recursos más rápidamente. La desventaja de esta restricción es que puede empeorar el funcionamiento de la red, porque incrementa el tráfico de datos repetidos al enviar una serie de peticiones que no son almacenadas en el servidor para determinar el contexto de la comunicación. Además, poniendo el estado de la aplicación en el lado del cliente, se reduce el control para obtener un comportamiento consistente en la aplicación, por lo que se hace muy dependiente de un correcto desarrollo de la semántica en las distintas versiones del cliente.

Caché

Las respuestas a una petición deben poder ser etiquetadas como cacheable o no cacheable, si una respuesta es cacheable, entonces al cliente caché se le da permiso para reutilizar la respuesta más tarde si se hace una petición equivalente (13). La ventaja de añadir esta restricción, es que se evitarán determinadas peticiones al servidor, mejorando así la eficiencia y rendimiento, reduciendo el tiempo

medio de espera de una serie de interacciones, y como desventaja, puede inducir a un mal funcionamiento de una aplicación si los datos obtenidos de la caché difieren de los que se hubiesen obtenido realizando la petición directamente al servidor.

Interfaz uniforme

La principal característica que distingue a *REST* del resto de estilos de arquitecturas es el énfasis de usar una interfaz uniforme entre los componentes, a los cuales, aplicándoles los principios de generalidad de la ingeniería del ‹‹*software*››, simplifican la arquitectura del sistema global, mejorando así, la visibilidad de interacciones, separando la programación de los servicios que proporciona y animando al desarrollo independiente. Como desventaja, degrada la eficiencia, porque la información transferida está en una forma estandarizada y no según las necesidades que tenga la aplicación. La interfaz está diseñada para ser eficiente con transferencias de datos de hipermedias, resultando optimizada para la mayor parte de la web pero no siendo así para otras formas de arquitectura de interacción. Para obtenerla, el estilo define cuatro restricciones (14):

- identificación de recursos,
- manipulación de recursos a través de sus representaciones,
- mensajes auto-descriptivos,
- hipermedias como el motor del estado de la aplicación.

Sistema de capas

Para poder mejorar el comportamiento del rendimiento en Internet, se añade la restricción del sistema de capas (15). Esto garantiza una arquitectura compuesta por jerarquías, limitando así el comportamiento de los componentes al no acceder más allá de la capa con la que se está comunicando. Se simplifican los componentes, moviendo las funcionalidades de uso infrecuente hacia sistemas intermedios compartidos que pueden usarse para mejorar el rendimiento, permitiendo un balanceo de la carga de los servicios a través de múltiples redes y procesadores. La principal desventaja del sistema de capas es que añade cabeceras y retrasos al procesamiento de datos.

Código bajo demanda

Esta restricción es opcional, consiste en permitir a los clientes tener la funcionalidad de descargar y ejecutar código. Simplifica el lado del cliente al reducir el número de funcionalidades que tiene que tener implementadas al crearse, las cuales pueden ser descargadas posteriormente, aumentando así el nivel de extensión del sistema. La principal desventaja es que reduce la visibilidad y puede influir en la seguridad del sistema. Se considera que en los contextos en los que no sea útil ni necesario, lo

mejor será no incluirlo, porque puede acarrear más problemas que beneficios.

Para finalizar este epígrafe se puede decir que Transferencia de Estado Representativo no es más que un estilo arquitectónico inspirado en el funcionamiento de la web, cuyas restricciones son realmente las novedades que pretende aportar a las nuevas arquitecturas web.

1.5. Modelado del Estilo Arquitectónico Orientado a Recurso

Durante la investigación, analizando en las diferentes bibliografías para lograr una comprensión que arroje una solución acorde al problema planteado, se valoraron algunos ejemplos de modelado de Arquitectura Orientada a Recursos. El primero de estos artículos lleva por nombre "*Colaboración más inteligente para la Industria de la Educación"*. El mismo describe casos de uso de colaboración para instituciones de educación superior como las Universidades. En el artículo se exponen las funcionalidades mediante una interfaz *REST* basada en *‹‹HTTP››* que ayuda al usuario, tal como un profesor en una institución de educación o investigación, a mantener una lista de sus publicaciones creadas (16). Además, se describe la implementación de una API *REST* detallando a grande rasgos cada clase, método, así como sus dependencias. En una tabla se expresan aspectos como los métodos, *‹‹URI››*, parámetros y se especifican las respuestas para cada método como resumen de la API. Finalmente modela en un diagrama de secuencia un proceso de solicitud referente a la aplicación. El diagrama expone cada paso lógico y las acciones ejecutadas en cada una de las clases que intervienen durante la ejecución del método. Sin embargo, este es solo una parte del proceso de modelado de aplicaciones *‹‹RESTful››* que especifica la interacción entre los objetos del sistema mientras se ejecuta el método. El artículo estudiado no expone la estructura de clases de la API con sus relaciones y demás elementos principales del análisis y diseño.

El programador necesita de un modelo limpio que refleje de una manera lógica y semánticamente correcta aspectos como relaciones, restricciones, jerarquía, propiedades de clases y todo un conjunto de semblantes importantes para el correcto desarrollo de la aplicación. El segundo artículo valorado pertenece a *Silvia Schreier* de la Universidad de *Hagen*, llamado "*Modeling RESTful applications*" (17). Luego de un breve estudio enmarcado en los conceptos de metamodelos así como en las tecnologías y herramientas para su representación, expone un *‹‹metamodelo››* estructural de *REST*, así como un modelo jerárquico y conceptual de los recursos en el estilo arquitectónico. Finaliza con un *‹‹metamodelo››* sencillo de un ejemplo de un álbum de fotos según su estudio, como se ilustra en la Figura.10:

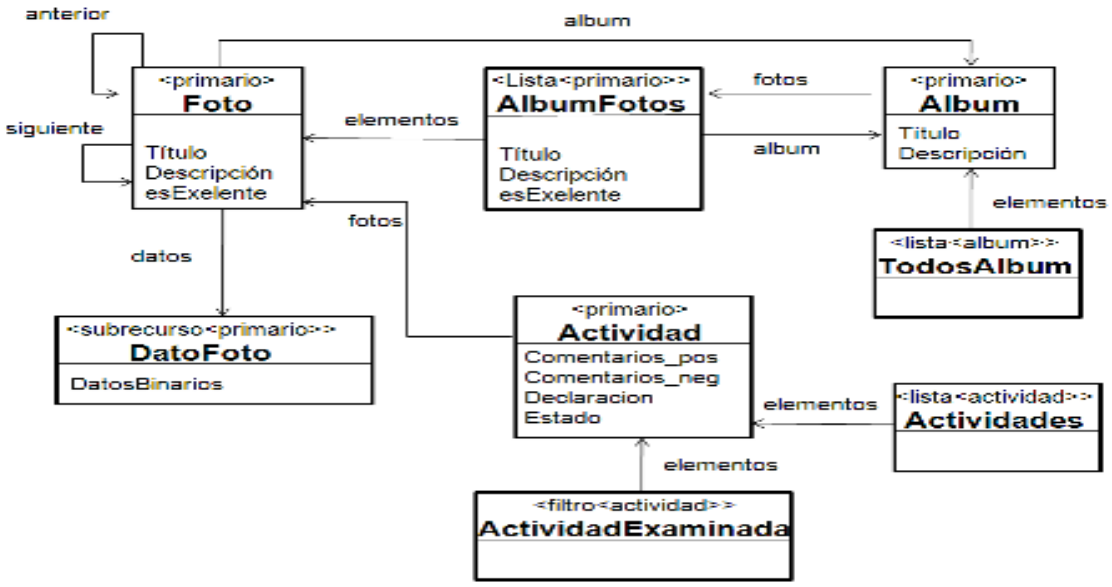


***Figura 10: Metamodelo de REST* (17).**

El modelo detalla de una manera sencilla la jerarquía de recursos como clases, sus relaciones, dependencias, atributos así como las acciones relacionadas a cada uno, de esta forma aproximándose a un modelo de clases del análisis en el desarrollo de *‹‹software››*. Aun así, la carencia de aspectos necesarios a especificar en el diagrama como; métodos soportados por los recursos, tipos de datos para representación de los mismos, *‹‹URL››* de los recursos y una serie de características importantes a tener en cuenta en los diagramas de clases al diseñar aplicaciones *‹‹RESTful››*, son elementos que imposibilitan el buen modelado de la aplicación, dificultándole al equipo de desarrollo la comprensión de las funcionalidades del sistema, por lo que el *‹‹metamodelo››* como representación o modelado de aplicaciones *‹‹RESTful››* no sería de gran ayuda para el equipo de desarrollo.

El tercer artículo valorado durante la investigación lleva por título "*RESOURCE ORIENTED MODELLING*" (18), el mismo profundiza en la Arquitectura Orientada a Recursos, así como en los diagramas de colaboración de *UML* para su modelado. En él se presenta un ejemplo sencillo de una orden de pizza a través de un diagrama de colaboración como muestra la Figura.11:

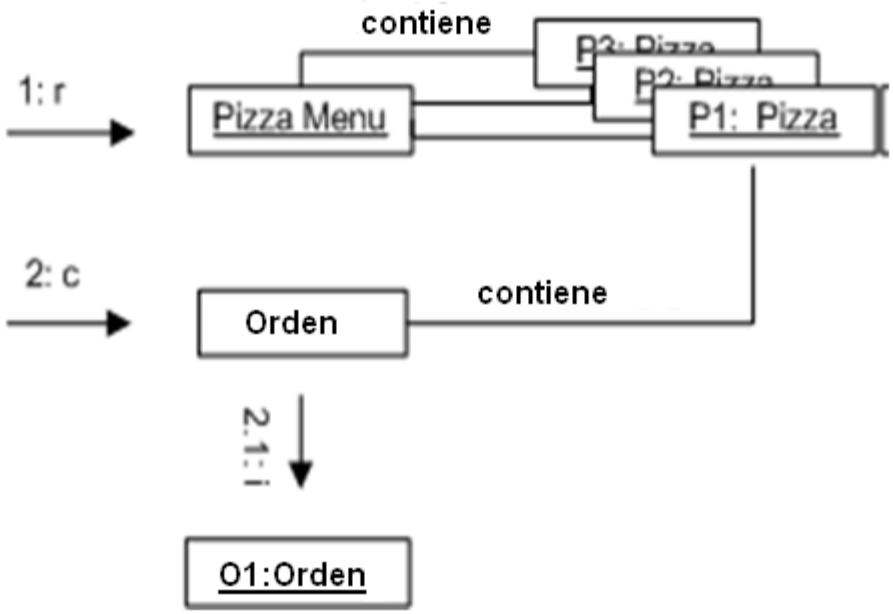


***Figura 11: Diagrama Orientado a Recurso* (18).**

Reflejando cada acción realizada en el escenario a través de mensajes, el diagrama expone la interacción entre los objetos del sistema. En el mismo se muestra una breve descripción de cómo se relacionan los recursos durante la ejecución de cada método, pasando por alto una vez más aspectos necesarios como los antes mencionados; "métodos *soportados por los recursos, tipos de datos para los elementos representados, ‹‹URL›› de los recursos y una serie de características importantes a tener en cuenta en los diagramas de clases al diseñar aplicaciones de este perfil".*

Las soluciones valoradas de una forma u otra cubren los escenarios en las que fueron expuestas. Pero ninguna satisface el problema en el que se centra la investigación, ya que no abarcan todos los aspectos significativos del estilo *REST* identificados en cada una. Esto dificultaría una correcta comprensión y comunicación entre los analistas e implementadores, obstaculizando el desarrollo de aplicaciones *‹‹RESTful››.*

1.6. Conclusiones parciales

En este capítulo se precisa la investigación necesaria para comprender lo que se conoce dentro de la Ingeniería de Software como modelado del *‹‹software››*, detallando y describiendo las actividades, tareas, así como los procedimientos y procesos de vital importancia que se realizan dentro de esta disciplina. Se indagó acerca de las herramientas, lenguajes, el modelado en *RUP* y en procedimientos para modelar Arquitectura Orientada a Recurso. Se realizó un estudio sobre los estándares de modelado para favorecer la eficiencia y calidad del proceso, proveyendo así una base para el

conocimiento más detallado del modelado de *‹‹software››* orientado a recurso. A raíz del estudio se conoce que:

- la investigación realizada arroja que existe poco conocimiento referente al modelado de sistemas *‹‹RESTful››,*
- no se cuenta con procedimientos, guías o notaciones estándar para desarrollar esta disciplina, dando como resultado que los productos que se desarrollan tengan dicha deficiencias y el trabajo resulte más engorroso.

En el presente capítulo se expondrá la propuesta solución de la investigación, teniendo en cuenta el objeto de estudio trazado, así como su campo de acción. Se definen aspectos necesarios para el modelado de aplicaciones *‹‹RESTful››,* así como las premisas y alcance de la propuesta. Se aborda sobre el *‹‹metamodelo››* expuesto por *Silvia Schreier* como punto de partida para la solución, finalizando con el desarrollo de una guía para el modelado de aplicaciones de este perfil, poniéndose en práctica en un caso de estudio.

2.1 Situación actual de la problemática analizada

El inminente auge que experimenta la actualidad en cuanto a la gestión de información de cualquier naturaleza y en todo tipo de soportes, esencialmente digitales, propicia el apoyo de herramientas y tecnologías que faciliten el desarrollo de sistemas informáticos para el adecuado almacenamiento, manejo y control de la información, consolidado, ordenado y supervisado cada paso, y cada avance del resultado por un robusto estilo arquitectónico.

El sistema informático de GDA eXcriba perteneciente a la UCI confía su nueva versión en el estilo arquitectónico *REST.* Lo expresado anteriormente desata a su paso un gran problema, ya que no existe un patrón estándar que permita representar en *UML* los servicios *‹‹RESTful››* durante todo el análisis y diseño del desarrollo de la aplicación. Esto trae consigo que no se pueda contar con una notación estándar para los modelos y diagramas, los cuales son concebidos en la documentación durante el proceso de desarrollo de la aplicación, volviéndose incompleto e insatisfactorio dicho proceso.

Modelar recursos en las aplicaciones *‹‹RESTful››,* sus enlaces y todo el contenido dinámico de su gestión en cada capa de la aplicación es muy importante. Según *RUP*, las aplicaciones web pueden representarse por un conjunto de modelos: de casos de usos, de implementación, de despliegue, de seguridad, entre otros (1). Por ejemplo: en una aplicación de este tipo, un diagrama de componentes puede verse como un mapa del sitio o de navegación. Sin embargo, con dicho diagrama no se puede modelar las colaboraciones que ocurren en cada capa de la aplicación. Este nivel de abstracción, extremadamente importante para todo el equipo de desarrollo, necesita ser parte del modelo, y para ello *UML* expone diagramas de clases para representar estas colaboraciones.

De acuerdo con lo antes planteado una posible solución al problema científico de la investigación

consistiría en modelar cada recurso lógico como una clase. ¿Pero qué problemas descartarían el tratar de utilizar el diagrama de clases tradicional para modelar recursos? Si un recurso es modelado como una clase, entonces:

- los métodos que define la interfaz uniforme para la gestión de los mismos como: GET, POST, PUT, DELETE, podrían interpretarse como operaciones de dicha clase,
- los atributos de esta clase pueden ser aquellas variables cuyo ámbito sea el recurso,
- y sea un enlace, el vínculo que existe entre dos recursos X,Y tal que, parte de la representación de X es dependiente del recurso Y obtenido mediante el consumo de un servicio; entonces se puede interpretar dicho enlace como una relación entre las clases que representan a X y Y respectivamente.

Al aplicar el procedimiento descrito anteriormente para modelar los recursos presentes en la siguiente situación: *en la Universidad de las Ciencias Informáticas se desea realizar una encuesta a los estudiantes con el objetivo de conocer el grado de satisfacción y compromiso de los mismos para con su estructura productiva. Dicha encuesta será aplicada a los estudiantes una sola vez –siempre que sea válida– y de forma personal*. Se obtiene el siguiente diagrama de clases.

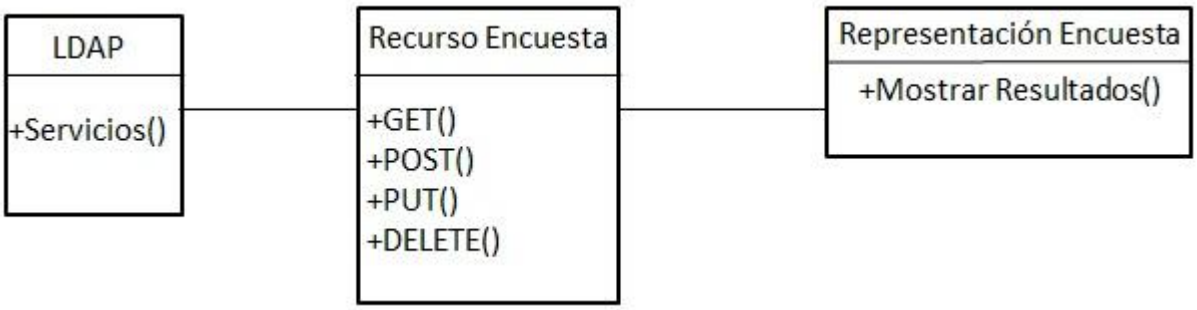


Figura 12: Diagrama de clases del ejemplo encuesta.

La figura representa el recurso **encuesta** realizada a los estudiantes. La misma apoyada en un servicio brindado por LDAP para cerciorarse que los encuestados sean solo estudiantes, tendría una representación asociada con los resultados de la misma. Al parecer todo ha sido muy sencillo. Sin embargo, existe un problema con este modelo**:** Aunque *REST* define una interfaz uniforme para la gestión de los recursos[5], no es necesario implementar todos los métodos de la interfaz. Al apoyarse en una clase de *UML* para modelarlo, no se sabe distinguir:

[5] Ver epígrafe 1.4.2 Restricciones de REST del capítulo 1

- las operaciones,
- los formatos de las representaciones,
- qué relaciones o enlaces están activos o son requeridos
- y qué relaciones forman *enlaces*, como el representado en la clase LDAP.

Por este motivo no es conveniente asemejar un recurso como una clase *UML*. Hay que tener en cuenta también que dicho lenguaje está creado para el modelado de aplicaciones orientadas a objeto, por tal motivo, los componentes que lo integran están pensados para modelar ese tipo de semántica, por lo que una clase de *UML* no corresponde semánticamente a un recurso del estilo *REST*. El llegar a representar tal abstracción por parte de los diseñadores del sistema acarrea un error conceptual. Esto provoca que la interpretación del modelo en los siguientes flujos y etapas, dependerá de aspectos como: experiencia, grado de conocimiento y sobre todo, del nivel de abstracción que puedan emplear los implementadores del equipo. Todo esto incita a errores durante el desarrollo que afecten la calidad de todo el proceso.

Para resolver el problema, se decide modelar los aspectos necesarios para la gestión de los recursos a través de *UML.* La investigación se apoya en las características que hacen el lenguaje ser una especificación que puede cubrir aspectos de modelado no especificados en su documento rector, permitiéndole extender su semántica en aquellas situaciones en las que no es posible capturar todas las características de una determinada arquitectura o dominio en particular. Teniendo en cuenta lo antes expuesto, se plantea la creación de una extensión a dicho lenguaje para el diagrama de clases en el flujo de diseño. La misma debe modelar los componentes que se ejecutan en la aplicación así como sus relaciones con los restantes artefactos de *UML.* La propuesta requiere de un nivel de abstracción adecuado para los diseñadores que les facilite la comprensión de los elementos a modelar, definiendo así una correcta semántica y facilitando el desarrollo en las siguientes etapas, sin requerir una compleja comprensión o abstracción por parte de los programadores del equipo.

La propuesta tiene como objetivo aportar una guía que cubra la necesidad de modelar servicios en el desarrollo de aplicaciones *‹‹RESTful››,* permitiendo una documentación legible del comportamiento del sistema en cada capa, etapa y flujo de trabajo durante su desarrollo. La misma se basa en los estudios realizados en el capítulo uno, además, de las bibliografías consultadas, así como el conocimiento y la experiencia de algunos conocedores del tema.

Para lograr los objetivos del trabajo, primeramente se presentarán algunos aspectos generales de la solución como el alcance y premisas para su uso. Seguidamente para una mejor comprensión se analizará y describirá el ‹‹*metamodelo*›› de *REST* expuesto por *Silvia Schreier* y para finalizar se

describe la extensión de *UML* para el modelado de aplicaciones *‹‹RESTful››*. Se explican los criterios utilizados en la definición de la extensión y se crea una guía para los usuarios que deseen enmarcarse en el desarrollo de aplicaciones de este tipo, dejando plasmado cada paso y aspecto a tener en cuenta para el modelado de estas aplicaciones. Se pone en práctica la solución arrojada en un caso de estudio en el que se usa la notación gráfica propuesta.

2.2 Alcance de la solución

La propuesta es aplicable a los proyectos de *‹‹software››* que deseen desarrollar una iniciativa *‹‹RESTful››* guiados por el Proceso Unificado de Desarrollo de Software. Describe de manera general los aspectos a tener en cuenta para el modelado de proyectos de este perfil, dejando una descripción clara de las entregas necesarias así como una descripción de su utilidad para el equipo de desarrollo que las necesite y todos los involucrados en general.

2.3 Premisas para el uso de la solución

Metodología de desarrollo RUP: La propuesta necesita partir del Proceso Unificado de Desarrollo de Software como base metodológica para el mismo.
Estilo arquitectónico REST: La propuesta necesita partir de una aplicación orientada a servicios, basada en estilo arquitectónico *REST.*
Personal de la organización calificado: La propuesta necesita de personal con conocimiento de técnicas, estándares y herramientas necesarios para el proceso de modelado o al menos que se encuentren abiertos al aprendizaje del mismo.

2.4 Metamodelo REST

El ‹‹*metamodelo*›› *REST* se divide en el modelo estructural y el modelo de comportamiento. El primero para describir los tipos de recursos, sus atributos y relaciones. El segundo para describir el comportamiento y el estado de los recursos.

Elementos necesarios para comprender el metamodelo REST

Para comprender el ‹‹*metamodelo*›› de REST se debe tener en cuenta que (17):
EClass: representa un concepto y se identifica por un nombre. Se puede definir; un conjunto de *EAttributes* que describen sus propiedades y un conjunto de *EReferences* que describen las relaciones con otras *EClass*: un conjunto de superclases, donde un *EClass* hereda todos los atributos y referencias de su superclase: un *EClass* puede ser abstracta lo que impediría crear instancias de ella.

EDataType: representa los tipos de datos primitivos como enteros y cadenas, se identifica por un nombre.

EAttribute: se identifica por un nombre y el tipo lo define un *EDataType*, posee propiedades para definir el límite inferior y superior de la cantidad de valores que están permitidos para el *EAttribute* y si los valores son ordenados.

EReference: dirige las relaciones entre las instancias de dos *EClass* definiendo el tipo de relación e identificándola con un nombre.

2.4.1 Modelo Estructural

Como primer paso, se identifican los elementos estructurales del ‹‹*metamodelo*››, usando el término recurso cuando se habla de un elemento concreto, *TipoRecurso* para describir los aspectos comunes de múltiples recursos. Los elementos identificados se modelan como *EClasses*, sus propiedades como *EAttributes* y sus relaciones como *EReferences*, el *EDataType* de *EAttributes* se omite debido a que generalmente se obvia. Debido a la interfaz uniforme, un *TipoMetodo* identificado por su nombre, tiene que ser definido para todos los métodos existentes. Un *TipoRecurso* está asociado con un conjunto de métodos compatibles que son responsables del comportamiento y determina el conjunto de *TipoMedio* producidos y consumidos. Adicionalmente cada método puede definir parámetros que pueden contenerse en un *TipoMedio* consumido o en el identificador del recurso.

Teniendo en cuenta lo antes planteado, se han identificado los elementos siguientes, incluyendo atributos y relaciones entre ellos:

TIPO DE RECURSOS	DESCRIPCIÓN
Recurso primario	Conceptos básicos del dominio de modelado, ej. fotos y álbum
Subrecurso	Parte de otro recurso que también deben ser direccionado
Lista de recurso	Lista de todos los recursos concretos de los recursos primarios
Filtro de recurso	Lista de recursos concretos con las propiedades deseadas
Proyección de recurso	Contiene sólo un subconjunto de atributos de otro recurso
Agregación de recurso	Atributos agregados de diferentes recursos para reducir la interacción
Recurso paginado	Divide los recursos en páginas
Actividad de recurso	Representa un sólo paso de un flujo de trabajo

***Tabla 1: Diferentes tipos de recursos* (17).**

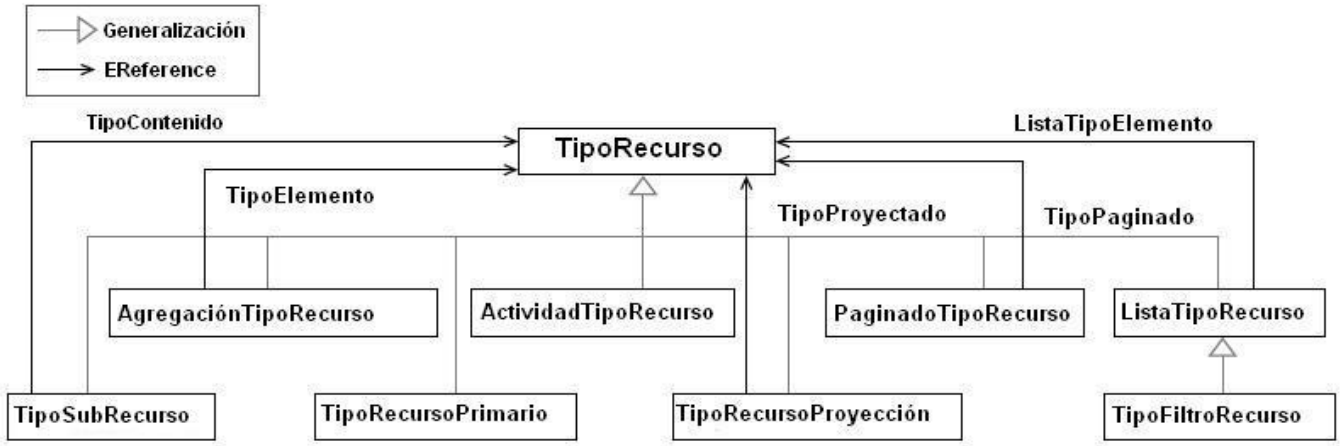


***Figura 13: Jerarquía y relaciones de los tipos de recurso (Se omiten detalles como los atributos y multiplicidades para mayor claridad)* (17).**

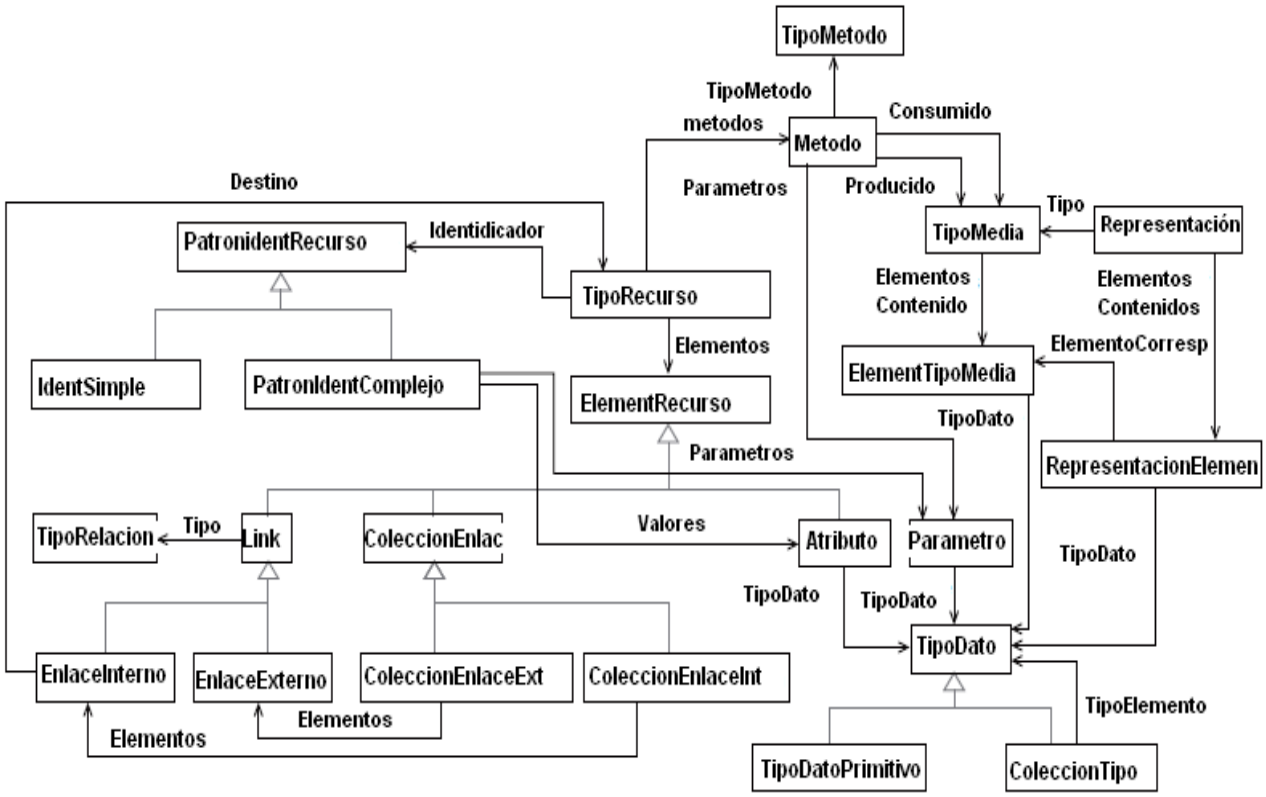


***Figura 14: Modelo estructural (Se omiten detalles como los atributos y multiplicidades para más claridad)* (17).**

2.4.2 Modelo de comportamiento

De acuerdo con *Fielding*, las representaciones se utilizan "para capturar el estado actual o previsto de ese recurso" (17).

Para describir el comportamiento de los *TipoRecurso*, se eligieron máquinas deterministas de estado finito, ofreciendo la posibilidad de representar el estado actual de los recursos y definir cómo reacciona un recurso a una solicitud determinada.

EspecifComportamiento

Cada método tiene que definir su comportamiento, esta especificación se compone de una acción, el termino *SecuenciaAccion* es usado para representar una secuencia de acciones tales como: *AccionCrear* para crear nuevos recursos; *AccionMensage* para mensajes internos o el envío de peticiones a otros recursos; *AccionModificar* ofrece la posibilidad de cambiar el valor de los atributos y enlaces; *AccionRetornar* permiten definir la respuesta, incluyendo la representación y todos los ‹‹*metamodelo*››. La definición *AccionCondicional* puede ser utilizada para definir el comportamiento deseado, la *AccionadicionarLista* y *AccioneliminarLista* permiten agregar y quitar elementos de las colecciones. *EspecifComportamiento* es una posibilidad para definir las restricciones semánticas de *TipoMetodo* al limitar las acciones permitidas, por ejemplo, ninguna *AccionModificar* se permitiría en una especificación de *GET* (17).

Creador

Para definir la creación de recursos de todos los *TipoRecurso* se puede definir un creador, cuyos métodos definen parámetros, *TipoMedio* consumidos y una *EspecifComportamiento* (17).

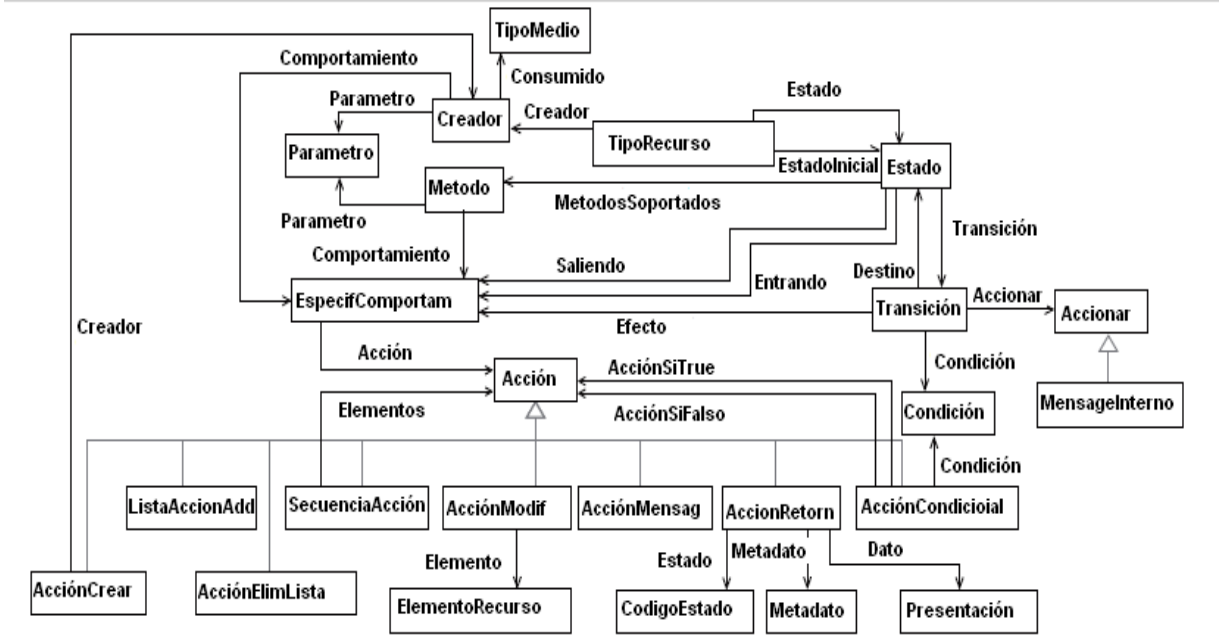


***Figura 15: Modelo de comportamiento (Se omiten detalles como los atributos y multiplicidades para más claridad)* (17).**

2.5 Criterios para la extensión

Para la elección de todos los estereotipos necesarios para representar en *UML* todos los componentes *‹‹RESTful››* y sus relaciones, se ha seguido el criterio siguiente:

- todos los recursos serán controlados y gestionados únicamente a través de la clase estereotipada como <<Interfaz uniforme>> según lo define una de las restricciones de *REST,* dicha clase es única en el modelo*,*
- los recursos serán considerados clases estereotipadas como <<Recurso>> ya que se encuentran de manera explícita definidos en *REST* y constituyen el elemento principal en aplicaciones de este perfil,
- la representación de cada servicio será mediante una clase estereotipada como <<Representación>> que será utilizada mediante una composición a la Interfaz uniforme,
- los enlaces a recursos externos a la aplicación serán representados mediante una asociación estereotipada como <<Enlace externo>>,
- las relaciones de uso entre los recursos serán representadas mediante las relaciones de asociación que propone *UML.*

2.6 Aspectos a tener en cuenta para la descripción de la extensión

Para la descripción de la extensión, la investigación se apoyara en aspectos que definan el estereotipo extendido, tales como:

- **Tipo:** define el tipo de categoría del elemento representado,
- **Descripción:** describe los aspectos más importantes de la categoría que representa,
- **Atributos:** define los atributos de la categoría representada, los mismos pueden ser o no opcional,
- **Restricciones:** define las restricciones impuestas para la lógica representación y semántica durante el modelado de la categoría representada,
- **Valores etiquetados:** define otros aspectos importantes de las propiedades o atributos de la categoría representada,
- **Relaciones:** define las relaciones de la categoría representada.

2.7 Descripción de la extensión

La extensión de *UML* define un conjunto de estereotipos, valores etiquetados y restricciones que permite representar en notación gráfica de *UML*, cada uno de los componentes *‹‹RESTful››.*

Clase Interfaz uniforme

Tipo: Clase

Descripción: Una clase <<Interfaz uniforme>> representa la clase a través la cual se gestionarán los servicios correspondientes a cada recurso.

Atributo: Opcional.

Restricciones: No admite métodos que vallan más allá de crear, leer, actualizar y eliminar. Debe tener asociada una representación definiendo el tipo ‹‹*MIME*›› como lo expresa dicha restricción.

Valores etiquetados: Definir las ‹‹*URI*›› amigables para cada recurso y describir que significa hacer *GET*, *POST, PUT, DELETE* a cada recurso y si se permite o no.

Relaciones: Se relacionará con los recursos garantizando su gestión, además, mediante una composición se relacionará con la case estereotipada como <<Representación>> para representar la gestión de los mismos, así como con los demás artefactos de *UML* que intervengan directamente con los recursos gestionados.

Clase Recurso

Tipo: Clase

Descripción: Una clase <<Recurso>> representará un recurso de la aplicación a modelar.

Atributo: Opcional

Restricciones: No admite métodos que vallan más allá de crear, leer, actualizar y eliminar, todos los recursos deben estar gestionados a través de la clase <<Interfaz uniforme>>.

Valores etiquetados:--

Relaciones: Se relacionarán entre los recursos para representar el uso de los mismos, así como la dependencia entre ellos.

Clase Representación

Tipo: Clase

Descripción: Una clase <<Representación>> representa el contenido gestionado del recurso en la aplicación.

Atributo: Una clara descripción del contenido representado que incluya: nombre, descripción, URL, formato, autenticación.

Restricciones: Cada servicio consumido debe especificar los argumentos requeridos por esta clase.

Valores etiquetados: Otros nombres que serán utilizados en la descripción del servicio.

Relaciones: Se relacionará mediante una composición a la clase estereotipada como <<Interfaz uniforme>>, y con las clases que intervengan en la gestión de los recursos.

Asociación Enlace externo

Tipo: Asociación

Descripción: Una asociación <<Enlace externo>> representa una relación con un recurso externo a la aplicación.

Atributo: Opcional

Restricciones: La asociación <<Enlace externo>> sólo se usara para relacionar a algún recurso externo a la aplicación.

Valores etiquetados:--

Relaciones:--

2.8 Pasos a seguir para el modelado

Para lograr un correcto modelado, la solución expone cuatros pasos necesarios a tener en cuenta:

- identificar los recursos que serán gestionados en la aplicación,
- modelar las relaciones entre los recursos de la aplicación,
- definir ‹‹*URL*›› amigables para cada uno de los recursos consumidos o producidos en la aplicación,
- comprender los métodos que soportan cada uno de los recursos de la aplicación.

REST lleva a considerar los servicios web como recursos, de este modo los servicios brindados por aplicaciones externas serán considerados como tal.

2.9 Caso de estudio

En esta sección se muestra un caso de estudio que usa la extensión de *UML* propuesta. Para ello en primer lugar, se explica detalladamente el caso y luego se muestra el uso de cada uno de los estereotipos propuestos.

Tal como se ha expuesto, se propone modelar aplicaciones *‹‹RESTful››* desarrolladas bajo el Proceso Unificado de Desarrollo de Software, utilizando para ello una extensión de *UML*. Partiendo del ‹‹*metamodelo*›› de *REST* y enfocados en la metodología *RUP*, se asume la premisa anteriormente expuesta: " *Personal de la organización calificado* " por lo que pasos como el estudio y análisis tales como alcance, visión, requerimientos y la selección de una arquitectura conveniente así como los artefactos arrojados perteneciente a cada flujo de trabajo en cada una de las etapas se asumirán como cumplidos para el caso de estudio, para un mejor enfoque en el proceso de modelado en dicha metodología.

Como caso de estudio se presenta un caso de uso del módulo Gestión de Metadatos Asociados a las Tipologías Documentales de eXcriba. Dicho caso de estudio solo se centrará en el caso de uso gestionar aspectos de un documento o carpeta, específicamente en el flujo de eventos eliminar aspectos de un documento o carpeta.

Partiendo de lo antes planteado el caso de uso quedaría representado en un diagrama de casos de usos del sistema como sigue:

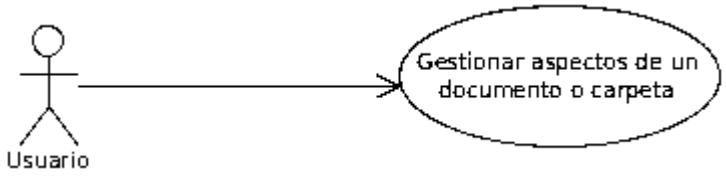


Figura 16 : Diagrama de casos de uso del sistema.

2.9.1 Definición del caso de uso

CU-2	Gestionar aspecto de un documento o carpeta
Actor	Usuario
Descripción	Esta funcionalidad permitirá que el actor adicione o elimine en función de sus necesidades determinados conjuntos de *‹‹metadatos››* en tiempo de ejecución a un documento o carpeta específico del sistema.
Referencias	R3 y R4

***Tabla 2: Definición de caso de uso: Gestionar aspectos de un documento o carpeta* (19).**

2.9.2 Descripción textual del caso de uso del sistema

Caso de Uso	**Gestionar aspectos de un documento o carpeta**
Actor	Usuario
Resumen	El caso de uso inicia cuando el usuario desee añadir o eliminar un conjunto de nuevos *‹‹metadatos››* a un documento o carpeta seleccionado en el sistema y finaliza cuando se añaden o se sustraen respectivamente estas propiedades del elemento especificado
Referencias	R3 y R4
Recomendaciones	El usuario ha sido autenticado en el sistema.
	El usuario tiene los permisos necesarios para poder realizar esta acción.
Poscondiciones	
Flujo de eventos	
Sección: "Adicionar aspecto a un documento o carpeta"	
Acción del Actor	**Respuesta del Sistema**
1. Selecciona el documento o carpeta al cual	2. Muestra una lista de acciones básicas.

le añadirá nuevos *‹‹metadatos››* y selecciona la opción "Acciones" en la barra lateral derecha.	
3. Selecciona la acción "Adicionar aspecto" del listado desplegado.	4. Despliega una lista con los aspectos definidos en el repositorio.
5. Procede a seleccionar el aspecto a adicionar.	6. Muestra las propiedades contenidas dentro del aspecto seleccionado.
7. Introduce los valores de las propiedades mostradas y presiona el botón Aceptar.	8. Actualiza y añade las nuevas propiedades contenidas dentro del aspecto seleccionados al documento o carpeta especificada.
Sección: "Eliminar aspectos de un documento o carpeta"	
1. Escoge el documento o carpeta al cual le eliminará uno o varios aspectos y selecciona la opción "Acciones" en la barra lateral derecha.	2. Muestra una lista de acciones básicas.
3. Selecciona la acción "Eliminar aspecto" del listado desplegado.	4. Muestra una lista con los aspectos asociados al elemento seleccionado.
5. Procede a seleccionar el o los aspectos a sustraer del documento o carpeta previamente escogido y presiona el botón Aceptar.	6. Actualiza y sustrae las propiedades contenidas dentro del o los aspectos seleccionados del documento o carpeta especificada.
	7. Termina el caso de uso.

***Tabla 3: Descripción textual del CU: Gestionar aspectos de un documento o carpeta* (19).**

2.9.3 Modelado del caso de uso según la solución arrojada por la investigación

Dando complimiento con el primer paso de la solución se identifican los recursos siguientes, representados con el estereotipo <<Recurso>> expuesto en la extensión:

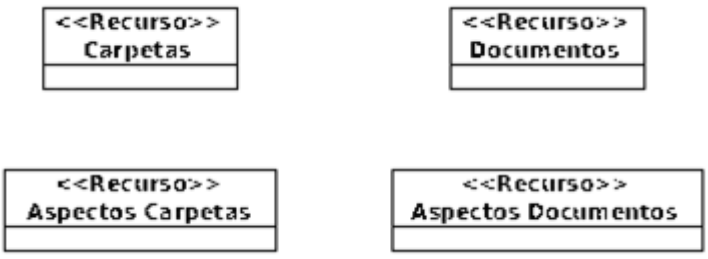


Figura 17: Recursos identificados.

Seguidamente se modelan las relaciones entre los recursos identificados:

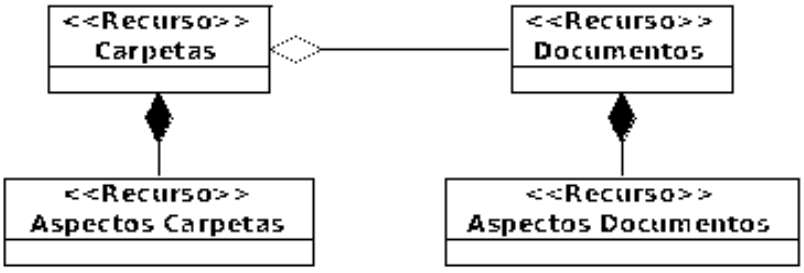


Figura 18: Relaciones entre los recursos identificados.

Los recursos serán gestionados a través de una interfaz uniforme como lo expone la restricción del estilo, la misma estereotipada como <<*Interfaz uniforme*>> propuesta en la extensión de *UML*. En dicha interfaz se definirán las *URL* amigables de acceso a cada recurso de la aplicación como valores etiquetados, mostrándose como una nota en el modelo, como lo expresa el tercer de los pasos a tener en cuenta para el modelado:

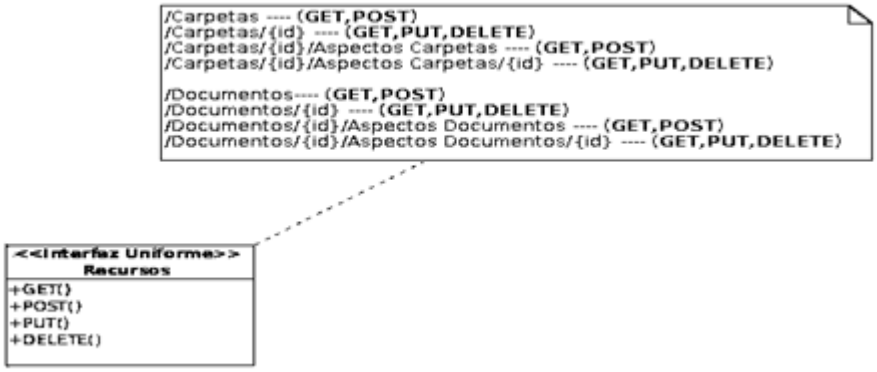


Figura 19: Interfaz uniforme para la gestión de los recursos identificados.

Para la gestión de los recursos se hace a través de una clase controladora guiada por la Interfaz uniforme, hará referencia mediante una asociación a los recursos gestionados y referenciará los recursos consumidos externos a la aplicación con la asociación estereotipada como <<Enlace externo>>:

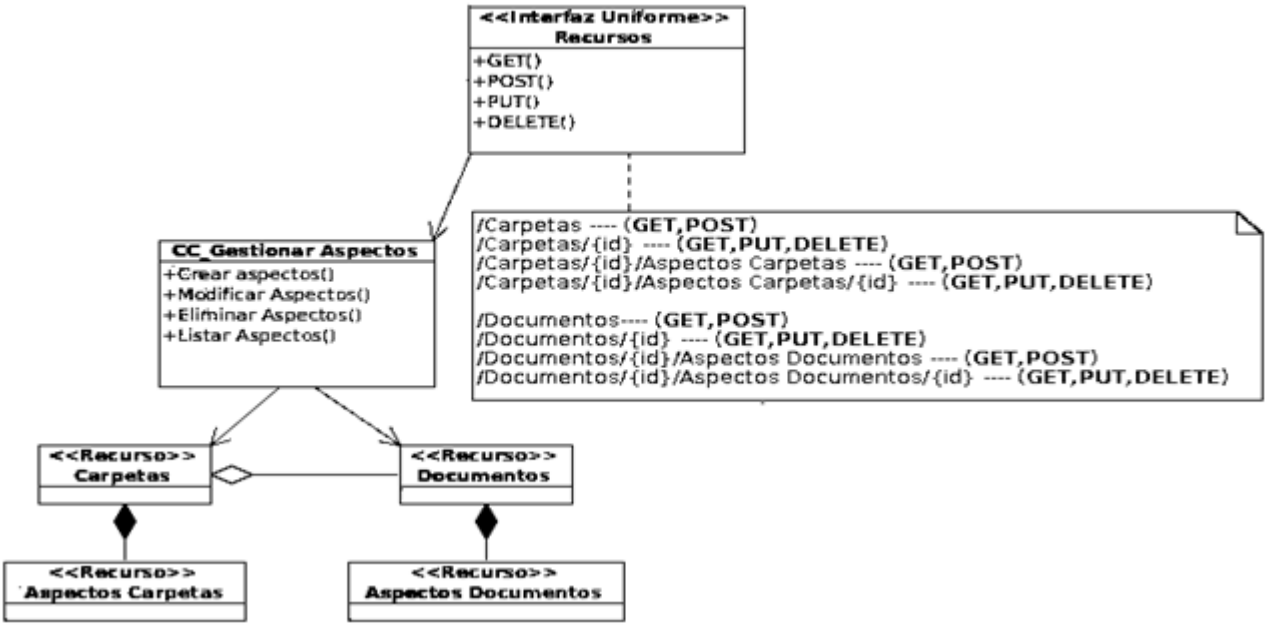


Figura 20: Gestión de recursos.

Cada contenido del recurso consumido o producido será representado mediante una clase estereotipada como <<Representación>>, la misma expondrá parámetros necesarios para su correcta representación, la cual hará referencia a la Interfaz uniforme a través de una composición como se muestra en la Figura 21.

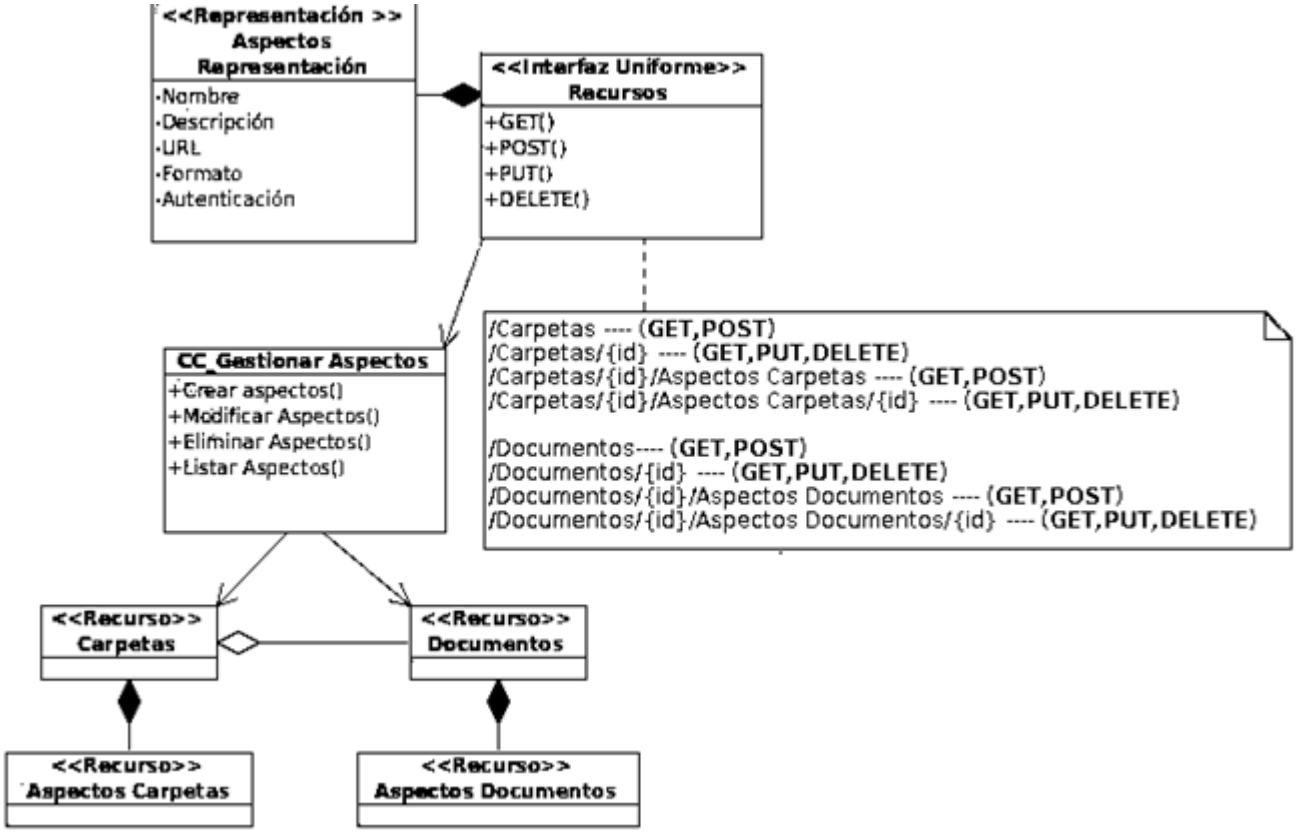


Figura 21: Representación de los recursos.

La gestión de los recursos según la propuesta de la investigación quedaría representada como lo muestra la Figura 21. En dicha solución, se añaden los estereotipos necesarios para modelar todo el proceso de una forma lógica y semánticamente correcta, guiando el mismo de manera natural y estándar, facilitando así la comprensión por parte del equipo de desarrollo.

Anteriormente no se contaba con una notación estándar que cubriera el proceso de modelado de las aplicaciones *‹‹RESTful››* durante todo el análisis y diseño de la aplicación, por lo que diagramas de clases durante el desarrollo de aplicaciones de este perfil como el del eXcriba, quedaban inconclusos y abstractos como lo muestra la Figura 22, dificultando su total comprensión para la implementación.

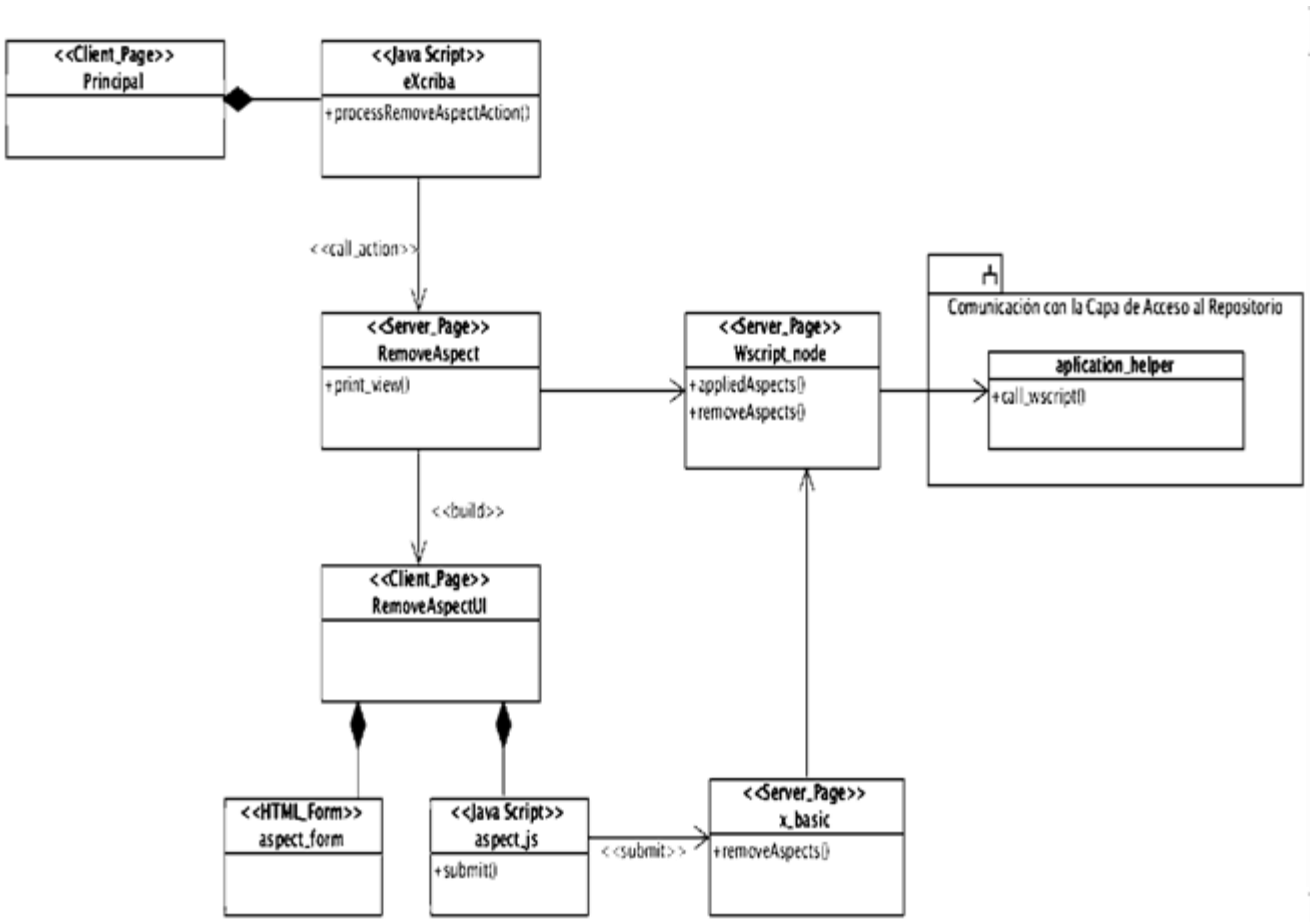


***Figura 22: Diagrama de clases del diseño del CU Gestionar Aspectos. Sección: Eliminar* (19).**

El desarrollo de la investigación arroja una nueva técnica a tener en cuenta para el modelado de aplicaciones *‹‹RESTful››,* por lo que ya sistemas como el eXcriba, podrán contar con una notación estándar que represente y guíe el consumo así como la gestión de los recursos durante todo el análisis y diseño, como se evidencia en la Figura 23, posibilitando una clara comprensión de la aplicación por parte de todo el equipo de desarrollo y por el personal ajeno también, facilitando así el trabajo durante el flujo de implementación.

La vista proporciona al usuario controles gráficos para realizar las operaciones necesarias en la gestión de los recursos, una vista general del caso de uso perteneciente al sistema eXcriba, quedaría reflejada contando con la solución de la investigación de la siguiente forma, como se expone en el diagrama de clases de la Figura 23.

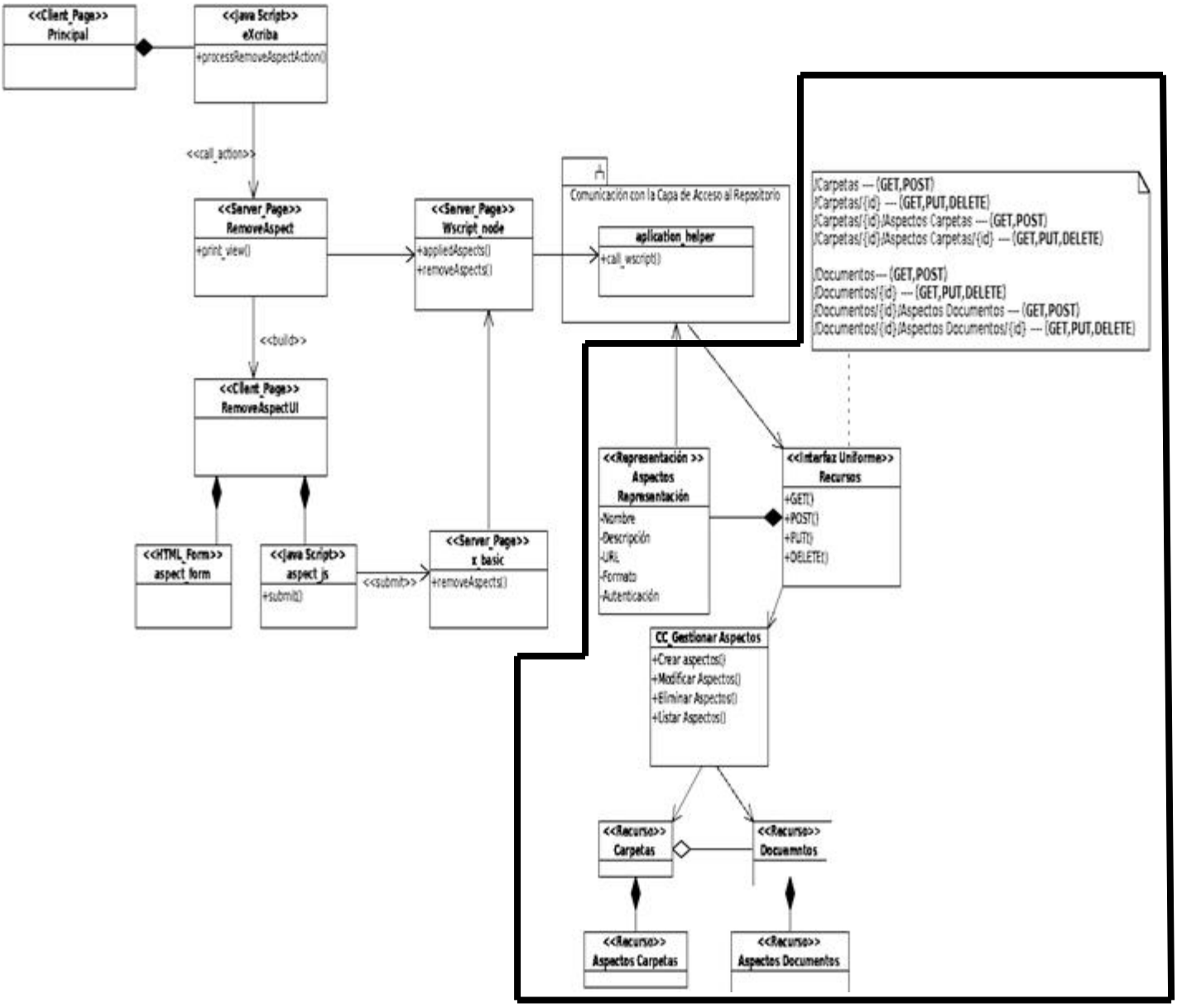


Figura 23: Diagrama de clases del caso de estudio.

2.10 Conclusiones parciales

Durante el capítulo se definieron las características fundamentales de la propuesta para el modelado de aplicaciones *‹‹RESTful››,* como son su alcance y premisas para su uso.

- la investigación apoyándose en los mecanismos que define *UML* para ampliar su semántica, arroja una extensión ha dicho lenguaje para el modelado de aplicaciones *‹‹RESTful.*
- se presenta un caso de estudio referente a un caso de uso del eXcriba, en el cual se ha empleado la notación gráfica propuesta en la investigación.

Para validar el modelo propuesto, es necesario contar con el criterio de personas expertas en el tema y que poseen el conocimiento necesario para saber si lo investigado, está realmente cercano a alcanzar la calidad que se espera obtener en los resultados. Para la validación y aceptación del modelo propuesto para modelar sistemas *‹‹RESTful››* se empleará una variante del método *Delphi*, donde se trabaja con un grupo de expertos que se mantienen aislados, los mismos pueden ser empleados de la organización o especialistas externos y además, se analizan los cambios o resultados esperados estimándose el tiempo en que puede ocurrir (20).

3.1 Descripción del método

Delphi es un método experto que se basa en la consulta a personas que tienen grandes conocimientos sobre el entorno en el que la organización desarrolla su labor, estas personas exponen sus ideas y finalmente se redacta un informe en el que se indican cuáles son, en su opinión, las posibles alternativas o sucesos que se obtendrán en el futuro (21).

La calidad de los resultados obtenidos por este método experto dependen de:

- la elaboración de los cuestionarios,
- las predicciones de los expertos consultados.

El *Delphi* es pronosticado como uno de los métodos más fiables porque se basa en la interrogación a expertos con la ayuda de cuestionarios sucesivos, con el fin de encontrar convergencias en las opiniones y así deducir eventuales consensos. Se basa en la consulta de un grupo de expertos de forma individual por medio de un conjunto de preguntas bien conformadas, que apoyadas por los resultados promedio de la ronda anterior, genera coincidencia de opiniones (22).

Algunas de las ventajas que ofrece el método *Delphi* son:

- permite la formación de un criterio con mayor grado de objetividad y el consenso logrado sobre la base de los criterios es muy confiable,
- la tarea de decisiones sobre la base de los criterios de expertos, obtenido por este tiene altas probabilidades de ser eficiente,
- permite valorar alternativas de decisión.
- un requisito imprescindible para garantizar el éxito del método, evitar conflictos entre expertos y crear un clima favorable a la creatividad, es ser anónimo,

- el experto se siente involucrado plenamente en la solución del problema y facilita su implantación,
- de ello es importante el principio de voluntariedad del experto en participar en la investigación y la confidencialidad de su opinión.

El método presenta cuatro características principales:

- **Anonimato:** ningún experto conoce la identidad de los otros que componen el grupo de debate.
- **Iteración y retroalimentación controlada:** la iteración se consigue al presentar varias veces el mismo cuestionario. Como se van presentando los resultados obtenidos de los cuestionarios anteriores, se consigue que los expertos vayan conociendo los distintos puntos de vista y puedan ir modificando su opinión si los argumentos presentados les parecen más apropiados que los suyos.
- **Respuesta del grupo en forma estadística:** la información que se presenta a los expertos no es sólo el punto de vista de la mayoría, sino que se presentan todas las opiniones indicando el grado de acuerdo que se ha obtenido.
- **Heterogeneidad:** pueden participar expertos de determinadas ramas sobre las mismas bases.

El criterio de expertos puede ser tomado mediante encuestas o entrevistas y empleado en cualquier momento de la investigación, cuya experiencia y opiniones pueden ser de una valiosa contribución, resultando fundamental durante el estudio exploratorio.

Para la aplicación del método se siguieron tres etapas fundamentales:

- elección de expertos,
- elaboración del cuestionario, para la validación de la propuesta,
- desarrollo práctico y explotación de resultados.

3.2 Resumen de fases

3.2.1 Elección de los Expertos

La selección de un grupo de expertos a encuestar deben ser personas experimentadas, independientes, con reconocida competencia, creativas e interesadas en participar y con conocimiento en el tema que garantice la confiabilidad de los resultados. Se especifican una serie de criterios de selección:

- graduado del Nivel Superior,
- conocimientos sobre Arquitectura Orientada a Servicios (*SOA*).
- conocimientos sobre Arquitectura Orientada a Recursos (*ROA*),
- experiencia laboral,

- capacidad de análisis y pensamiento lógico,
- disposición para participar en la validación.

La selección de expertos atendiendo a estos criterios, proporciona la obtención de resultados con calidad, junto a otras cualidades propias de ellos, que pueden ser: la honestidad, la sinceridad y responsabilidad; haciendo que las opiniones brindadas sean confiables y válidas para el objetivo propuesto.

Primeramente para la selección de los expertos finales se hace necesario conocer el grado de conocimiento del experto en cuestión, la misma se realiza con la ayuda del Coeficiente de Competencia. Este coeficiente se determina mediante la fórmula:

K= ½ (Kc + Ka), donde:

Kc: es el Coeficiente de Conocimientos del experto sobre el tema.

Ka: es el Coeficiente de Argumentación del experto sobre el tema.

Kc se obtiene de la siguiente tabla que recoge una autoevaluación del posible experto.

1	2	3	4	5	6	7	8	9	10
								X	

Tabla 4: Autovaloración del Coeficiente de Conocimientos (Kc).

El presunto experto marcará en la casilla enumerada, según su criterio acerca de la capacidad que él tiene sobre el tema que se la ha sometido a su consideración, en una escala del 1 al 10 y que después para ajustarla a la teoría de las probabilidades se multiplicará por 0,1; de esta forma, si selecciona el 9 en la Tabla 4, al multiplicarlo por 0.1 se considerada en la tabla, Kc = 0.9. La evaluación "1" indica que el experto no tiene absolutamente ningún conocimiento de la problemática correspondiente, mientras que la evaluación "10" significa que el experto tiene pleno conocimiento de la problemática tratada.

Para calcular el coeficiente de argumentación se procede de la siguiente forma. En la siguiente tabla el experto debe marcar, según su criterio, su grado de competencia sobre los aspectos sometidos a consideración. Las marcas de los expertos se traducen a puntos, según la siguiente escala:

Fuentes de Argumentación	**Nivel de Influencia de cada cuente**		
	(A) Alto	**(M) Medio**	**(B) Bajo**
Análisis realizado por usted.	**0.3**	**0.2**	**0.1**
Experiencia en el tema de investigación.	**0.5**	**0.4**	**0.2**
Trabajos de autores nacionales.	**0.05**	**0.05**	**0.05**
Trabajos de autores extranjeros.	**0.05**	**0.05**	**0.05**
Su propio conocimiento del tema.	**0.05**	**0.05**	**0.05**
Su intuición	**0.05**	**0.05**	**0.05**
Totales.	**1.0**	**0.8**	**0.5**

Tabla 5: Autovaloración del Coeficiente de Argumentación (Ka).

Con estos elementos es suficiente para obtener el Coeficiente de Competencia (K). Por ejemplo, si las selecciones del experto en la tabla son las siguientes:

Fuentes de Argumentación	Nivel de Influencia de cada cuente		
	(A) Alto	(M) Medio	(B) Bajo
Análisis realizado por usted.	X		
Experiencia en el tema de investigación.		X	
Trabajos de autores nacionales.			X
Trabajos de autores extranjeros.		X	
Su propio conocimiento del tema.		X	
Su intuición		X	

Tabla 6: Autovaloración del Coeficiente de Argumentación. Ejemplo.

Se Busca en la Tabla 5 el valor que coincide con el de la Tabla 6 y se realiza el cálculo:

K = (Kc + Ka) / 2.

Se establece el código para la interpretación de tales Coeficientes de Competencia (K) quedando de la siguiente manera:

- Si 0.8 ≤ K < 1.0, el Coeficiente de Competencia es alto y confiable.
- Si 0.5 ≤ K < 0.8, el Coeficiente de Competencia es medio.
- Si K < 0.5 el Coeficiente de Competencia es bajo.

Observación: Como a la categoría de "bajo" se le otorgaron puntos, siempre el Coeficiente de Competencia (K= ½ (Kc + Ka) quedará comprendido entre:

$$(0+0.5)/2 \leq K \leq (1+1)/2 <==> 0.25 \leq K \leq 1$$

No existe una norma generalizada para determinar el número óptimo de expertos. Para su selección es necesario determinar el número de expertos que debe tener el grupo, hasta 7 expertos el error disminuye exponencialmente, después de 30, aunque el error disminuye, lo hace de manera poco significativa y no compensa el incremento de costos y esfuerzo, por lo que se sugiere utilizar un número de expertos en el intervalo de 7 a 30.

Para obtener la autovaloración de los expertos se utilizó la siguiente encuesta de autovaloración para obtener los coeficientes de competencia de los expertos.

ENCUESTA PARA DETERMINAR EL COEFICIENTE DE CONOCIMIENTO DE LOS EXPERTOS

Objetivo: Determinar el nivel de competencia de los posibles expertos en la temática de la investigación.

Usted fue seleccionado como posible experto, teniendo en cuenta su aval y experiencia en el campo de desarrollo y modelado de software. Se le solicita que responda las siguientes interrogantes con el objetivo de poder llevar a feliz término la investigación. Se le agradece de antemano su cooperación. Muchas Gracias.

Temática que se investiga: Modelado de sistemas RESTful.

Nombre y apellidos: _____

Centro de trabajo: _____

Grado científico: _____ Categoría docente: _____

Años de experiencia docente: _____ Asignatura: _____ Grado: _____

1- Se solicita que usted valore su nivel de competencia sobre la problemática que se investiga marcando con una cruz el valor que considere en una escala de 1 a 10 (donde la máxima competencia se corresponde con el # 10).

1	2	3	4	5	6	7	8	9	10

2- En la siguiente tabla se le propone que indique con una cruz en cada fila, el grado de influencia (alto, medio, o bajo) que tiene en sus criterios cada fuente de argumentación y marque con una “X” la que considere que más ha fluido.

Fuentes de Argumentación	**Nivel de Influencia de cada cuente**		
	(A) Alto	**(M) Medio**	**(B) Bajo**
Análisis realizado por usted.			
Experiencia en el tema de investigación.			
Trabajos de autores nacionales.			
Trabajos de autores extranjeros.			
Su propio conocimiento del tema.			
Su intuición			

Los expertos seleccionados para formar parte del grupo para la validación de la propuesta fueron aquellos cuyos resultados arrojaron un K alto y medio.

Para la validación de la propuesta se tuvo en cuenta la opinión de 8 expertos seleccionados para la ejecución del método. Para la selección de los expertos se apeló a la experiencia de los mismos en cuanto al objeto de estudio y campo de acción de la investigación, seleccionándose 7 Máster en

Ciencias y un Doctor en Ciencias para conformar el panel de expertos. Todos ocupan cargos en las diferentes esferas productivas de la Universidad de las Ciencias Informáticas. Los resultados se muestran a continuación:

ID de experto	Conocimiento	P1	P2	P3	P4	P5	P6	Ka	Kc	K	Competencia
E1	10	0.3	0.4	0.05	0.05	0.05	0.05	0.9	1	0.95	Alto
E2	7	0.3	0.4	0.05	0.05	0.05	0.05	0.9	0.7	0.80	Alto
E3	7	0.2	0.4	0.05	0.05	0.05	0.05	0.8	0.7	0.75	Medio
E4	8	0.2	0.5	0.05	0.05	0.05	0.05	0.9	0.8	0.85	Alto
E5	9	0.3	0.5	0.05	0.05	0.05	0.05	1	0.9	0.95	Alto
E6	8	0.2	0.5	0.05	0.05	0.05	0.05	0.9	0.8	0.85	Alto
E7	8	0.2	0.4	0.05	0.05	0.05	0.05	0.8	0.8	0.80	Alto
E8	7	0.2	0.5	0.05	0.05	0.05	0.05	0.9	0.7	0.80	Alto

Tabla 7: Coeficiente de competencia de los expertos.

El panel de expertos queda de la siguiente manera, 7 expertos posee un K alto mientras que 1 posee un K medio; por tanto se obtiene un 12.5 por ciento de competencia media y un 87.5 por ciento de competencia alta.

Figura 24: Representación del coeficiente de competencia

3.2.2 Elaboración y lanzamiento de los cuestionarios

Se formula el cuestionario que se presentará a los expertos para evaluar sus opiniones. Las preguntas deben ser precisas, independientes, de manera que faciliten la respuesta por parte de los encuestados. Es importante definir con precisión que el campo de investigación abarque los conocimientos de los expertos seleccionados para precisar que todos poseen la misma noción de este campo.

Una vez seleccionados los expertos, se prosigue con la elaboración de la encuesta, para lo cual se

hace necesario confeccionar un cuestionario que se adapte a las características de los expertos.

Se utilizó un Cuestionario de Validación el cual posee como objetivo principal la validación de los elementos básicos que conforman la propuesta. Este cuestionario no representa solamente un documento que contiene una lista de afirmaciones para su validación sino que es el documento que consigue que los expertos no interactúen entre si evitando los roces sociales indeseados y de esta forma eliminando el efecto líder que pueden causar algunos expertos. El cuestionario presenta 17 preguntas las cuales están orientadas a aspectos críticos de la solución propuesta, con los cuales se puede asegurar la validación de la solución en general.

El cuestionario fue creado de forma tal que las respuestas fueran categorizadas en muy adecuado (C1), bastante adecuado (C2), adecuado (C3), poco adecuado (C4) y no adecuado (C5).

El cuestionario para la validación se muestra a continuación.

ENCUESTA A EXPERTOS PARA LA VALIDACIÓN DEL MODELO

Compañero (a):

La presente encuesta forma parte de la aplicación del método de valoración de expertos. Con este fin se solicita su valiosa colaboración para evaluar si las técnicas y herramientas que se proponen son correctas, para lograr este objetivo se han elaborado un conjunto de preguntas que permiten medir la efectividad del modelo. De antemano se le asegura que nadie puede saber quién es el encuestado y además, se garantiza que sus opiniones se tendrán en cuenta para la posterior aplicación del modelo.

Preguntas	Criterio del Experto				
	C1	C2	C3	C4	C5
1. La propuesta de un mecanismo para el modelado de sistemas ‹‹*RESTful*›› es necesario para su desarrollo.					
2. La utilización de diagramas resulta una técnica necesaria para un buen modelado de *‹‹software››.*					
3. El uso de herramientas de apoyo para el modelado en el desarrollo de *‹‹software››* es necesario para una clara comprensión del sistema a desarrollar.					
4. *UML* es el lenguaje más indicado para modelar el desarrollo de sistemas ‹‹*RESTful*›› en *RUP*.					
5. Lograr una extensión de *UML* es una solución factible al problema estudiado.					
6. Se proponen tres aspectos fundamentales para la extensión de *UML*. Evalúe el empleo de cada uno de ellos en la propuesta.					
6.1 Definición de estereotipos.					
6.2 Definición de restricciones					

6.3 Definición de los valores etiquetados					
7. La propuesta cuenta con una descripción detallada de todos los estereotipos, restricciones y valores etiquetados definidos en la extensión.					
8. Uno de los procesos fundamentales en el método es la selección de los recursos como base para el modelado de sistemas ‹‹*RESTful*››.					
9. La propuesta expone la organización de los recursos de manera jerárquica.					
10. La propuesta detalla:					
10.1 Los recursos y sus propiedades					
10.2 La jerarquía entre recursos.					
10.3 Las relaciones entre recursos					
10.4 La correcta gestión de los recursos					
11. La propuesta cubre la necesidad de modelado de sistemas ‹‹*RESTful*››.					
12. La propuesta cubre las restricciones que expone el estilo arquitectónico REST.					
13. Exprese otros criterios o recomendaciones que pudieran servir para perfeccionar el modelo propuesto:					

Gracias por su colaboración

3.2.3 Explotación de resultados

Los expertos que conforman el panel recibieron un resumen de la propuesta de solución como documentación primaria para responder los temas encuestados, además del cuestionario con un total de 17 preguntas. El cuestionario fue enviado vía correo, con una breve explicación de las condiciones prácticas del desarrollo de la encuesta (plazo de respuesta y garantía de anonimato), además de ser presentado en formato duro a cada experto. Se realizó una sola ronda de preguntas y luego se prosiguió a analizar los resultados. Con el objetivo de recoger y visualizar los resultados aportados se fueron confeccionando tablas. Los resultados se recogieron en una tabla de doble entrada como la siguiente:

Tabla de Frecuencias Acumuladas							
No	**Elementos**	**C1**	**C2**	**C3**	**C4**	**C5**	**Total**
1	1	5	0	3	0	0	8
2	2	7	1	0	0	0	8
3	3	5	3	0	0	0	8
4	4	2	4	2	0	0	8
5	5	2	3	3	0	0	8
6	6.1	4	3	1	0	0	8
7	6.2	4	3	1	0	0	8
8	6.3	4	2	1	1	0	8
9	7	3	3	2	0	0	8
10	8	5	3	0	0	0	8
11	9	5	3	0	0	0	8
12	10.1	4	3	1	0	0	8
13	10.2	5	2	1	0	0	8
14	10.3	4	2	2	0	0	8
15	10.4	2	4	2	0	0	8
16	11	1	5	2	0	0	8
17	12	3	5	0	0	0	8

Tabla 8: Frecuencias acumuladas.

La tabla de frecuencias acumuladas quedaría representada gráficamente como lo demuestra la Figura 25:

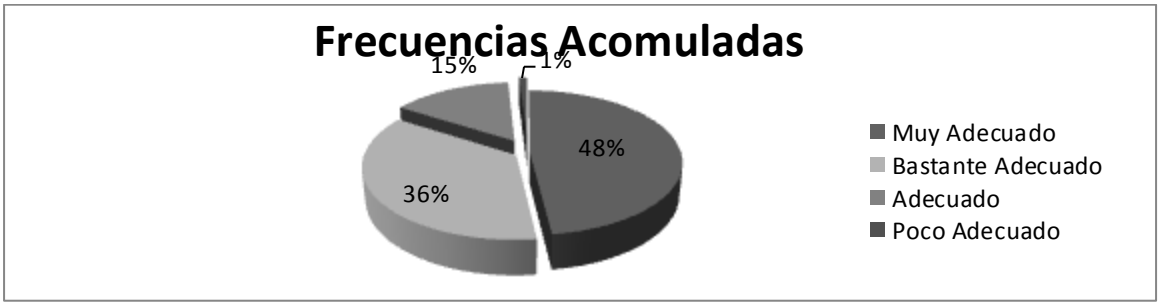


Figura 25: Representación de las frecuencias acumuladas

Luego de capturar los datos se realizan los siguientes pasos para obtener los resultados:

Primer paso: Se construye una tabla de frecuencias absolutas acumuladas.

Tabla de Frecuencias Absolutas Acumuladas						
No	**Elementos**	**C1**	**C2**	**C3**	**C4**	**NA**
1	1	5	5	8	8	8
2	2	7	8	8	8	8
3	3	5	8	8	8	8
4	4	2	6	8	8	8
5	5	2	5	8	8	8
6	6.1	4	7	8	8	8
7	6.2	4	7	8	8	8
8	6.3	4	6	7	8	8

9	7	3	6	8	8	8
10	8	5	8	8	8	8
11	9	5	8	8	8	8
12	10.1	4	7	8	8	8
13	10.2	5	7	8	8	8
14	10.3	4	6	8	8	8
15	10.4	2	6	8	8	8
16	11	1	6	8	8	8
17	12	3	8	8	8	8

Tabla 9: Frecuencias absolutas acumuladas.

Observación: En la frecuencia acumulada desaparece la última columna.

Segundo paso: Se copia la tabla anterior y se borran los resultados numéricos. Ahora, en esta nueva tabla, se construye la tabla de frecuencias relativas acumulativas.

Los datos de la tabla 10 se obtienen dividiendo cada uno de los números de la tabla 9 por el número total de expertos “8”.

Tabla de Frecuencias Relativas Acumulativas						
No	**Elementos**	**C1**	**C2**	**C3**	**C4**	**C5**
1	1	0,625	0,625	0,9999	0,9999	0,9999
2	2	0,875	0,9999	0,9999	0,9999	0,9999
3	3	0,625	0,9999	0,9999	0,9999	0,9999
4	4	0,25	0,75	0,9999	0,9999	0,9999
5	5	0,25	0,625	0,9999	0,9999	0,9999
6	6.1	0,5	0,875	0,9999	0,9999	0,9999
7	6.2	0,5	0,875	0,9999	0,9999	0,9999
8	6.3	0,5	0,75	0,875	0,9999	0,9999
9	7	0,375	0,75	0,9999	0,9999	0,9999
10	8	0,625	0,9999	0,9999	0,9999	0,9999
11	9	0,625	0,9999	0,9999	0,9999	0,9999
12	10.1	0,5	0,875	0,9999	0,9999	0,9999
13	10.2	0,625	0,875	0,9999	0,9999	0,9999
14	10.3	0,5	0,75	0,9999	0,9999	0,9999
15	10.4	0,25	0,75	0,9999	0,9999	0,9999
16	11	0,125	0,75	0,9999	0,9999	0,9999
17	12	0,375	0,9999	0,9999	0,9999	0,9999

Tabla 10: Frecuencias relativas acumuladas.

Tercer paso: Se buscan las imágenes de los elementos de la tabla anterior por medio de la función *(Dist.Normal.Standard.Inv).*

A la misma tabla se le adicionan tres columnas y una fila para colocar los resultados que se mencionan a continuación: suma de las columnas, suma de filas (S), promedio de las columnas, promedios de las filas (P), N (se divide la suma de las sumas entre el resultado de multiplicar el número de indicadores

por el número de preguntas) y el valor N menos P. La tabla siguiente resume lo dicho anteriormente.

No	Elementos	C1	C2	C3	C4	S	P	N = 1.77 N-P	Grado de Adecuación
1	1	0,32	0,32	3,72	3,72	8,07	2,02	-0,25	Muy Adecuado
2	2	1,15	3,72	3,72	3,72	12,31	3,08	-1,31	Muy Adecuado
3	3	0,32	3,72	3,72	3,72	11,47	2,87	-1,10	Muy Adecuado
4	4	-0,67	0,67	3,72	3,72	7,44	1,86	-0,09	Muy Adecuado
5	5	-0,67	0,32	3,72	3,72	7,08	1,77	-0,0005	Bastante Adecuado
6	6.1	-1,39	1,15	3,72	3,72	8,59	2,15	-0,38	Muy Adecuado
7	6.2	-1,39	1,15	3,72	3,72	8,59	2,15	-0,38	Muy Adecuado
8	6.3	-1,39	0,67	1,15	3,72	5,54	1,38	0,38	Bastante Adecuado
9	7	-0,32	0,67	3,72	3,72	7,79	1,95	-0,18	Muy Adecuado
10	8	0,32	3,72	3,72	3,72	11,47	2,87	-1,10	Muy Adecuado
11	9	0,32	3,72	3,72	3,72	11,47	2,87	-1,10	Muy Adecuado
12	10.1	-1,39	1,15	3,72	3,72	8,59	2,15	-0,38	Muy Adecuado
13	10.2	0,32	1,15	3,72	3,72	8,91	2,23	-0,46	Muy Adecuado
14	10.3	-1,39	0,67	3,72	3,72	8,11	2,03	-0,26	Muy Adecuado
15	10.4	-0,67	0,67	3,72	3,72	7,44	1,86	-0,09	Muy Adecuado
16	11	-1,15	0,67	3,72	3,72	6,96	1,74	0,03	Bastante Adecuado
17	12	-0,32	3,72	3,72	3,72	10,84	2,71	-0,94	Muy Adecuado
Suma		-1,07	27,88	60,65	63,22	142,04	**Total Muy Adecuado**		14
Putos de Corte		-0,06	1,64	3,57	3,72	**Total Bastante Adecuado**			3

Tabla 11: Puntos de corte.

El promedio de las sumas obtenidas en las cuatro primeras columnas dan los puntos de cortes, los cuales sirven para determinar la categoría o grado de adecuación de cada paso de la metodología según la opinión de los expertos consultados, los rangos quedarían como sigue:

Muy Adecuado	Bastante Adecuado	Adecuado	Poco Adecuado	No Adecuado
Menor -0,06	(-0.06 -- 1,64)	(1.64 -- 3,57)	(3.57 -- 3,72)	Mayor 3.72

Tabla 12: Rangos obtenidos a partir de los puntos de cortes.

Si para alguno de los criterios propuestos se obtiene un resultado poco adecuado o adecuado, este criterio debe ser reelaborado, por resultar poco adecuado según el criterio de los expertos consultados, y hacer una nueva iteración del método, el resto pueden darse por concluidos en cuanto a su elaboración teórica.

A continuación se muestran dos gráficos que muestran los resultados obtenidos. El primero muestra el número de categorías otorgadas por los 8 expertos en la validación de la propuesta.

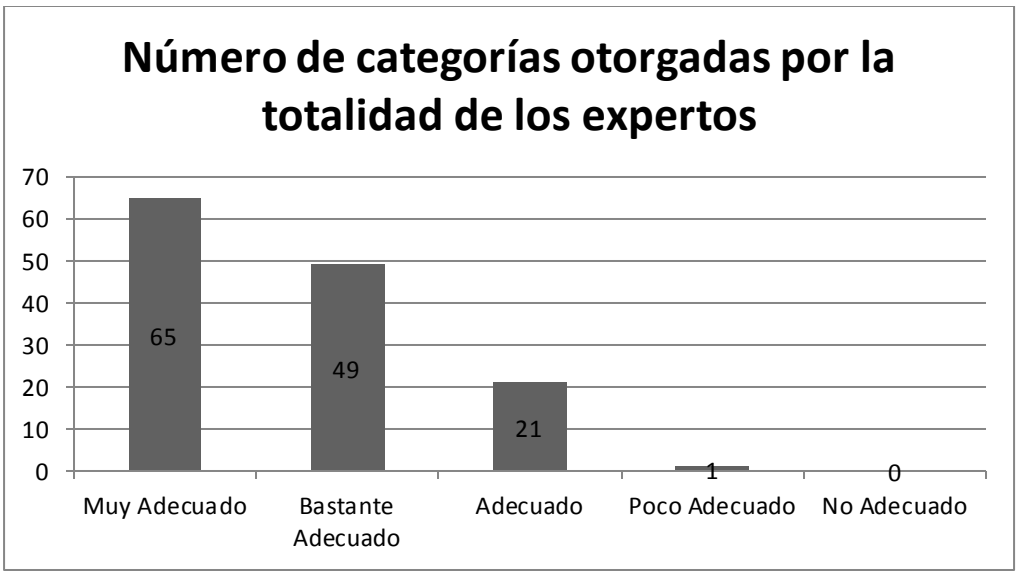


Figura 26: Gráfico de categorías otorgadas por los expertos.

La segunda tabla que se muestra a continuación representa los puntos de cortes por las preguntas realizadas a los expertos, evidenciando en qué rango se encuentra la pregunta con respecto al punto de corte para determinar si la misma es muy adecuada, bastante adecuada, adecuada o poco adecuada.

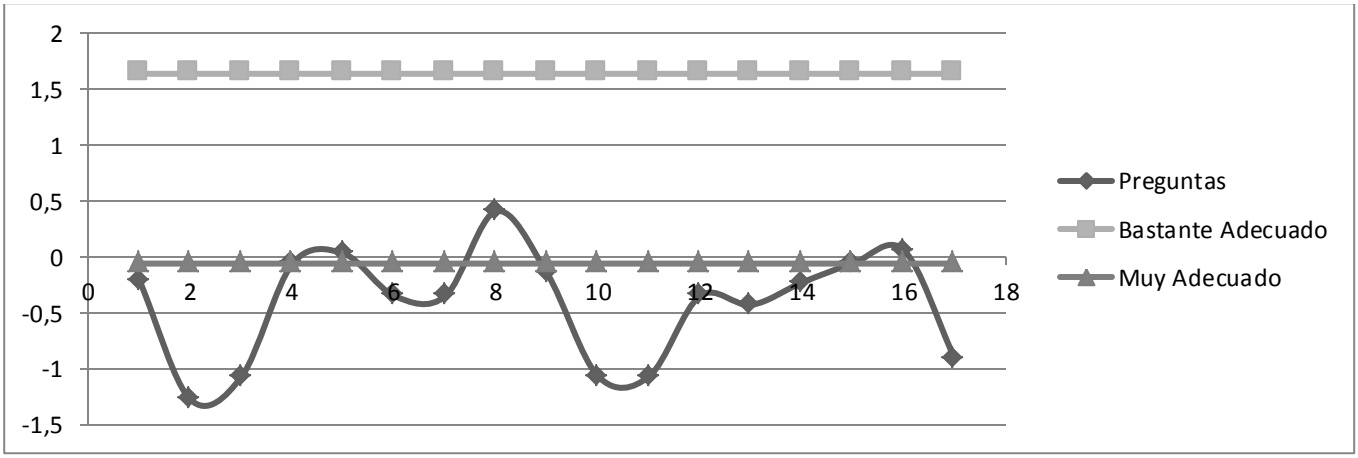


Figura 27: Representación de los puntos de corte.

Las preguntas que resultarían muy adecuadas serían las que poseen valores menores que -0.06; bastante adecuado los valores entre -0.06 y 1.64; adecuado entre 1.64 y 3.57; poco adecuado entre 3.57 y 3.72; no adecuado más de 3.72. De esta manera, se obtienen los siguientes resultados por afirmaciones:

No	Elementos	Grado de Adecuación
1	1	Muy Adecuado
2	2	Muy Adecuado
3	3	Muy Adecuado
4	4	Muy Adecuado
5	5	Bastante Adecuado
6	6.1	Muy Adecuado
7	6.2	Muy Adecuado
8	6.3	Bastante Adecuado
9	7	Muy Adecuado
10	8	Muy Adecuado
11	9	Muy Adecuado
12	10.1	Muy Adecuado
13	10.2	Muy Adecuado
14	10.3	Muy Adecuado
15	10.4	Muy Adecuado
16	11	Bastante Adecuado
17	12	Muy Adecuado

Tabla 13: Resultados por afirmación.

De acuerdo con los resultados logrados mostrados en la tabla anterior se puede dar por concluida la validación en cuanto a su elaboración teórica, ya que los resultados arrojados fueron satisfactorios. Se presenta a continuación un resumen con los resultados obtenidos.

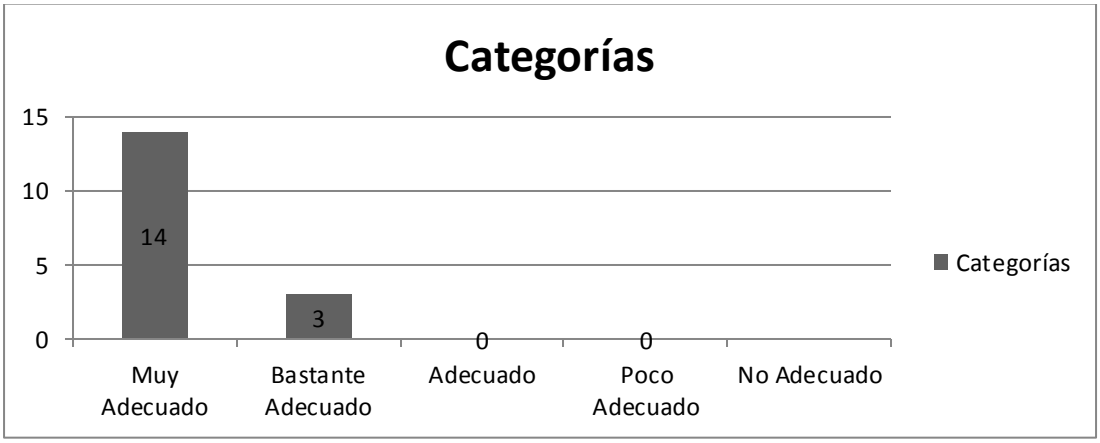


Figura 28: Representación de los resultados generales

3.3 Conclusiones parciales

En este capítulo se validó la propuesta de solución realizada para modelar aplicaciones *‹‹RESTful››,* utilizando el método de *Delphi*. Se seleccionaron 8 expertos y se validaron 17 afirmaciones relacionadas con la solución planteada. Los resultados obtenidos fueron satisfactorios, evaluados de muy adecuado y bastante adecuado. De este modo se concluye que la propuesta realizada es válida y que puede ser empleada en proyectos que deseen iniciar el desarrollo de aplicaciones *‹‹RESTful››.*

CONCLUSIONES

Una vez finalizado el trabajo de diploma se puede concluir que:

- Se realizó un estudio enmarcado en cuanto al objeto de estudio y campo de acción de la investigación, en el cual se evidenció la necesidad de lograr una notación estándar para el modelado de aplicaciones *‹‹RESTful››.*
- Con la extensión a *UML* como propuesta de solución de la investigación, los sistemas *‹‹RESTful››* pueden contar con un mecanismo de modelado para la documentación durante su desarrollo, como se evidencia en el caso de estudio analizado con la solución.
- La propuesta de solución define una guía que puede ser empleada en cualquier proyecto que desee iniciar el desarrollo de aplicaciones orientadas a recursos.
- La propuesta de solución fue validada mediante el método de experto *Delphi*, arrojando resultados de muy aceptada y bastante aceptada, por los que se concluye de satisfactoria y valida la solución.

RECOMENDACIONES

Para lograr una implantación eficiente de la solución propuesta se tienen las siguientes recomendaciones:

- Trabajar en función de la integración de la propuesta a *framework*, entornos de desarrollo como *NetBeans, Eclipse* y otros que permitan la generación automática de código en la medida de lo posible, aumentado así los niveles de productividad.
- Analizar los resultados de la solución de manera que se pueda perfeccionar constantemente la propuesta.

REFERENCIAS BIBLIOGRÁFICAS

1. **Ivar Jacobson, Grady Booch, James Rumbaugh.** *El Proceso Unificado de Desarrollo de Software.* 1999.

2. **Universidad Carlo III de Madrid, Diseño de software avanzado, Dpto. de Informática.** OpenCourseWare. *La necesidad de modelar.* [En línea] [Citado el: 08 de 01 de 2012.] http://ocw.uc3m.es/ingenieria-informatica/diseno-de-software-avanzado/material-de-clase-1/02-La_necesidad_de_Modelar.pdf.

3. *Curso de Ingeniería de Software para Web.* **6, Conferencia.** s.l. : Universidad de las Ciencias Informáticas.

4. **Pérez, Juan Diego.** *Notaciones y lenguajes de procesos. Una visión global.*

5. **Wendy Boggs, Michael Boggs.** *Mastering UML with Rational Rose.* 2002.

6. Visual Paradigm. *Visual Paradigm.* [En línea] [Citado el: 24 de 11 de 2011.] http://www.visual-paradigm.com/..

7. **Mauro Callejas Cuervo, Oscar Yovany Baquero Moreno.** ER VISUAL 1.00, Open Source Software. *ER VISUAL 1.00, Open Source Software.* [En línea] 2005. [Citado el: 15 de 03 de 2012.] http://ervisual.hostoi.com/pdf/er_tecnico.pdf.

8. **Stephen J. Mellor, Anthony N. Clark, Takao Futagami.** MDD – MODEL-DRIVEN DEVELOPMENT. [En línea] [Citado el: 04 de 04 de 2012.] http://www.google.com.cu/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCYQFjAA&url=http%3A%2F%2Fproyectograd o2010.googlecode.com%2Ffiles%2FMDD-RESUMEN.doc&ei=-ql8T8-WJoiSgQfczsn9Cw&usg=AFQjCNEsDJCYKfMU165-bgib8RBqXQbh5Q.

9. Object Management Group. [En línea] [Citado el: 15 de 03 de 2012.] http://jeffsutherland.org/omg/cf/96-01-04.pdf.

10. **Fielding, Roy Tomas.** *Architectural Styles and the Design of Network-based Software Architectures.* 2000.

11. —. Architectural Styles and the Design of Network-based Software Architectures. 2000.

12. —. 5.1.3 Stateless. [aut. libro] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* 2000.

13. —. 5.1.4 Cache. [aut. libro] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* 2000.

14. —. 5.1.5 Uniform Interface. [aut. libro] Roy Fielding. *Architectural Styles and the Design of Network-*

based Software Architectures. 2000.

15. —. 5.1.6 Layered System. [aut. libro] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* 2000.

16. Vladislav Ponomarev, Carl Osipov. *Colaboración más inteligente para la Industria de la Educación utilizando Lotus Connections, Parte 1: Integrar Lotus Connections con una aplicación web RESTful.* [En línea] 2011. [Citado el: 29 de 01 de 2012.] www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=Smarter+Collaboration+for+the+Education+Industry+using+Lotus+Connections.

17. **Schreier, Silvia.** *Modeling RESTful applications.* University of Hagen: : s.n., 2011.

18. **Alba, Julio.** *SOA ARQUITECTURA ORIENTADA A SERVICIO.* 2009.

19. **Noa, Daliana.** *Gestión de metadatos asociados a las tipologías documentales definidas en el Gestor de Documentos Administrativos eXcriba.* Habana : Universidad de las Ciencias Informáticas, Junio 2012.

20. **Quesada, Gilberto.** *Método Delphi.*

21. Grupo de Tecnologías de la Información y las Comunicaciones. El método Delphi. [En línea] 2006. [Citado el: 26 de 04 de 2012.] http://www.gtic.ssr.upm.es/encuestas/delphi.htm..

22. **Moreno Izquierdo, Paula Cecilia , y otros.** Metodo Delphi. [En línea] [Citado el: 26 de 05 de 2012.] http://74.125.45.132/search?q=cache:oX1iZz_XnuIJ:www.uam.es/personal_pdi/economicas/rmc/prevision/pdf/DELPHI.ppt+metodo+delphi%2Bdefinicion&cd=2&hl=es&ct=clnk&gl=cu..

23. **Raimundo Llerena Ferrer, Pedro Martinez Rey.** Microsoft Visio 2007. *Propuesta de modelo para desarrollar el modelado del negocio en proyectos BPM/SOA.* Habana : s.n., 2009.

1. **Ivar Jacobson, Grady Booch, James Rumbaugh.** *El Proceso Unificado de Desarrollo de Software.* 1999.

2. **Universidad Carlo III de Madrid, Diseño de software avanzado, Dpto. de Informática.** OpenCourseWare. *La necesidad de modelar.* [En línea] [Citado el: 08 de 01 de 2012.] http://ocw.uc3m.es/ingenieria-informatica/diseno-de-software-avanzado/material-de-clase-1/02-La_necesidad_de_Modelar.pdf.

3. *Curso de Ingeniería de Software para Web.* **6, Conferencia.** s.l. : Universidad de las Ciencias Informáticas.

4. **Pérez, Juan Diego.** *Notaciones y lenguajes de procesos. Una visión global.*

5. **Wendy Boggs, Michael Boggs.** *Mastering UML with Rational Rose.* 2002.

6. Visual Paradigm. *Visual Paradigm.* [En línea] [Citado el: 24 de 11 de 2011.] http://www.visual-paradigm.com/..

7. **Mauro Callejas Cuervo, Oscar Yovany Baquero Moreno.** ER VISUAL 1.00, Open Source Software. *ER VISUAL 1.00, Open Source Software.* [En línea] 2005. [Citado el: 15 de 03 de 2012.] http://ervisual.hostoi.com/pdf/er_tecnico.pdf.

8. **Stephen J. Mellor, Anthony N. Clark, Takao Futagami.** MDD – MODEL-DRIVEN DEVELOPMENT. [En línea] [Citado el: 04 de 04 de 2012.] http://www.google.com.cu/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCYQFjAA&url=http%3A%2F%2Fproyectograd o2010.googlecode.com%2Ffiles%2FMDD-RESUMEN.doc&ei=-ql8T8-WJoiSgQfczsn9Cw&usg=AFQjCNEsDJCYKfMU165-bgib8RBqXQbh5Q.

9. Object Management Group. [En línea] [Citado el: 15 de 03 de 2012.] http://jeffsutherland.org/omg/cf/96-01-04.pdf.

10. **Fielding, Roy Tomas.** *Architectural Styles and the Design of Network-based Software Architectures.* 2000.

11. —. Architectural Styles and the Design of Network-based Software Architectures. 2000.

12. —. 5.1.3 Stateless. [aut. libro] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* 2000.

13. —. 5.1.4 Cache. [aut. libro] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* 2000.

14. —. 5.1.5 Uniform Interface. [aut. libro] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* 2000.

15. —. 5.1.6 Layered System. [aut. libro] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* 2000.

16. Vladislav Ponomarev, Carl Osipov. *Colaboración más inteligente para la Industria de la Educación utilizando Lotus Connections, Parte 1: Integrar Lotus Connections con una aplicación web RESTful.* [En línea] 2011. [Citado el: 29 de 01 de 2012.] www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=Smarter+Collaboration+for+the+Education+Industry+using+Lotus+Connections.

17. **Schreier, Silvia.** *Modeling RESTful applications.* University of Hagen: : s.n., 2011.

18. **Alba, Julio.** *SOA ARQUITECTURA ORIENTADA A SERVICIO.* 2009.

19. **Noa, Daliana.** *Gestión de metadatos asociados a las tipologías documentales definidas en el Gestor de Documentos Administrativos eXcriba.* Habana : Universidad de las Ciencias Informáticas, Junio 2012.

20. **Quesada, Gilberto.** *Método Delphi.*

21. Grupo de Tecnologías de la Información y las Comunicaciones. El método Delphi. [En línea] 2006. [Citado el: 26 de 04 de 2012.] http://www.gtic.ssr.upm.es/encuestas/delphi.htm..

22. **Moreno Izquierdo, Paula Cecilia , y otros.** Metodo Delphi. [En línea] [Citado el: 26 de 05 de 2012.] http://74.125.45.132/search?q=cache:oX1iZz_XnuIJ:www.uam.es/personal_pdi/economicas/rmc/prevision/pdf/DELPHI.ppt+metodo+delphi%2Bdefinicion&cd=2&hl=es&ct=clnk&gl=cu..

23. **Raimundo Llerena Ferrer, Pedro Martinez Rey.** Microsoft Visio 2007. *Propuesta de modelo para desarrollar el modelado del negocio en proyectos BPM/SOA.* Habana : s.n., 2009.

24. Wikipedia. *Wikipedia.* [En línea] [Citado el: 2 de 11 de 2011.] http://wikipedia.com.

25. Ecured. *Ecured*. [En línea] [Citado el 2 de 11 de 2011.] http://ecured.cu

26. Google. *Google* [En línea] [Citado el 2 de 11 de 2011.] http://google.com

27. Biblioteca de la Universidad de las Ciencias Informáticas. *Biblioteca de la Universidad de las Ciencias Informáticas.* [En línea] [Citado el 12 de 04 de 2012.] http://biblioteca.uci.cu

28. Entorno Virtual de Aprendizaje. e*va* [En línea] [Citado el 12 de 04 de 2012.] http://eva.uci.cu

29. Fabián Bermeo Pérez. *Fabián Bermeo Pérez.* [En línea] [Citado el 18 de 04 de 2012.] http://fabianbermeop.blogspot.com/

30. PmWiki. *PmWiki.* [En línea] [Citado el 6 de 11 de 2011.] http://petra.euitio.uniovi.es/~i6950404/wiki/pmwiki.php?n=Tema6

31. Dosiedas. *Dosideas* [En línea] [Citado el 8 de 11 de 2011] http://www.dosideas.com/

32. Infotechnology. *Infotechnology* [En línea] [Citado el 8 de 11 de 2011] http://www.infotechnology.com/

33. Scribd. *Scribd.* [En línea] [Citado el 27 de 11 de 2011] http://es.scribd.com/

34. Infociberland. *Infociberland* [En línea] [Citado el 16 de 10 de 2011] http://infociberland.comxa.com/

GLOSARIO DE TÉRMINOS

HTML: Lenguaje de marcado de hipertexto que se usa para crear documentos de hipertexto que son de plataformas independientes.

Fuente: *T. Berners-Lee, D. Connolly , Hypertext Markup Language - 2.0, Disponible en: http://www.hjp.at/doc/rfc/rfc1866.html*

HTTP: Hypertext Transfer Protocol o HTTP (en español protocolo de transferencia de hipertexto) es el protocolo usado en cada transacción de la World Wide Web.

Fuente: *HTTP - Hypertext Transfer Protocol, Disponible en:* http://www.w3.org/Protocols/

JavaScript Object Notation: Es un formato ligero, basado en texto, independiente del lenguaje de formato de intercambio de datos. Fue derivado de la Norma de programación del lenguaje ECMAScript. JSON define un conjunto reducido de reglas de formato para la representación portátil de datos estructurados.

Fuente: The application/json Media Type for JavaScript Object Notation (JSON), Disponible en:

https://tools.ietf.org/html/rfc4627

Metadato: Cathro, apunta que aún cuando existe cierta dificultad para definir claramente este concepto, su esencia establece que son los datos que describen un recurso de información o ayuda a proveer acceso a este, y coincide en que un registro del catálogo de una biblioteca es una colección de elementos de metadatos.

Fuente: *¿Catalogación en el entorno digital?: una breve aproximación a los metadatos, Disponible en: http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1024-94352006000500009*

Metamodelo: Los metamodelos son modelos para crear otros modelos y se usan en el contexto de la arquitectura de MOF (pertenecen a nivel M2).

Fuente: *Dpto. de Tecnologías y Sistemas de Información, disponible en: http://www.uclm.net/dep/tsi/pdf/UCLM-TSI-001.pdf*

RESTful: Es el nombre que toman las aplicaciones desarrolladas sobre el estilo arquitectónico REST.

Fuente: Fielding, Roy T. *"Architectural Styles and the Design of Network-based Software Architectures".* 2000.

Sistemas hipermedias distribuidos: sistemas que posibiliten elaborar y recuperar documentos complejos cuyas partes (textos, gráficos, imágenes, animaciones, sonido, índices de bases de datos, etc.) pueden estar distribuídas en distintos ordenadores conectados a la Internet, es decir, en cualquier lugar del mundo (BERNERS-LEE, CAILLAU, GROFF, POLLERMAN, 1992a, 1992b).

Fuente: *Hipermedia distribuido en el Mac: el proyecto World-Wide Web, disponible en: http://tecnologiaedu.us.es/cuestionario/bibliovir/15.pdf*

Software: Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.

Fuente: *REAL ACADEMIA ESPAÑOLA, Disponible en: http://buscon.rae.es/draeI/SrvltGUIBusUsual?LEMA=software*

URI: A Uniform Resource Identifier (URI) es una secuencia compacta de caracteres que identifica un recurso abstracto o físico.

Fuente: *Uniform Resource Identifier (URI): Generic Syntax, Disponible en: http://merlot.tools.ietf.org/html/rfc3986*

URL: Un localizador de recursos uniforme, más comúnmente denominado URL (sigla en inglés de Uniform Resource Locator), es una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su localización o identificación.

Fuente: T. Berners-Lee, L. Masinter, M. McCahill , Uniform Resource Locators (URL), Disponible en: http://www.hjp.at/doc/rfc/rfc1738.html

XML: inglés de eXtensible Markup Language ('lenguaje de marcas extensible'), Extensible Markup Language, abreviado XML, describe una clase de objetos de datos llamados documentos XML y parcialmente describe el comportamiento de los programas de ordenador que los procesa.

Fuente: *Extensible Markup Language (XML) 1.0 (Fifth Edition), Disponible en: http://www.w3.org/TR/REC-xml/*