Universidad de las Ciencias Informáticas

Facultad 6

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Título: “Perfil de UML para los proyectos de la línea Soluciones Integrales”

Autor:

Yaneisy González Blanco

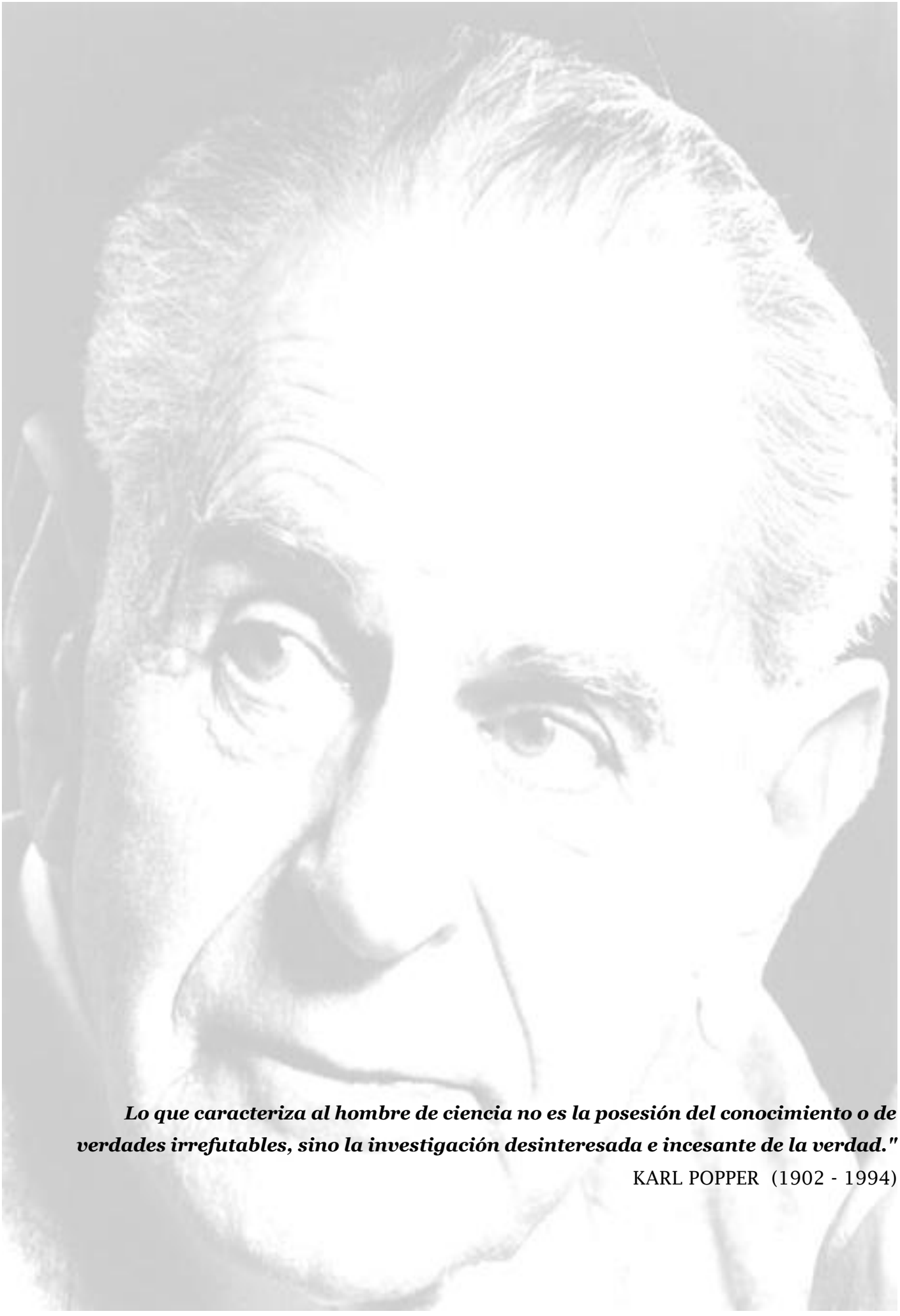
Tutores:

Ing. Armando Robert Lobo

Ing. Diana Monné Roque

La Habana, Junio 2011

“Año 53 de la Revolución”



Lo que caracteriza al hombre de ciencia no es la posesión del conocimiento o de verdades irrefutables, sino la investigación desinteresada e incesante de la verdad."

KARL POPPER (1902 - 1994)

Declaración de autoría

Declaro que soy la única autora de la presente tesis y autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio.

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

_____

Yaneisy González Blanco

_____ _____

Ing. Diana Monné Roque Ing. Armando Robert Lobo

Datos de contacto

Tutores:

- Ing. Armando Robert Lobo

Universidad de las Ciencias Informáticas

Email: arobert@uci.cu

- Ing. Diana Monné Roque

Universidad de las Ciencias Informáticas

Email: dmonne@uci.cu

Agradecimientos

A las tres personas que más quiero en mi vida y que adoro con todo mi corazón, que me guiaron en todo momento, que me aman y protegen siempre, mis padres y hermana: Bárbara, Emerio y Carito.

A mis dos abuelas y abuelo, por transmitirme un mundo de experiencia, sabiduría y amor durante toda mi vida.

A mis tías y tío por formar parte de mi vida y estar presentes en cada etapa de ella.

A mi novio, por ser tan especial y maravilloso, por su paciencia y por soportar mis momentos de pesimismo. Por quererme mucho.

A mis amigas de los cinco años de universidad Wilmis, Ana, Azalia y Katia, por demostrarme día a día que la importancia de los hechos es relativa, que lo más importante es levantarse siempre y continuar caminando un paso a la vez.

A mis viejos y nuevos amigos, que compartieron cada alegría, cada tristeza, cada victoria y cada derrota como si fuesen mis familiares. Muy especial a mi amiga Rachi, por ser mi amiga desde que tengo uso de razón y a mi maravilloso grupo de cadetes de tercer año por ser el mejor del mundo. Los quiero mucho a todos.

A mis tutores Diana y Lobo por guiarme en todo momento, con paciencia y sabiduría.

A todos los profesores que me enseñaron todo lo que sé.

A todas las personas que de una forma u otra colaboraron en el desarrollo de esta investigación y en mi preparación como futura ingeniera en Ciencias Informáticas.

Yaneisy González Blanco

Dedicatoria

Dedico este trabajo y todo lo que he logrado hasta el momento en mi vida:

A mis padres,

A mi hermana Carito

Y a Leonard.

Resumen

El desarrollo de proyectos a partir de Líneas de Productos de Software (LPS), constituye un progreso en el desarrollo de sistemas de gestión, debido a que simplifica el trabajo y crea una organización dentro de la empresa, que permite que el producto se termine en menor tiempo, con mayor eficiencia y con más calidad. El Centro de Tecnologías de Gestión de Datos (DATEC), ha tomado esta idea para su mejor funcionamiento. Una de estas LPS que conforman el centro, es la línea Soluciones Integrales.

El presente trabajo de diploma constituye una propuesta de perfil de UML (Lenguaje Unificado de Modelado), encaminada a favorecer el entendimiento y la comunicación entre el Grupo de Análisis, el Grupo de Desarrollo y el Grupo de Arquitectura que conforman la línea Soluciones Integrales. Se realiza un estudio acerca de las tendencias presentadas por autores reconocidos e instituciones existentes, acerca de UML y sus propios mecanismos de extensión, en aras de lograr una propuesta sustentada en fundamentos teóricos de la actualidad. La propuesta de perfil fue verificada de forma satisfactoria por un panel de expertos, mediante la aplicación del método Delphi.

Palabras clave: LPS, DATEC, Soluciones Integrales, UML, perfil UML, Delphi.

TABLA DE CONTENIDOS

Introducción

La idea de una Línea de Productos de Software consiste esencialmente en el ensamblaje de partes de un sistema de software previamente elaborado, el cual se inspira en los procesos de producción de sistemas físicos, como por ejemplo la producción de aviones, vehículos, computadores, aparatos electrónicos, entre otros. La realización de esta se fundamenta en la reutilización de software, en la cual se asume la existencia de una industria de partes, sobre la base de que no es más que el proceso de crear sistemas de software a partir del existente, en lugar de desarrollarlo desde el comienzo. (1)

Una definición más formal, es que LPS es un conjunto de sistemas de software que tienen características comunes con algunas variaciones, lo común descansa en el uso recurrente de un conjunto de activos reutilizables como por ejemplo requisitos, diseños, componentes y servicios. Las LPS tienen múltiples beneficios, entre los que se puede citar la entrega de productos de software de una manera más rápida, económica y con una mejor calidad. Además producen mejoras en tiempo de entrega del producto, costos de ingeniería y tamaño del portafolio. (1)

Cuba no ha quedado exenta de todos estos adelantos, que se han venido llevando a vías de hecho en cuanto a LPS se refiere, a pesar del bloqueo y las circunstancias que lo recrudecen. Las empresas Informáticas cubanas tienen como reto ofrecer soluciones o respuestas lo más rápidas posible y de la forma más eficiente a los individuos o clientes que solicitan sus servicios. Como ejemplo de las instituciones o empresas con las que cuenta Cuba se tienen: DeSoft, TranSoft, Softel y la Universidad de las Ciencias Informáticas (UCI).

La UCI es un centro que fue creado principalmente para formar ingenieros en Ciencias Informáticas, o sea, profesionales con los que se pueda contar para hacerle frente a cualquier situación en esta rama, además se caracteriza por la producción de software, servicios informáticos, constituye un soporte de la informatización del país y se enmarca en la competitividad internacional de la industria cubana del software.

Actualmente la producción en la UCI se realiza a través de centros de desarrollo, los cuales pertenecen administrativamente a facultades. El Centro de Tecnologías de Gestión de Datos de la Facultad 6, está constituido por las líneas PostgreSQL, Soluciones Integrales, Almacenes de Datos y Bioinformática.

En la línea Soluciones Integrales, los activos de software, transitan por cada uno de los grupos que componen esta línea, estos grupos son: el Grupo de Análisis, el Grupo de Arquitectura y el Grupo de Desarrollo. El Grupo de Análisis, es el encargado de realizar los activos que describe la metodología en el tiempo establecido y la calidad requerida, los analistas, son el puente de comunicación entre los clientes y los desarrolladores y se encargan de validar durante todo el desarrollo del proyecto que se satisfagan las necesidades de los usuarios finales. El Grupo de Arquitectura define la arquitectura base de todos los productos que se desarrollarán en la línea, investiga y propone las tecnologías y herramientas candidatas para el desarrollo de los productos del centro y promueve el uso de buenas prácticas y utilización de patrones en el diseño. El Grupo de Desarrollo, implementa los productos y las herramientas del centro a partir de las propuestas del Grupo Arquitectura, siguiendo las buenas prácticas recomendadas y usando las herramientas y tecnologías propuestas por este, que se basa en lo definido por el Grupo de Análisis.

Cuando las funciones relativas a los roles tienen este grado de desagregación la comunicación se transforma en una cuestión fundamental para el éxito de un proyecto. El medio a través del cual se propaga el conocimiento entre los grupos son los artefactos de ingeniería, con énfasis particular en los modelos contenidos en las diferentes vistas de la arquitectura expresados en UML. El consenso al que se llega respecto a la interpretación de lo que expresa un determinado diagrama es en la actualidad un conocimiento tácito resultado de múltiples discusiones internas, algunas no exentas de conflicto. Mantener ese know-how desarrollado requiere en hacer explícito tal conocimiento, algo que se ha transformado en una necesidad urgente dada la continua renovación de los recursos humanos de la línea. A esto se agrega la necesidad de que al desarrollador llegue de manera precisa y sin ambigüedades las decisiones de las etapas de análisis y diseño, las cuales serán reflejadas por este en el código fuente durante la implementación.

Lo anteriormente expresado da paso al **problema de la investigación:** ¿Cómo formalizar elementos del modelado en UML de un software durante las etapas de análisis y diseño, permitiendo la traducción precisa en implementación?

Como **objeto de estudio:** Lenguaje Unificado de Modelado.

Enmarcado en el **campo de acción:** Extensión del Lenguaje Unificado de Modelado para la modelación durante las etapas de análisis, arquitectura y desarrollo.

Se puntualiza como **objetivo general**:

Definir un perfil de UML para los proyectos de la línea Soluciones Integrales que facilite la comunicación entre los grupos de Análisis, Arquitectura y el de Desarrollo.

Desglosado en los sucesivos **objetivos específicos:**

- Fundamentar conceptos y herramientas utilizadas para la elaboración de un perfil de UML.
- Describir un perfil de UML para representar las decisiones de análisis, diseño e implementación de los productos de la línea Soluciones Integrales.
- Verificar la propuesta realizada.

Para dar cumplimiento a los objetivos se precisan las siguientes **tareas de la investigación:**

1. Análisis de los mecanismos de extensión de UML y los conceptos MDD (Desarrollo Dirigido por Modelos) y MDA (Arquitectura Dirigida por Modelos).
2. Profundización de las prácticas arquitectónicas, los principios y patrones de diseño y los mecanismos de análisis, diseño e implementación utilizados en la línea Soluciones Integrales.
3. Confección del Modelo de Dominio para el Análisis, descripción de las entidades, diseño del perfil para el Análisis y semántica de los estereotipos.
4. Confección del Modelo de Dominio para el Diseño, descripción de las entidades, especificación del perfil para el diseño y semántica de los estereotipos.
5. Confección del Modelo de Dominio para la Implementación, descripción de las entidades, diseño del perfil para la Implementación y semántica de los estereotipos.
6. Verificación de la propuesta mediante el Método Delphi.

Como **resultado esperado,** se pretende obtener: Perfil de UML para representar las decisiones de análisis, diseño e implementación de los productos de la línea Soluciones Integrales.

El trabajo de diploma está estructurado de la siguiente forma:

Capítulo 1: Fundamentación teórica. En este capítulo se realiza una investigación la cual está dirigida al Lenguaje Unificado de Modelado que constituye la base para la creación de mecanismos para personalizar lenguajes de modelado (perfil de UML y metamodelos), estos mecanismos pueden ser empleados para representar las decisiones de Análisis y Diseño de los productos de la línea Soluciones Integrales. Además se hace referencia a la importancia de los modelos, para guiar el desarrollo del software, enfatizando en los siguientes estándares de la OMG (Object Management Group): MDD y MDA.

Capítulo 2: Descripción del perfil UML. En este capítulo se explica detalladamente cuáles son los pasos fundamentales a la hora de definir un perfil UML, a partir de estos pasos se precisa el mismo para el Modelo del Análisis, Diseño e Implementación para la Línea de Soluciones Integrales, DATEC. En este perfil se sistematizarán las extensiones a utilizar a la hora de confeccionar los modelos en el expediente digital. Su uso en DATEC reportará beneficios al formalizar elementos del modelado durante la etapa de análisis y diseño permitiendo su traducción precisa en implementación. El objetivo de este perfil es servir de directriz a la modelación para las vistas lógica y de implementación aumentando la expresividad del UML convencional.

Capítulo 3: Verificación del perfil propuesto. En este capítulo, se realiza un estudio relacionado al método Delphi, el cual será empleado con el objetivo de verificar la propuesta del perfil de UML elaborada en el capítulo 2. Luego del estudio de este método se selecciona un grupo de expertos que no son más que personas con el conocimiento necesario para determinar si la investigación realizada se aproxima a la realidad y a través de su criterio, facilitarán la verificación de la propuesta y darán un resultado negativo o positivo de la misma, estableciendo si el perfil de UML creado, está en condiciones de ser aplicado o no. Además la correcta elaboración del mismo, se puede comprobar a partir de lo establecido por la superestructura de UML, descrito en el capítulo anterior, ya que para la realización de este perfil, se siguieron correctamente los pasos que indica el paquete UML Profile 2.0, definido por la OMG.

Capítulo 1: Fundamentación teórica.

Introducción.

En este capítulo se realiza una investigación la cual está dirigida al Lenguaje Unificado de Modelado que constituye la base para la creación de mecanismos para personalizar lenguajes de modelado (perfil UML y metamodelo), estos mecanismos pueden ser empleados para representar las decisiones de análisis y diseño de los productos de la línea Soluciones Integrales. Además se hace referencia a la importancia de los modelos, para guiar el desarrollo del software, enfatizando en los siguientes estándares de la OMG: MDD y MDA.

1.1. Lenguajes de modelado.

La forma de representar los diseños de manera gráfica ha sido durante mucho tiempo, considerado una ventaja, ya sea en ramas de la ingeniería o la arquitectura. A partir de los inicios de la Informática en el mundo se comenzaron a utilizar diferentes formas de representar los diseños de una manera más acercada a lo personal o con algún modelo gráfico. La escasez de estandarización a la hora de representar gráficamente un modelo, no permitía que los diseños gráficos cometidos, se lograran compartir sencillamente entre los diferentes diseñadores.

Los lenguajes de modelado le hacen la vida más fácil al desarrollador de software, porque reducen la complejidad del sistema al abstraerlo de detalles innecesarios. El nivel de complejidad que las plataformas han alcanzado actualmente, unido a la necesidad de adaptar el software a requerimientos en constante cambio, han creado la necesidad de simplificar aún más el desarrollo. Por lo que era necesario aumentar aún más el nivel de abstracción y ocultar aún más los detalles de implementación, para así poder reducir la complejidad que el ingeniero de software debe enfrentar a la hora de crear un sistema. Es por ello que nacen los lenguajes de modelado: notaciones en su mayoría visuales, que intentan representar un sistema de software a un nivel mucho más alto que los lenguajes de programación, representándolo en forma más intuitiva para personas sin especialización en Informática. (2)

De dichos esfuerzos nace el UML. Este lenguaje, es una notación visual que representa elementos como: los requerimientos del sistema, estructura general, comportamiento del sistema e implantación (2). Las raíces del UML provienen de tres métodos distintos. El método de Grady Booch, la Técnica de Modelado de Objetos de James Rumbaugh y “Objetory”, de Ivar Jacobson. Conocidas estas tres

personalidades como “los tres amigos”. En 1994 Booch, Rumbaugh y Jacobson dieron forma a la versión 1.0 de UML y en 1997 fue aceptado por la OMG, este grupo gestiona estándares relacionados con la tecnología orientada a objetos (metodologías, bases de datos objetuales, CORBA (Common Object Request Broker Architecture, lo que en español significa Arquitectura Común de Intermediarios en Peticiones a Objetos), entre otros). Fue también en esta fecha, que fue lanzada la versión 1.1 del UML. Desde entonces, este lenguaje atravesó varias revisiones y refinamientos hasta llegar a la versión: UML 2.0. (3)

1.2. Lenguaje Unificado de Modelado.

Para comprender qué es el UML, basta con analizar cada una de las palabras que lo componen, por separado.

* Lenguaje: el UML es precisamente un lenguaje. Lo que implica que este cuenta con una sintaxis y una semántica. Por lo tanto, al modelar un concepto en UML, existen reglas sobre cómo deben agruparse los elementos del lenguaje y el significado de esta agrupación.

* Modelado: el UML es visual. Mediante su sintaxis se modelan distintos aspectos del mundo real que permiten una mejor interpretación y entendimiento de este.

* Unificado: unifica varias técnicas de modelado en una única. (4)

Ya que el UML proviene de técnicas orientadas a objetos, se crea con la fuerte intención de que este permita un correcto modelado orientado a objetos. El lenguaje UML tiene una notación gráfica muy expresiva que permite representar en mayor o menor medida todas las fases de un proyecto informático: desde el análisis y el diseño, hasta la implementación y configuración. Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones en: visualizar, especificar, construir y documentar. (5)

Aunque UML es bastante independiente del proceso de desarrollo que se siga, los mismos creadores de UML han propuesto su propia metodología de desarrollo, denominada el Proceso Unificado de Desarrollo (RUP). El Proceso Unificado de Desarrollo, está basado en componentes, lo cual quiere decir que el sistema de software en construcción está formado por componentes de software interconectados a través de interfaces bien definidos. Además, RUP utiliza el UML para expresar gráficamente todos los esquemas de un sistema de software. Pero realmente, los aspectos que

definen este Proceso Unificado son tres: es iterativo e incremental, dirigido por casos de uso y centrado en la arquitectura. (5)

El lenguaje UML, es el lenguaje de modelado de software más usado en el mundo. Las causas elementales que lo determinaron, son primeramente que la OMG la aceptó como estándar, además, actualmente casi todas las herramientas CASE (Computer Aided Software Engineering, lo que en español significa Ingeniería de Software Asistida por Computadora) y de desarrollo de software han adoptado al UML, como lenguaje de modelado. En su creación y desarrollo participaron estudiosos e investigadores de gran prestigio y fue apoyado por la mayoría de las instituciones más importantes del mundo de la Informática.

1.2.1. UML 2.0

Los perfiles UML se definieron originalmente en la versión 1.0 de UML, aunque era difícil saber si se estaban aplicando correctamente debido a que su definición era un tanto ambigua. La nueva versión UML 2.0 mejora substancialmente su definición, especificándose de forma más clara las relaciones permitidas entre los elementos del modelo a especificar y el uso de las metaclases de un metamodelo dentro de un perfil UML. Esta nueva versión, ha sido probada en la definición de otros perfiles a nivel mundial. Al momento de desarrollar el nuevo estándar 2.0 de UML, la OMG se planteó, entre otros, dos objetivos que se podrían considerar principales, debido a la influencia de estos en la nueva versión del estándar: hacer el lenguaje de modelado más extensible y permitir la validación y ejecución de modelos. UML 2.0 se desarrolla sobre la base de estos dos objetivos, causando un quiebre respecto a versiones anteriores. (6)

En las versiones previas de UML, se hacía un fuerte hincapié en que este no era un lenguaje de programación, un modelo creado mediante UML no podía ejecutarse. En el UML 2.0, esta asunción cambió de manera drástica y se modificó el lenguaje, de manera tal que permitiera capturar mucho más comportamiento. De esta forma, se permitió la creación de herramientas que soporten la automatización y generación de código ejecutable, a partir de modelos UML. (6)

1.2.2. Reestructuración del lenguaje UML 2.0.

Para lograr los objetivos enunciados, varios aspectos del lenguaje fueron modificados. La especificación se separó en cuatro especificaciones (paquetes) bien definidas. Es interesante destacar que el UML 2.0 puede definirse a sí mismo. Es decir, su estructura y organización es modelable

utilizando el propio UML 2.0; de esta manera, se da un ejemplo de utilización del UML en un dominio distinto al del desarrollo de software. En este caso, cada paquete del diagrama representa cada una de las cuatro especificaciones que componen el lenguaje. (7)

1.2.3. Estándares que conforman UML 2.0.

Los estándares que conforman UML 2.0 son:

- ✓ Superestructura: La superestructura de UML es la definición formal de los elementos que lo componen. Esta definición es utilizada por los desarrolladores de aplicación. Es aquella sobre la que la mayoría conoce de versiones anteriores del lenguaje.

- ✓ Infraestructura: Son los conceptos de bajo nivel. En este estándar los metamodelos dan soporte a la superestructura. En la infraestructura de UML se definen los conceptos centrales y de más bajo nivel. La infraestructura es un metamodelo, mediante el que se modela el resto de UML. No es empleada por usuarios finales de UML, pero ofrece la piedra fundamental sobre la cual se define la superestructura, que es la utilizada por la mayoría de los usuarios. La infraestructura brinda, también, varios mecanismos de extensión que hacen de UML un lenguaje configurable. Para los usuarios normales de UML, basta con saber que la infraestructura existe y con conocer cuáles son sus objetivos.

- ✓ OCL (Object Constraint Language): en español Lenguaje de Restricción de Objetos, es de gran utilidad para especificar conceptos ambiguos sobre los distintos elementos del diagrama. Define un lenguaje simple para escribir restricciones y expresiones sobre elementos de un modelo. Suele ser útil cuando se está especificando un dominio particular mediante UML y es necesario restringir los valores permitidos para los objetos del dominio. OCL brinda la posibilidad de definir invariantes, precondiciones, poscondiciones y restricciones en los elementos de un diagrama. Fue incorporado a UML en la versión 1.1 y es un ejemplo más de las muchas herramientas agregadas a UML.

- ✓ XMI (XML Metadata Interchange): Es una especificación para el intercambio de diagramas y permite compartir diagramas entre diferentes herramientas de modelado UML. La especificación para el intercambio de diagramas fue escrita para poder compartir modelos realizados mediante UML entre diferentes herramientas de modelado. (7)

1.3. Metamodelos y perfiles UML.

A pesar de todas sus formidables características, UML tiene una insuficiencia y es que no se ajusta con facilidad a determinados dominios que se caracterizan por un nivel más alto de formalidad, sin embargo esta insuficiencia, se contrasta con sus propios mecanismos de extensión, a partir de estos se puede concebir el modelado con un nivel más extenso de detalle en lo que se refiere a semántica y sintáctica.

Se especifican lenguajes de modelado porque el dominio no se encuentra totalmente cubierto, además personalizar lenguajes de modelado permite la reutilización de los lenguajes existentes, editor, generación de código o ingeniería inversa, transformaciones, servicios, estructura y definición de la semántica.

Los perfiles UML y los metamodelos son dos mecanismos para personalizar lenguajes de modelado. El mecanismo de perfiles, no es un mecanismo de extensión de primera clase (es decir, no permite modificar los metamodelos existentes), por el contrario, la intención de los perfiles es dar un mecanismo sencillo para la adaptación de un metamodelo existente con construcciones que son específicas de un dominio particular, plataforma o método. Cada adaptación se agrupa en un perfil. No es posible quitar ninguna de las restricciones que se aplican a un metamodelo como UML mediante un perfil, pero es posible añadir nuevas limitaciones que son específicas para el mismo. Las otras restricciones son inherentes al mecanismo de perfiles, que tienen por objeto, limitar la forma en que se ha personalizado un metamodelo.

Los perfiles UML han sido definidos específicamente para proporcionar un mecanismo de extensión de ligera para UML. En UML 1.1, los estereotipos y valores etiquetados se utilizan como extensiones basadas en cadenas que se podría unir a los elementos del modelo UML de una manera flexible. En las revisiones posteriores de UML, la noción de un perfil fue definida con el fin de proporcionar más estructura y precisión a la definición de los valores y estereotipos etiquetados. La infraestructura y las especificaciones de la superestructura de UML 2.0 lo han definido como una técnica específica de metamodelado. Los estereotipos son metaclases especificadas, los valores etiquetados son metaatributos estándar, y los perfiles son determinados tipos de paquetes. Los siguientes requisitos se han llevado a la definición de la semántica de un perfil:

- ✓ Un perfil debe proporcionar mecanismos para la especialización de un metamodelo de referencia (como un conjunto de paquetes de UML), de tal manera que la semántica

especializada no esté en contradicción con la semántica del metamodelo de referencia. Es decir, las limitaciones de perfil típicamente pueden definir reglas de buena formación que son más restrictivos a los especificados por el metamodelo de referencia.

- ✓ Debe ser posible intercambiar los perfiles entre los instrumentos, junto con los modelos a los que se han aplicado, utilizando los mecanismos de intercambio de XMI. Un perfil debe definirse como un modelo UML intercambiable.

- ✓ Un perfil debe ser capaz de hacer referencia a las bibliotecas UML específicas del dominio, donde ciertos elementos del modelo se predefinen.

- ✓ Debe ser posible especificar qué perfiles se aplican a un determinado paquete (o cualquier especialización de ese concepto). Esto es particularmente útil en el intercambio de modelos para que la importación de un entorno pueda interpretar un modelo correctamente.

- ✓ Debe ser posible definir una extensión de UML que combine perfiles y librerías de modelos (incluyendo bibliotecas de plantillas) en una única unidad lógica. Esto se hace para que dentro de dicha unidad exista mayor claridad de definición y mayor facilidad de intercambio.

- ✓ Un perfil debe ser capaz de especializar la semántica de los elementos estándar del metamodelo UML. Por ejemplo, en un modelo con el perfil "modelo de Java," la generalización de las clases debe ser capaz de limitarse a la herencia simple sin tener que asignar explícitamente un estereotipo «clase de Java» a todos y cada instancia de la clase.

- ✓ Una convención de notación para las definiciones de estereotipos gráficos como parte de un perfil, debe ser proporcionado.

- ✓ Con el fin de satisfacer el primer requisito, perfiles UML debe formar un mecanismo de extensión de metamodelo que imponga determinadas restricciones a la forma en que el metamodelo UML puede ser modificado. El metamodelo de referencia se extiende sin cambios en los perfiles. Por lo tanto, se prohíbe insertar nuevas metaclases en la jerarquía de metaclases de UML o modificar el estándar UML. Estas restricciones no se aplican en el contexto de MOF (Meta-Object Facility) donde, en principio, puede ser cualquier metamodelo reelaborado en cualquier dirección.

- ✓ La gran mayoría de las herramientas CASE que usan UML deberían ser capaces de aplicar perfiles. El diseño de los perfiles UML no debe limitar estas herramientas para tener una implementación interna basada en una arquitectura metametamodelo/metamodelo.

- ✓ Los perfiles se pueden aplicar de forma dinámica o retractada de un modelo. Es posible en un modelo existente, aplicar nuevos perfiles y cambiar el conjunto de perfiles de aplicación.

- ✓ Los perfiles pueden ser combinados de forma dinámica. Con frecuencia, varios perfiles se aplicarán al mismo tiempo en el mismo modelo. Esta combinación de perfil no se pronostica en el momento de definición del perfil.

- ✓ Los modelos pueden ser cambiados sin importar los perfiles conocidos por el destino de llegada. El destino del intercambio de un modelo extendido por un perfil, no está obligado a interpretar una descripción del perfil específico. El entorno de destino interpreta extensiones sólo si este cuenta con los perfiles requeridos.

La especificación de la infraestructura de UML se utiliza en varios metaniveles de las diversas especificaciones que la OMG se ocupa de estandarizar. Por ejemplo, la superestructura de UML la utiliza para modelar el modelo UML, mientras que MOF la utiliza para proporcionar capacidad a metamodelos.

Un metamodelado se basa en la idea de reedificar las entidades que forman cierto tipo de modelo y describir las propiedades comunes del mismo, en forma de un modelo de objetos, cuando se ve la clase como un objeto, la clase es una instancia de otra clase. La idea fundamental en el metamodelado es que las entidades del modelo (clases) juegan dos papeles: el papel de plantilla (cuando se ve como clase) y el papel de instancia (cuando se ve como objeto). El metamodelado además, define la estructura, semántica y restricciones para una familia de modelos.

La OMG define una arquitectura basada en cuatro niveles de abstracción que van a permitir distinguir entre los distintos niveles conceptuales que intervienen en el modelado de un sistema. A estos niveles se les denomina comúnmente con las iniciales M0, M1, M2 y M3. (6)

El nivel M0 – Las instancias. El nivel M0 modela el sistema real, y sus elementos son las instancias que componen dicho sistema.

El nivel M1 – El modelo del sistema. Los elementos del nivel M1 son los modelos de los sistemas concretos.

El nivel M2 – El modelo del modelo (el metamodelo). Los elementos del nivel M2 son los lenguajes de modelado. El nivel M2 define los elementos que intervienen a la hora de definir un modelo del nivel M1.

El nivel M3 – El modelo de M2 (el meta-metamodelo). Finalmente, el nivel M3 define los elementos que constituyen los distintos lenguajes de modelado. De esta forma, el concepto de "clase" definido en UML (que pertenece al nivel M2) puede verse como una instancia del correspondiente elemento del nivel M3, en donde se define de forma precisa ese concepto, así como sus características y las relaciones con otros elementos. (6)

La OMG definió un lenguaje para describir los elementos del nivel M3, este lenguaje es MOF que se utiliza para crear metamodelos. UML está definido como un metamodelo MOF y el mismo es aplicable a cualquier dominio, además es un medio universal para definir lenguajes de modelado.

MOF permite expresar metadatos y es independiente de la plataforma, también es usado para definir y manipular un conjunto de metamodelos interoperables y se considera como la evolución de los repositorios orientados a objetos e investigación en metamodelos. MOF está descrito con la notación UML, OCL y texto informal. La semántica de UML viene descrita por su metamodelo, que va a venir expresado en MOF. Si se quisiera definir un lenguaje diferente a UML, también se expresaría en MOF. (8)

La extensibilidad de primera clase se maneja a través de los metamodelos, particularmente MOF, donde no hay restricciones en lo que se puede hacer con un metamodelo: se puede agregar y quitar metaclases y relaciones que se consideren necesarias. Por supuesto, entonces es posible imponer restricciones que no permitan modificar metamodelos existentes, sólo extenderlos. En este caso, los mecanismos de extensibilidad de primera clase y los perfiles UML se unen. La OMG para UML 2.0, señala varias razones por las que un diseñador puede querer extender y adaptar un metamodelo existente, sea el del propio UML o incluso el de otro perfil:

- ✓ Disponer de una terminología y vocabulario propio de un dominio de aplicación o de una plataforma de implementación concreta.
- ✓ Definir una sintaxis para construcciones que no cuenten con una notación propia.

- ✓ Definir una nueva notación para símbolos ya existentes, más acorde con el dominio de la aplicación objetivo (por ejemplo, una figura con un ordenador en lugar del símbolo para representar un nodo que por defecto ofrece UML para representar ordenadores en una red).
- ✓ Añadir cierta semántica que no aparece determinada de forma precisa en el metamodelo (por ejemplo, la incorporación de prioridades en la recepción de señales en una máquina de estados de UML).
- ✓ Añadir cierta semántica que no existe en el metamodelo (por ejemplo relojes y tiempo continuo).
- ✓ Añadir restricciones a las existentes en el metamodelo, restringiendo su forma de utilización (por ejemplo, impidiendo que ciertas acciones se ejecuten en paralelo dentro de una transición, o forzando la existencia de ciertas asociaciones entre las clases de un modelo).
- ✓ Añadir información que puede ser útil a la hora de transformar el modelo a otros modelos, o a código. (6)

A partir de la investigación realizada en cuanto a estos dos mecanismos de extensión (perfiles UML y metamodelos), se arribó a la conclusión de que cualquiera de estos dos mecanismos pudieran ser utilizados. A pesar de que definir un nuevo lenguaje, descrito por MOF, permite mayor grado de expresividad y correspondencia con los conceptos del dominio de una aplicación en particular, se decidió extender el lenguaje utilizando perfiles UML. A continuación se argumenta el por qué de dicha elección:

Para proporcionarle solución al problema de esta investigación no es necesario modificar la semántica de UML, sino sólo particularizar algunos de sus conceptos, por lo que definir un nuevo lenguaje utilizando MOF, alargaría el proceso, extendiendo el tiempo de realización de este trabajo al redefinir metaclases, metaatributos y estereotipos, que ya define por defecto el lenguaje UML, ya que cuando un lenguaje se describe con MOF, no respeta el metamodelo estándar de UML y esto dificulta que las herramientas UML existentes puedan manejar sus conceptos de una forma natural. Además, el paquete UMLProfiles 2.0 define una serie de mecanismos para extender y adaptar las metaclases de un metamodelo cualquiera a las necesidades concretas de un dominio de aplicación.

1.4. Ejemplos de perfiles aplicados.

En las indagaciones realizadas fueron encontradas un grupo de extensiones UML para disímiles dominios que formalizan elementos tanto para análisis, para arquitectura, como para implementación. Como por ejemplo un perfil UML para software orientado a aspecto, un perfil UML para representar XML Schemas y un perfil UML para modelar software de servicios. También se han creado otros perfiles que fueron estandarizados por la OMG como por ejemplo el perfil UML para EDOC (Enterprise Distributed Integration) y el perfil UML para planificación, prestaciones y tiempo. En la bibliografía consultada se evidenció una gran variedad de perfiles, pero no se encontró ninguna idea que aborde el uso de perfiles UML que permitan formalizar elementos del análisis, arquitectura y desarrollo, que pudieran aplicarse a la línea Soluciones Integrales.

A continuación se muestran 3 ejemplos de perfiles UML, que simbolizan el análisis realizado al resto de los antes mencionados, estos muestran incluso la especificación de los mismos en los anexos del trabajo; a partir de ellos se emitirá una valoración, en la que se describen las causas del por qué estos no pueden ser aplicados al perfil UML perteneciente a esta investigación. Los mismos responden a la problemática de cada ejemplo, en los que a partir de los mecanismos de extensión propios de UML, brindan una solución rápida y eficiente.

Ejemplo No1.

Tema: Perfil UML para el Aspecto de Notificación en Entornos Distribuidos CORBA.

Resumen: El servicio de notificación de CORBA permite la comunicación asíncrona entre objetos desarrollados bajo esta plataforma. Sin embargo, la utilización de este servicio a nivel de implementación conlleva a un fuerte acoplamiento entre el servicio y los objetos que lo usan, debido a la mezcla de código que se produce al tratar dos aspectos diferentes en el mismo objeto: comunicación asíncrona de eventos y funcionalidad. Teniendo en cuenta que el uso de técnicas orientadas a aspecto permite evitar situaciones de mezcla y código desordenado, y con el objetivo de tratar la comunicación asíncrona de eventos en sistemas distribuidos bajo CORBA de manera uniforme, se propone la definición de un perfil UML para el modelado del aspecto de notificación. Este perfil permite, además, tener perfectamente separados desde la fase de modelado, los detalles relativos a la notificación de eventos de la funcionalidad propia de los componentes desarrollados bajo la plataforma CORBA. Finalmente, el uso de este perfil facilita la generación de código referente al aspecto de notificación y su posterior integración con los objetos participantes en la comunicación. (9)

Desarrollo: Para la definición de este perfil UML y para tratar el aspecto de notificación en entornos distribuidos CORBA, se ha optado por la definición de varios estereotipos: <<Supplier>>, que representa un productor de eventos; <<Consumer>>, que representa un consumidor de eventos; <<Channel>>, que modela el canal de eventos que se utilice. Estos estereotipos tienen como elemento base el elemento <<Class>> del metamodelo UML. Por tanto, cuando se utiliza un elemento de tipo <<Supplier>> en los modelos UML, se está asegurando que ese elemento sigue las mismas reglas semánticas que un elemento de tipo <<Class>>, ya que se ha extendido el metamodelo, pero en ningún caso se ha modificado su semántica. Además, el elemento <<Supplier>> representa el aspecto de producción de eventos, por lo que al relacionarlo con una clase, se está convirtiendo esa clase en un elemento productor de eventos. Para lograr esta transformación, se han definido los estereotipos <<Supplier_crosscut>> y <<Consumer_crosscut>> que serán aplicados sobre elementos de tipo <<Association>> del metamodelo. (9) (Ver anexo No. 1)

Ejemplo No2.

Tema: Perfil de UML para la definición de componentes inteligentes.

Resumen: El desarrollo de sistemas basados en componentes tiene el problema del posible cambio en las interfaces de los mismos al adquirir o usar una nueva versión de un componente o sustituirlo por otro nuevo, ya que dichos componentes se comportan como cajas negras que pueden ser remotos o locales (adquiridos por compra, descarga, entre otros), por lo que se presenta, cómo usar los mecanismos de extensión del lenguaje de modelado UML y en concreto el uso de perfiles UML para definir un metamodelo de componentes inteligentes cuya generación de código produzca unos componentes con una base de conocimiento capaz de adaptarse a los cambios de interfaz de los componentes que usa. (10)

Desarrollo: El perfil creado define un estereotipo, correspondiente a la clase <<Proxy>> definida en el metamodelo, así como las metaclases de UML sobre las que pueden aplicarse tales estereotipos. Dichas metaclases corresponden al metamodelo a ser extendido, en este caso al metamodelo de UML. (10) (Ver anexo No. 2)

Ejemplo No3.

Tema: Perfil de UML 2.0 para servicios de software.

Resumen: Este perfil de UML 2.0 permite la modelación de los servicios, SOA (Service Oriented Architecture) y soluciones orientadas a servicios. El objetivo del perfil es proporcionar un lenguaje común para describir servicios que cubra un gran número de actividades a través del ciclo vital de desarrollo y también proporcione vistas a los distintos interesados. Así, por ejemplo, el perfil proporciona funciones para que el arquitecto correlacione los servicios pronto en el ciclo vital usando particiones lógicas para describir toda la cartera de servicios de la empresa. Esta vista es detallada posteriormente por los diseñadores que desarrollan especificaciones de servicio, tanto estructurales como de comportamiento, que actúan como los contrastes entre los clientes de servicio y los implementadores. La vista de mensaje ofrece la posibilidad de que los diseñadores reutilicen los modelos de información para definiciones de datos de servicios comunes. El perfil utiliza correctamente modelos de desarrollo de escenarios de cliente complejos y también educa a las personas en los problemas relevantes del desarrollo de soluciones orientadas a servicios. (11)

Desarrollo: En este perfil de UML 2.0, se muestran los detalles reales del perfil con cada estereotipo, sus metaclases usando la notación de extensión. También se agregan algunas restricciones de este modelo, en especial aquellas correstricciones entre elementos de perfil. (11) (Ver anexo No. 3)

A partir del análisis de los ejemplos anteriormente mostrados se puede concluir que ninguno de los tres resuelve el problema de esta investigación, debido a que los dificultades que solucionan cada uno por separado, constituyen una buena solución pero para cumplir el propósito que conllevó a la realización de ese perfil, no para la de este trabajo ya que no se ajusta a la necesidad que tiene la línea Soluciones Integrales, al no formalizar elementos durante la etapa de análisis y diseño que permitan la traducción precisa en implementación.

1.5. Herramientas que soportan UML 2.0.

Una herramienta UML o una herramienta de modelado UML es una aplicación de software compatible con algunas o todas las de la notación y la semántica UML asociada. Las herramientas UML admiten los siguientes tipos de funcionalidad: diagramación, ingeniería de ida y vuelta, generación de código, ingeniería inversa, modelo e intercambio de diagramas y modelo de transformación.

Algunos de los requisitos utilizados a la hora de elegir una herramienta de UML son los siguientes: que posea código abierto, tipo de licencia, lenguaje de programación utilizado, integración con entornos de desarrollo, coste, versión de UML, diagramas que soporta, soporte para MDA, soporte para XMI,

generación de código (y qué lenguajes de programación soporta), capacidades de ingeniería inversa, sistema operativo y requisitos de instalación.

1.5.1. Ejemplo de herramientas.

A continuación, se nombran algunas herramientas que soportan UML 2.0: Enterprise Architect (EA), Altova UModel 2005, Together, Rational Software Architect, Embarcadero Describe, Visual Paradigm (VP), entre otras. Es importante destacar que hasta la fecha el más alto nivel de conformidad lo tiene Enterprise Architect.

Enterprise Architect es una herramienta flexible, completa y potente de modelado para plataformas Windows. Se trata de una herramienta CASE orientada a objetos que proporciona una ventaja competitiva para el desarrollo de sistemas, gestión de proyectos y análisis de negocio. Enterprise Architect posee un ciclo de vida integral, análisis de UML y herramientas de diseño, que cubre el desarrollo de software que abarca desde la recogida de requisitos, a través de las fases de análisis, modelos de diseño, el mantenimiento de pruebas y finalmente, la reutilización. EA ha demostrado ser muy notoria, con una comunidad fuerte de usuarios y un crecimiento continuo.

EA se utiliza además, para el desarrollo de diversos tipos de sistemas de software para una amplia gama de industrias, incluyendo: servicios bancarios, desarrollo web, ingeniería, finanzas, medicina, investigación, académicos, transporte, comercio minorista, servicios públicos e ingeniería eléctrica. También se utiliza con eficacia para fines comerciales y en la arquitectura, conjuntamente de que es muy usado en la formación de profesionales de muchas empresas y universidades importantes.

Con esta herramienta sin dudas sería muy ventajoso contar, haciendo uso de todas sus formidables características ya que se considera que es la más indicada para el desarrollo del trabajo, pero por políticas de la universidad, el perfil se desarrollará en Visual Paradigm.

1.5.2. Visual Paradigm.

Es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y documentación. Visual Paradigm es una herramienta multiplataforma de modelado visual UML y una herramienta CASE muy potente y fácil de utilizar. VP-UML aporta a los desarrolladores de

software una plataforma de desarrollo, puntera para construir aplicaciones de mejor calidad, más baratas y con mayor rapidez. Aporta una excelente interoperabilidad con otras herramientas CASE y muchos de los entornos IDE líderes del mercado. (12)

Además VP proporciona: diseño de software con UML, captura de requisitos con diagrama de requisitos SysML, bases de datos de diseño con el Diagrama Entidad Relación (ERD), entorno unificado de diseño de software para analista de sistemas y desarrollador de software para analizar, diseñar y mantener aplicaciones de software en una disciplina. (12)

1.6. Arquitectura Dirigida por Modelos y Desarrollo Dirigido por Modelos.

El papel de los modelos se ha vuelto mucho más importante en los últimos años y los beneficios potenciales usándolos son significativamente mayores en el software que en cualquier otra disciplina de la ingeniería. Teniendo en cuenta lo antes planteado para el desarrollo de este trabajo es necesario encontrar la manera más rápida de modelar agregando combustible a la generación de código. Para ello se hará uso de dos estándares definidos por la OMG que dirigen la arquitectura y el desarrollo a través de los modelos, lo cual constituye una aproximación a la arquitectura y desarrollo de software en donde los enfoques fundamentales son los modelos y no los programas, utilizando los lenguajes de modelado como herramientas de la implementación, esto implica la generación automática de programas a partir de los modelos.

1.6.1. MDA.

La Object Management Group fue el creador de MDA. Existe un grupo de beneficios mediante el uso del enfoque de MDA (por ejemplo, más rápido y más barata la producción de aplicaciones, optimización del código fuente y el uso automático de patrones de diseño). Ciertamente hay también problemas como por ejemplo no hay una explicación de cómo las entidades en la infraestructura se relacionan con el mundo real, ya que esto es posible mediante el desarrollo de aplicaciones a mano.

El enfoque de MDA en general separa la funcionalidad del sistema de la implementación de detalles. Es un marco para el modelo impulsado por desarrollo de software definido por OMG. MDA ofrece también la plataforma de interoperabilidad, portabilidad, independiente de la plataforma y la productividad. Arquitectura Dirigida por Modelos, defiende la definición de modelos como elementos de primera clase para el diseño e implementación del sistema. En MDA, las actividades más importantes

pasan a ser las de modelado de los diferentes aspectos de un sistema y la definición de transformaciones de un modelo a otro de forma que puedan ser automatizadas.

MDA define tres niveles conceptuales. En primer lugar, los requisitos del sistema se modelan en un modelo independiente de la computación (CIM, Computation Independent Model) que define el sistema en el entorno en el cual va a operar. En el siguiente nivel se encuentra el PIM (Platform Independent Model), un modelo que describe la funcionalidad del sistema, pero omitiendo detalles sobre cómo y dónde va a ser implementado (por ejemplo, el PIM puede ser independiente de la distribución y de la plataforma de objetos en donde se ejecutará, ya sea CORBA, J2EE y .NET). El objetivo es entonces transformar el PIM en un modelo dependiente de la plataforma final, denominado PSM (Platform Specific Model). De esta forma se consigue una gran independencia entre la descripción de la funcionalidad, y los detalles de implementación propios de la plataforma objetivo. (6)

Es precisamente a la hora de describir el modelo de la plataforma y las reglas de transformación entre modelos cuando los perfiles UML pueden desempeñar un papel muy importante. Si se emplean perfiles UML para especificar el modelo de una plataforma, eso garantiza que los modelos derivados serán modelos consistentes con UML. De hecho, la principal regla para aplicar MDA con éxito es usar en lo posible estándares (modelos estándares y perfiles UML de lenguajes o plataformas estándares). La Arquitectura Dirigida por Modelos, es una gran forma de gestionar la complejidad, alcanzar altos niveles de reúso y lograr una significativa reducción de los esfuerzos requeridos en los proyectos de desarrollo de software. (6)

Además de aumentar la productividad y la calidad, MDA ofrece otras ventajas respecto a otros menos avanzados enfoques MDD, incluyendo: modelos independientes de la plataforma, la promoción del sistema, la longevidad y la flexibilidad en el despliegue, modelos ejecutables, además es construida con una plataforma que usa lenguaje de acción independiente al incremento del nivel del modelo de la abstracción, la calidad y productividad de los desarrolladores.

1.6.2. MDD.

El Desarrollo Dirigido por Modelos constituye una serie de enfoques de desarrollo de software que se basan en el uso de software de modelado como una forma principal de expresión. A veces los modelos se construyen para un determinado nivel de detalle y otras veces se construyen modelos completos incluyendo las acciones ejecutables. El código puede ser generado de los modelos, que van desde esqueletos de sistema para completar, hasta los productos de despliegue.

Con la introducción del lenguaje UML, MDD se ha vuelto muy notorio hoy en día con un amplio cuerpo de profesionales y herramientas de apoyo. Es posible concentrar el trabajo en el desarrollo de los modelos y llegar mucho más cerca del dominio del problema por definición teniendo en cuenta la aplicación subyacente. Esto es posible porque la necesidad de técnicas de automatización han madurado y han surgido estándares industriales. Sin embargo también hay riesgos de terminación manual de código. Si el código se integra con la mano más adelante en el proceso de desarrollo, puede suceder que este cause algunos daños a la aplicación. El objetivo es permitir que la herramienta genere todo el código.

El nivel de abstracción del enfoque MDD es bastante alto y no todo el mundo es capaz de llevar a buen crear un buen modelo en este nivel de abstracción. Los modelos tienen que ser credos por personas cualificadas.

La motivación subyacente para MDD es mejorar la productividad. Hay dos formas básicas para ofrecer este beneficio: **a corto plazo**: al aumentar el valor de un artefacto de software primario en términos de la funcionalidad que ofrece; a **largo plazo**: al reducir la velocidad a la que un artefacto de software primario viene obsoleto.

Conclusiones parciales.

Con el estudio de los principales elementos teóricos, se determinó que los perfiles UML, constituyen el mecanismo que soluciona la problemática de esta investigación, pues define una serie de recursos para extender y adaptar las metaclases de un metamodelo cualquiera a las necesidades concretas de un dominio de aplicación. Este mecanismo no pretende modificar la semántica de UML, sino sólo particularizar algunos de sus conceptos, además pueden integrarse conceptos de la teoría y metodología de la línea Soluciones Integrales con los mecanismos de extensión que ofrece UML 2.0. También se observó que el modelado de aplicaciones tiene cada vez mayor protagonismo frente a la codificación, y el papel de los perfiles UML es fundamental en la definición de modelos, por lo que el perfil se acopla con facilidad a MDA y MDD, lo que ofrece la posibilidad de generar de manera automática la traducción precisa en implementación, a partir del modelo conceptual.

Capítulo 2: Descripción del perfil UML.

Introducción.

En este capítulo se explica detalladamente cuáles son los pasos fundamentales a la hora de definir un perfil UML, a partir de estos pasos se precisa el mismo para el Modelo del Análisis, Diseño e Implementación para la Línea de Soluciones Integrales, DATEC. En este perfil se sistematizarán las extensiones a utilizar a la hora de confeccionar los modelos en el expediente digital. Su uso en DATEC reportará beneficios al formalizar elementos del modelado durante la etapa de análisis y diseño permitiendo su traducción precisa en implementación. El objetivo de este perfil es servir de directriz a la modelación para las vistas lógica y de implementación aumentando la expresividad del UML convencional.

2.1. Elementos que conforman un perfil UML.

La OMG definió un estándar para extender un metamodelo existente mediante perfiles UML. Los elementos que conforman este estándar son los siguientes:

Class. A partir de las clases se indica la forma en que puede extenderse por uno o varios estereotipos. Los estereotipos son el único tipo de definiciones de metaclase que no se pueden extender por ellos mismos. Una clase que se extiende por un estereotipo puede ser ampliada por el estereotipo «metaclase». A continuación se muestra un ejemplo donde se hace explícito que la clase extendida Interfaz es en realidad una metaclase. (13)

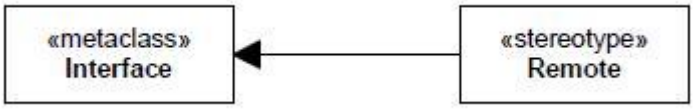


Figura No1. Ejemplo de una extensión.

Extensión. Una extensión se usa para indicar que las propiedades de una metaclase se extienden a través de un estereotipo, y da la posibilidad de añadir flexibilidad a los estereotipos de las clases. La extensión es una especie de asociación. Un extremo de la extensión es una propiedad común y el otro extremo es un ExtensionEnd, esta última se utiliza para atar a un estereotipo al extender una definición de metaclase. La notación para una extensión es una flecha que apunta desde el estereotipo de la clase extendida, donde se muestra la punta de flecha en un triángulo lleno. Una extensión puede tener

los mismos elementos que una asociación común, pero las flechas de navegación nunca se muestran. Un ejemplo común lo constituye el ejemplo mostrado en la definición de Class.

Profile. Un perfil define las extensiones limitadas a un metamodelo de referencia con el fin de adaptar el metamodelo a una determinado plataforma o dominio. Un perfil utiliza la misma notación que un paquete, con la particularidad de que la palabra clave «profile» se muestra sobre el nombre del paquete. Un metamodelo de referencia consiste típicamente de metaclases que son importados o de propiedad local. Todas las metaclases que se extienden por un perfil tienen que ser miembros del metamodelo de referencia. Las metaclases de referencia importan elementos y los metamodelos de referencia importan paquetes. Las reglas siguientes son utilizadas para determinar si un elemento del modelo se ve o está oculto cuando el perfil se ha aplicado. Los elementos del modelo son visibles si:

- ✓ Hace referencia explícita a metaclases.
- ✓ Está contenido en un paquete que hace referencia explícita al metamodelo de referencia.
- ✓ Están extendidos por un estereotipo del perfil de aplicación.

En el ejemplo del anexo No.4 (Ver anexo No.4) se muestra el metamodelo MyMetamodel que contiene dos metaclases: Metaclass1 y Metaclass2. El perfil UML MyProfile hace referencia a MyMetamodel y Metaclass2. Sin embargo, también hay una referencia explícita de la metaclase Metaclass2, que reemplaza la del modelo de referencia. Una aplicación de MyProfile para algún modelo basado en MyMetamodel mostrará las instancias de Metaclass2 (debido a que hace una referencia explícita a una metaclase). Además, las instancias de Metaclass1 que se extienden por una instancia de MyStereotype serán visibles. Sin embargo, los casos de Metaclass1 que no se extienden por MyStereotype permanecen ocultos. (13)

ProfileApplication. Un perfil de aplicación se utiliza para mostrar los perfiles que se han aplicado a un paquete. Los nombres de los perfiles se muestran con una flecha discontinua con una punta de flecha abierta desde el paquete al perfil aplicado. La palabra «apply» se muestra cerca de la flecha. El ejemplo del anexo No.5 (Ver anexo No.5), muestra los perfiles Java y EJB, que son aplicados al paquete WebShopping. (13)

Package. Un paquete puede tener uno o más ProfileApplications para indicar cuales son los perfiles que se han aplicado. Debido a que un perfil es un paquete, es posible aplicar un perfil no sólo a los paquetes, sino también a los perfiles.

Estereotipo. Un estereotipo define una metaclase existente que puede ser extendida, y permite el uso de la plataforma o dominio de la terminología específica o anotación además de lo definido por la metaclase extendida. Los estereotipos vienen definidos por un nombre, y por una serie de elementos del metamodelo sobre los que puede asociarse.

2.2. Pasos para describir un perfil.

A partir de los estándares definidos por la OMG, la cual define los elementos que conforman un perfil UML, a continuación se muestra como se enlazan estos elementos de modo tal que permita la realización del mismo:

Un perfil UML se define en un paquete UML, estereotipado «profile», que extiende a un metamodelo o a otro perfil. Existen tres mecanismos que se utilizan para definir perfiles: estereotipos (<<stereotypes>>), restricciones (<<constraints>>), y valores etiquetados (<<tagged values>>). Los estereotipos vienen definidos por un nombre, y por una serie de elementos del metamodelo sobre los que puede asociarse. Gráficamente, los estereotipos se definen dentro de cajas, estereotipadas «stereotype». Las restricciones («constraints») imponen condiciones sobre los elementos del metamodelo que han sido estereotipados. De esta forma pueden describirse, entre otras, las condiciones que ha de verificar un modelo bien formado de un sistema en un dominio de aplicación. Los valores etiquetados son metaatributos adicionales que se asocia a una metaclase del metamodelo extendido por un perfil. Todo valor etiquetado ha de contar con un nombre y un tipo, y se asocia a un determinado estereotipo, estos se representan de forma gráfica como atributos de la clase que define el estereotipo. (6)

A continuación se describe un método de elaboración de perfiles UML. La especificación de UML 2.0 sólo se limita a definir el concepto de perfil y sus contenidos. A la hora de definir un perfil UML se propone seguir los siguientes pasos:

1. Definir el dominio de aplicación a modelar con el perfil. Si no existiese, entonces se definiría dicho dominio utilizando los mecanismos del propio UML (ejemplo: clases, relaciones de herencia y asociaciones), de la forma usual como se haría si el objetivo no fuese definir un perfil

UML. Se debe incluir la definición de las entidades propias del dominio, las relaciones entre ellas, así como las restricciones que limitan el uso de estas entidades y de sus relaciones.

2. Definir el perfil. Dentro del paquete «profile» se incluye un estereotipo por cada uno de los elementos del dominio que se pretende incluir en el perfil. Estos estereotipos tendrán el mismo nombre que los elementos del dominio, estableciéndose de esta forma una relación entre el dominio y el perfil. En principio cualquier elemento que se hubiera necesitado para definir el dominio puede ser etiquetado posteriormente con un estereotipo.

3. Definir cuáles son los elementos de UML del dominio que se están extendiendo sobre los que es posible aplicar un estereotipo. Ejemplo de tales elementos son las clases, sus asociaciones, sus atributos, las operaciones, las transiciones y los paquetes. De esta forma cada estereotipo se aplicará a la metaclase de UML que se utilizó en el dominio para definir un concepto o una relación.

4. Definir los valores etiquetados de los elementos del perfil. Estos valores etiquetados van a ser los atributos que aparezcan en el dominio. Se debe incluir además, la definición de sus tipos, y sus posibles valores iniciales.

5. Definir las restricciones que forman parte del perfil, a partir de las restricciones del dominio. Por ejemplo, las multiplicidades de las asociaciones que aparecen en el dominio, o las propias reglas de negocio de la aplicación deben traducirse en la definición las correspondientes restricciones. (6)

2.3. Perfil para el Modelo del Análisis en la Línea Soluciones Integrales (DATEC).

Los patrones de interfaz de usuario favorecen el desarrollo del proyecto porque se proyecta con antelación los elementos que se van a requerir en el mismo, previniendo que se realice una inversión de tiempo en la arquitectura de componentes que no sean necesarios. Además de lo antepuesto, estos patrones permiten concretar un alcance que deja claro los resultados que se quieren conseguir. El objetivo de este perfil es proporcionar un lenguaje común para describir la aplicación de patrones de interfaz de usuario clásicos de los sistemas de gestión, estos patrones son los utilizados en la línea Soluciones Integrales, los mismos se conocen como: Vista Detalle, Vista Maestro, Vista Maestro Activa, Vista Maestro/Detalle y Acción. Además constituyen buenas prácticas en el desarrollo de sistemas de gestión y están en correspondencia con la experiencia del usuario.

Modelo del Dominio.

En el diagrama se muestran los patrones utilizados por el Grupo de Análisis. Estos patrones están orientados específicamente a las aplicaciones de gestión y son utilizados al elaborar los prototipos de IU durante la especificación de los casos de uso.

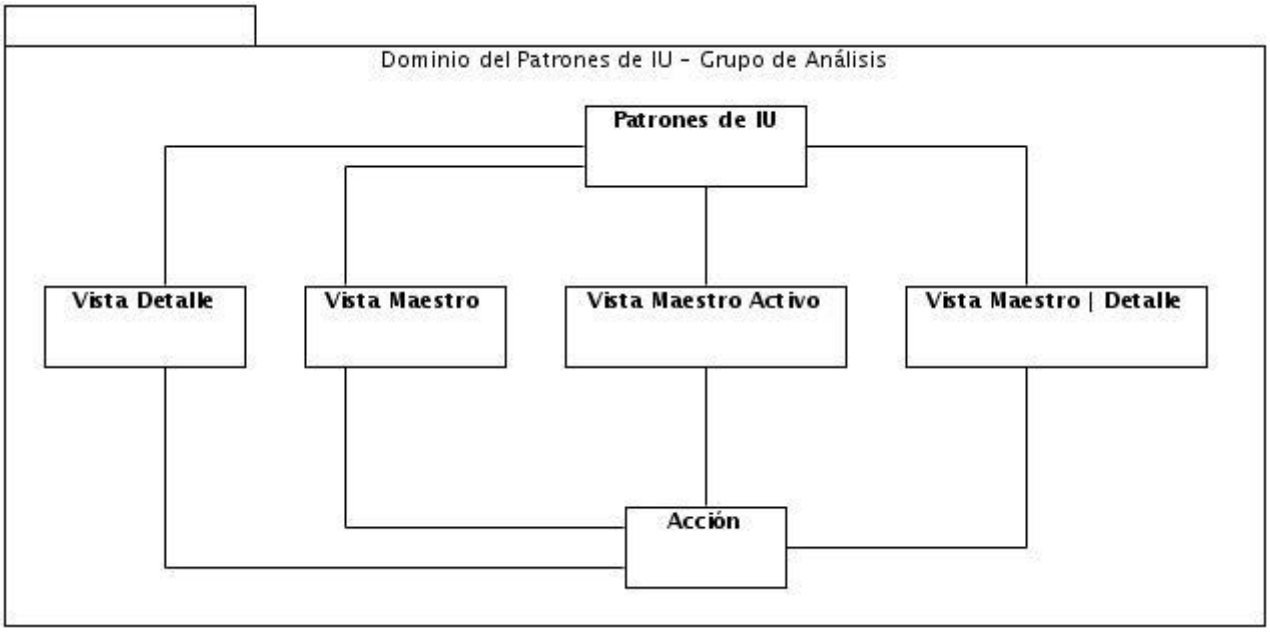


Figura No3. Modelo del Dominio para el Análisis.

El Modelo de Dominio para el Análisis está compuesto por los patrones de interfaz de usuario Vista Detalle, Vista Maestro, Vista Maestro Activo, Vista Maestro Detalle y Acción. El mismo se resume, en que cada una de las Vistas antes mencionadas tienen acciones, las cuales constituyen las operaciones que van a tener las mismas, ejemplo de estas operaciones pueden ser: Filtrar, Ordenar y Buscar. La Vista Detalle generalmente se relaciona con la información que se puede actualizar o modificar de una entidad individual. La vista Maestro se asocia más bien al listado de un grupo de entidades. La Vista Maestro Activo, es igual que la Vista Maestro con la diferencia de que al seleccionar uno de los elementos que la componen se pueden modificar o actualizar los campos de esa entidad, y la Vista Maestro Detalle, también consiste en lo mismo que la vista Maestro, pero con la particularidad de que al seleccionar uno de las entidades que la conforman, muestra una Vista Detalle en la cual se pueden actualizar o modificar los campos de esa entidad. A continuación se amplía más el concepto de cada uno de estos patrones, a partir de la descripción de las entidades.

Descripción de las entidades.

1. **Vista Detalle**. La Vista Detalle es el patrón utilizado para prototipar la IU asociada a la información de una entidad en particular, generalmente es un formulario mostrando cada uno de los campos de la entidad. **Estereotipo:** detail view**. Metaclase UML 2.0 a la que amplía:** Class**.**

2. **Vista Maestro**. La Vista Maestro como patrón se utiliza para mostrar una colección generalmente en forma de lista de entidades, permitiendo en muchos casos la ordenación, la búsqueda, el filtrado a través de los atributos de la entidad mostrados. **Estereotipo:** master view**. Metaclase UML 2.0 a la que amplía:** Class**.**

3. **Acción**. Son las acciones que el usuario puede realizar, las acciones en un diseño de IU se corresponden con botones u opciones de menú. **Estereotipo:** action**. Metaclase UML 2.0 a la que amplía:** Method**.**

4. **Vista Maestro Activo**. La Vista Maestro Activo como patrón realiza las mismas funciones que la vista maestro con la única diferencia que esta le permite al usuario editar y modificar en la misma Vista. **Estereotipo:** master active view**. Metaclase UML 2.0 a la que amplía:** Class**.**

5. **Vista Maestro Detalle**. La Vista Maestro Detalle realiza las mismas funciones que la Vista Maestro con la diferencia que en esta vista al seleccionar una entidad del conjunto, el sistema muestra una Vista Detalle con los elementos de esa entidad (puede ser en forma de formulario). **Estereotipo:** detail master view**. Metaclase UML 2.0 a la que amplía:** Class**.**

Especificación del Perfil.

En el siguiente diagrama se muestra el perfil definido, utilizando la notación de UML 2.0 para la especificación del mismo. En esta especificación se define un estereotipo por cada una de las entidades definidas en el Modelo de Dominio, las cuales tienen el mismo nombre y de este modo se establece una relación entre el modelo de dominio y el perfil. Los estereotipos definidos fueron: detail view, master view, active master view y master detail view que extienden a la metaclase Class, esto significa que cuando se utiliza un elemento de tipo detail view, master view, active master view y master detail view en los modelos UML, se está asegurando que ese elemento sigue las mismas

reglas semánticas que un elemento de tipo Class, ya que se ha extendido el metamodelo, pero en ningún caso se ha modificado su semántica, de igual modo ocurre con el estereotipo action que extiende a la metaclase Method. Esta especificación tiene como restricción que los estereotipos propuestos son aplicables solamente a clases de tipo <<boundary>>. A continuación se define la semántica para cada uno de los estereotipos definidos en la especificación del perfil.

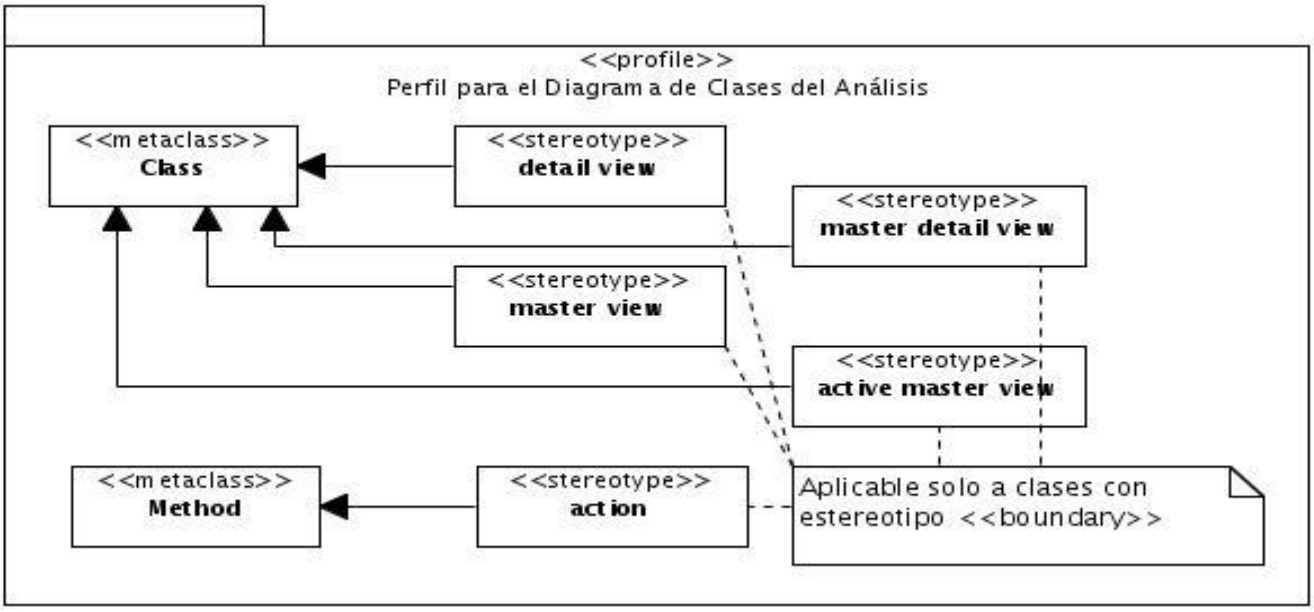
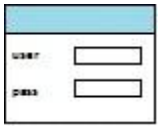


Figura No4. Especificación del perfil UML para el Análisis.

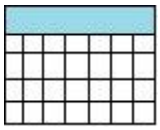
Especificación de los estereotipos.



Estereotipo: <<detail view>>

Meta clase(s) UML a la que amplía: Class

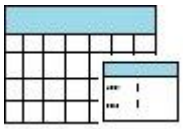
Semántica del estereotipo: Debe ser aplicado a clases boundary en un diagrama de clases del análisis, esto implica para el analista, que a la hora de realizar el prototipo se debe cumplir con las pautas ó descripción de este patrón para esta clase en particular. Para un arquitecto se trataría de un componente de la capa de presentación cuyo modelo está asociado a la entidad que visualiza. Para un desarrollador significa un panel de formulario en Ext JS, con cada uno de los campos de la entidad.



Estereotipo: <<master view>>

Meta clase(s) UML a la que amplía: Class

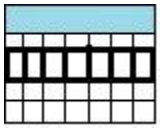
Semántica del estereotipo: Debe ser aplicado a clases <<boundary>> en un diagrama de clases del análisis esto implica para el analista que se va a trabajar con este patrón, por lo que a la hora de realizar el prototipo se debe cumplir con las pautas de este patrón para esta clase en particular. Para un arquitecto se trataría de un componente cuyo modelo está asociado a un listado de entidades que se visualizan. Para un desarrollador significa un grid en Ext JS, listando cada una de las entidades.



Estereotipo: <<master detail view>>

Meta clase(s) UML a la que amplía: Class

Semántica del estereotipo: Debe ser aplicado a clases <<boundary>> en un diagrama de clases del análisis esto implica para el analista que se va a trabajar con este patrón por lo que a la hora de realizar el prototipo se debe cumplir con las pautas de este patrón para esta clase en particular. Para un arquitecto se trataría de un componente cuyo modelo está asociado a un listado de entidades y que al seleccionar algún elemento de este listado daría paso a un componente de la capa de presentación cuyo modelo está asociado a la entidad que visualiza. Para un desarrollador significa un panel que contiene un grid con el listado de las entidades y un formulario en Ext JS, visualizando los datos de la entidad seleccionada, permitiendo su edición.



Estereotipo: <<active master view>>

Meta clase(s) UML a la que amplía: Class

Semántica del estereotipo: Debe ser aplicado a clases <<boundary>> en un diagrama de clases del análisis esto implica para el analista que se va a trabajar con este patrón por lo que a la hora de realizar el prototipo se debe cumplir con las pautas de este patrón para esta clase en particular. Para un arquitecto se trataría de un componente cuyo modelo está asociado a un listado de entidades y que

al seleccionar algún elemento de este listado da la oportunidad de editar en esa misma interfaz. Para un desarrollador significa un grid editable que permita modificar los campos que sean editables de una entidad seleccionada.

Estereotipo: <<action >>

Meta clase(s) UML a la que amplía: Method.

Semántica del estereotipo: Debe ser aplicado a clases <<boundary>> en un diagrama de clases del análisis esto implica para el analista que se va a trabajar con este patrón por lo que a la hora de realizar el prototipo se debe cumplir con las pautas de este patrón para esta clase en particular. Para un arquitecto se trataría de un componente cuyo modelo está asociado al comportamiento y restricciones del mismo. Para un desarrollador significa las operaciones que se visualizan en la interfaz de usuario de manera explícita.

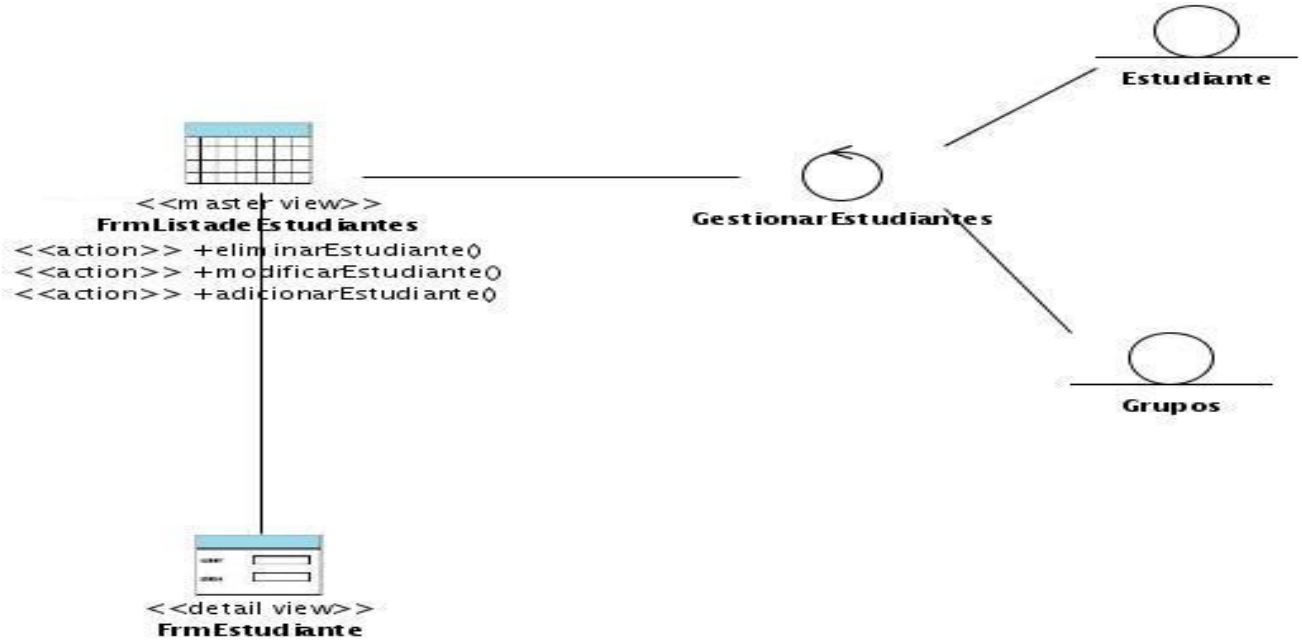


Figura No5. Ejemplo de uso en un modelo.

Ej.: En el siguiente modelo se visualizan las interfaces “FrmListadeEstudiantes” y “FrmEstudiantes”, la primera muestra un listado con todos los estudiantes en forma de grid, en la que al seleccionar uno de ellos se muestra la segunda interfaz en forma de formulario con los datos del estudiante seleccionado, posibilitando modificar cualquier valor editable del mismo, dichos valores son enviados a la clase controladora “GestionarEstudiantes”, la cual procesa dicha información haciendo uso de las clases

entidades “Estudiantes” y “Grupos”.

2.4. Perfil para el Modelo del Diseño en la Línea de Soluciones Integrales (DATEC).

El objetivo de este perfil es describir las clases del diseño que están presentes en los patrones de interfaz de usuario que fueron seleccionados durante la etapa de análisis mostrando sus atributos, métodos y propiedades así como relaciones entre dichas clases, adicionalmente estos patrones constituyen buenas prácticas en el desarrollo de sistemas de gestión y están en correspondencia con la experiencia del usuario.

Modelo del Dominio de Aplicaciones Symfony.

En el diagrama se muestran las estructuras que conforman un proyecto Symfony, así como las relaciones entre ellas permitiendo conocer la arquitectura y organización de los elementos dentro del proyecto.

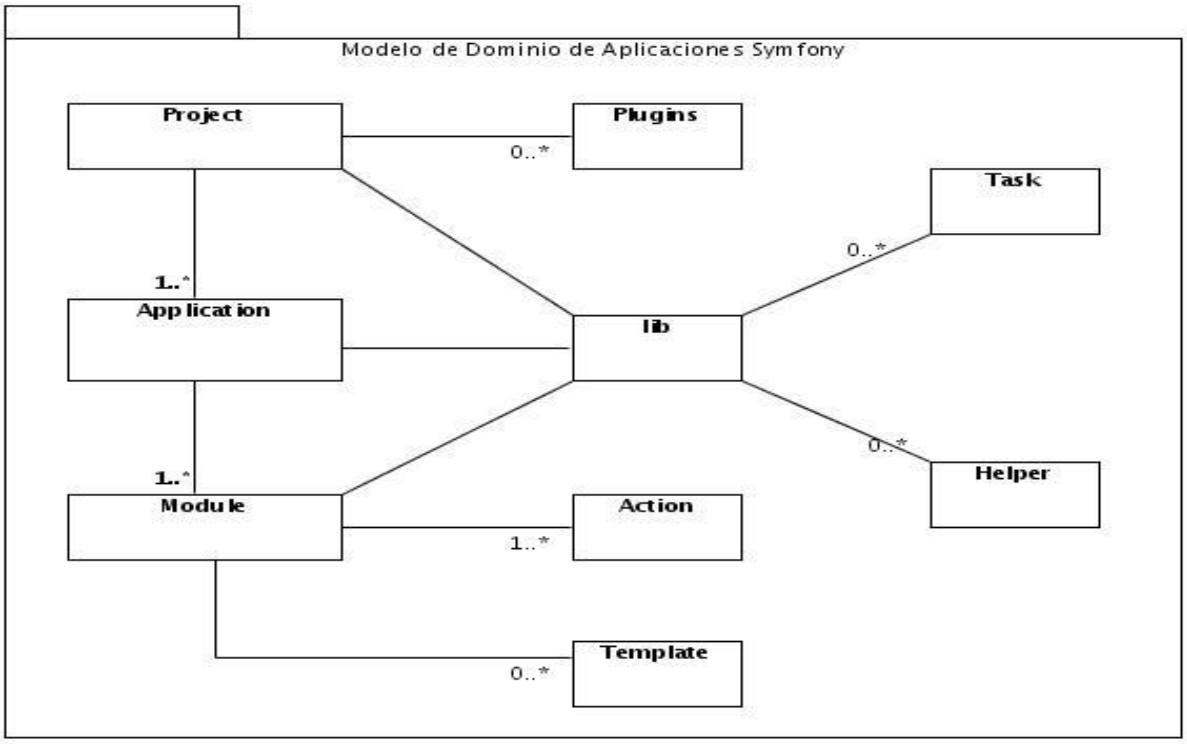


Figura No6. Modelo del Dominio de Aplicaciones Symfony.

En este Modelo de Dominio para Aplicaciones Symfony se definieron las entidades Project, Plugins, Application, Lib, Task, Helper, Module, Action y Templete. Según lo expresado por el modelo, dentro de

un proyecto en Symfony, las operaciones se agrupan de forma lógica en aplicaciones. Cada aplicación está formada por uno o más módulos. Un módulo normalmente representa a una página web o a un grupo de páginas con un propósito relacionado, los mismos pueden tener una o muchas acciones y pueden o no tener plantillas. Los módulos, las aplicaciones y los proyectos tienen además, asociadas librerías y estas pueden o no, tener varios Task y Helper, la primera se usa para generar las clases del modelo de datos y la segunda genera un grupo de ayudas las cuales facilitan el trabajo de los miembros del equipo de proyecto. Los proyectos en general pueden o no tener asociados plugins, los cuales permiten agrupar todo el código diseminado por diferentes archivos y reutilizar este en otros proyectos. A continuación se realiza de una manera más amplia la descripción de cada una de las entidades que conforman este Modelo de Dominio.

Descripción de las entidades.

1. **Project**. Es un conjunto de servicios y operaciones disponibles bajo un determinado nombre de dominio y que comparten el mismo modelo de objetos. **Estereotipo**: symfony project. **Metaclase(s) UML a la que amplía:** Package.

2. **Application**. Son todas las operaciones dentro de un proyecto que se agrupan de forma lógica. **Estereotipo**: symfony application. **Metaclase(s) UML a la que amplía:** Package.

3. **Module**. Representa a una página web o a un grupo de páginas con un propósito relacionado. Cada módulo tiene su propio subdirectorio dentro del directorio modules y el nombre del directorio es el que se elige durante la creación del mismo. **Estereotipo**: symfony module. **Metaclase(s) UML a la que amplía:** Class.

4. **Action**. Son cada una de las operaciones que se puede realizar en un módulo. **Estereotipo**: symfony action. **Metaclase(s) UML a la que amplía:** Method.

5. **Task**. Es usada para crear la estructura inicial de directorios del proyecto y de la aplicación y también para generar las clases del modelo de datos. **Estereotipo**: symfony task. **Metaclase(s) UML a la que amplía**: Class.

6. **Helper**. Incluyen un grupo de funcionalidades, ayudas y utilidades que facilitan el trabajo del desarrollador. **Estereotipo**: symfony helper. **Metaclase(s) UML a la que amplía:** Class.

7. **Plugin**. Es una extensión encapsulada para un proyecto Symfony. Los plugins permiten agrupar todo el código diseminado por diferentes archivos y reutilizar este código es otros proyectos**. Estereotipo**: symfony plugin. **Metaclase(s) UML a la que amplía:** Package.

8. **Templete**. Plantillas ubicadas dentro de los módulos de las aplicaciones, solo contiene código web. **Estereotipo**: symfony templete. **Metaclase(s) UML a la que amplía:** Class.

Especificación del perfil.

En este perfil se definió un estereotipo por cada elemento definido en el Modelo de Dominio para Aplicaciones Symfony. Se definieron los estereotipos symfony project, symfony plugin y symfony application que extienden a la metaclase Package, lo cual significa que cuando se usa un elemento de tipo symfony project, symfony plugin y symfony application en los modelos UML, se está asegurando que ese elemento sigue las mismas reglas semánticas que un elemento de tipo Package, ya que se ha ampliado el metamodelo, pero en ningún caso se ha modificado su semántica, de igual manera ocurre con los estereotipos symfony templete, symfony task, symfony module y symfony helper que extienden a la metaclase Class, y el estereotipo symfony action que extiende a la metaclase Method. A continuación se define la especificación para cada uno de los estereotipos propuestos.

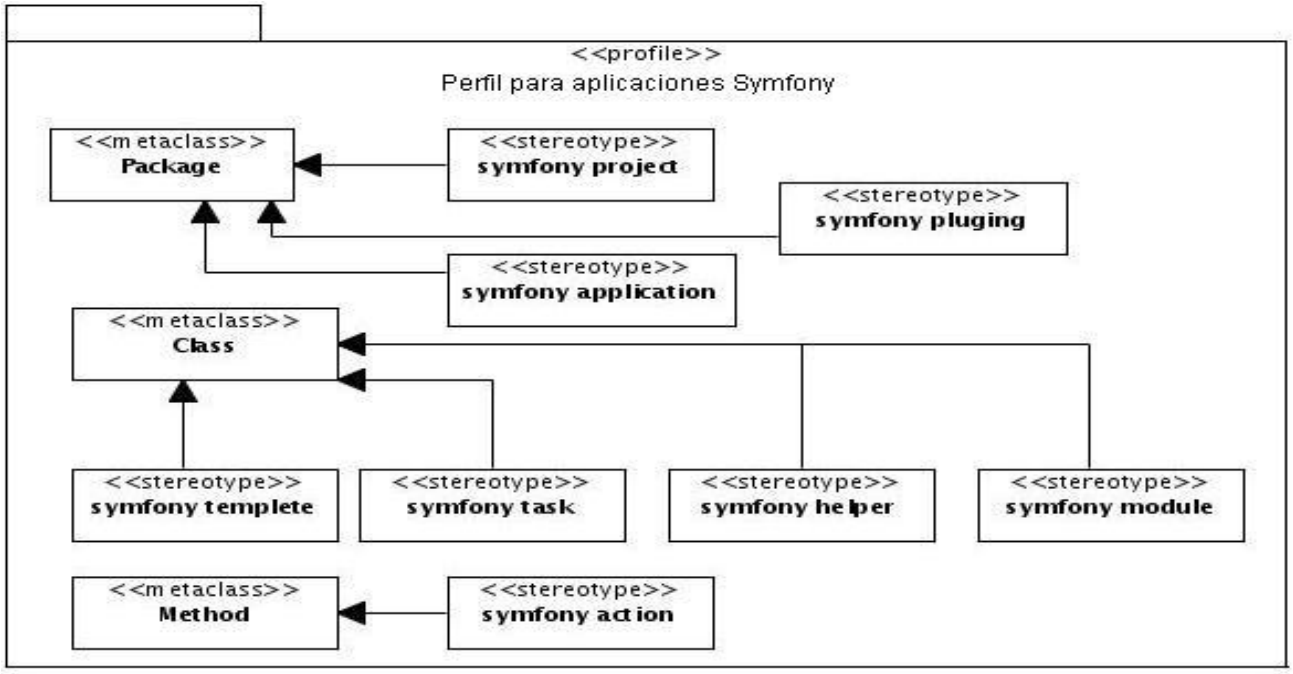


Figura No7. Especificación del perfil para aplicaciones Symfony.

Especificación de los estereotipos.

Estereotipo: <<symfony project>>

Metaclase(s) UML a la que amplía: Package

Semántica del estereotipo: Este estereotipo representa un proyecto en el que dentro de él, las operaciones se agrupan de forma lógica en aplicaciones.

Estereotipo: <<symfony application>>

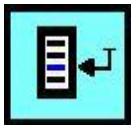
Metaclase(s) UML a la que amplía: Package

Semántica del estereotipo: Cada aplicación está formada por uno o más módulos. Un módulo normalmente representa a una página web o a un grupo de páginas con un propósito relacionado. Normalmente un proyecto debe contener dos aplicaciones: una para la parte pública y otra para la parte de gestión, compartiendo ambas la misma base de datos. En este caso, es importante tener en cuenta que los enlaces entre aplicaciones se deben indicar de forma absoluta.

Estereotipo: <<symfony module>>

Metaclase(s) UML a la que amplía: Class

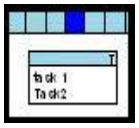
Semántica del estereotipo: Este estereotipo representa un módulo en el que se van a almacenar las acciones, las cuales constituyen cada una de las operaciones que se puede realizar en el mismo.



Estereotipo: <<symfony action>>

Metaclase(s) UML a la que amplía: Method

Semántica del estereotipo: Este estereotipo representa las operaciones o métodos que se realizan dentro de los módulos, normalmente las acciones se describen mediante verbos. Trabajar con acciones es muy similar a trabajar con las páginas de una aplicación web tradicional.



Estereotipo: <<symfony task>>

Metaclase(s) UML a la que amplía: Class

Semántica del estereotipo: Este estereotipo representa un conjunto de funciones de Symfony como la rotación de los logs y la actualización de las aplicaciones, durante la ejecución de una aplicación en el servidor de producción.

Estereotipo: <<symfony helper>>

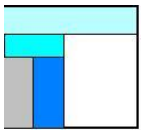
Metaclase(s) UML a la que amplía: Class

Semántica del estereotipo: Su representación debe verse como un método sencillo para mantener la aplicación limpia y fácil de mantener. Además, los <<symfony helper>> incluyen una serie de utilidades y atajos que no son recomendables desaprovechar.

Estereotipo: <<symfony plugins>>

Metaclase(s) UML a la que amplía: Package

Semántica del estereotipo: El estereotipo <<symfony plugin>> permiten agrupar todo el código diseminado por diferentes archivos y reutilizar este código en otros proyectos además permite encapsular clases, filtros, mixins, helpers, archivos de configuración, tareas, módulos, esquemas y extensiones para el modelo. Este estereotipo además permite no solamente reutilizar código propio, sino también aprovechar los desarrollos realizados por otros programadores y añadir al núcleo de Symfony extensiones realizadas por otros desarrolladores.



Estereotipo: <<symfony templete>>

Metaclase(s) UML a la que amplía: Class

Semántica del estereotipo: Este estereotipo representa un archivo que está ubicado dentro de un módulo, y su nombre está compuesto por el nombre de la acción y el resultado de la misma sólo debe contener código de presentación y no es más que la parte web de la aplicación.

Ejemplo del perfil aplicado.

Este ejemplo representa un proyecto en Symfony a través del estereotipo <<symfony project>> que está compuesto por una aplicación, representada por el estereotipo <<symfony application>> y esta aplicación está compuesta por cinco módulos los cuales se representan a través del estereotipo symfony module.

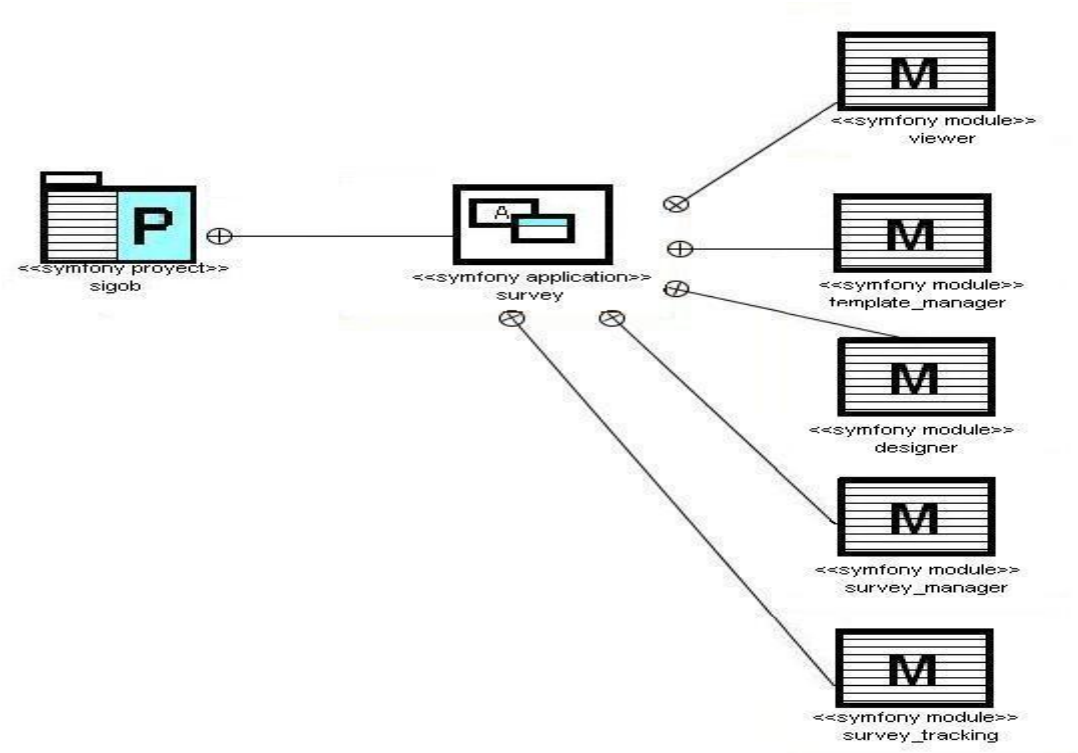


Figura No8. Ejemplo del perfil aplicado para Symfony.

Modelo del Dominio para Aplicaciones con Java Script y Ext JS.

En el diagrama se muestran las estructuras que conforman un proyecto en Java Script y Ext JS, así como las relaciones entre ellas permitiendo conocer la arquitectura y organización de los elementos

dentro del proyecto. A continuación se muestra el Modelo de Dominio para Aplicaciones con Java Script y Ext JS.

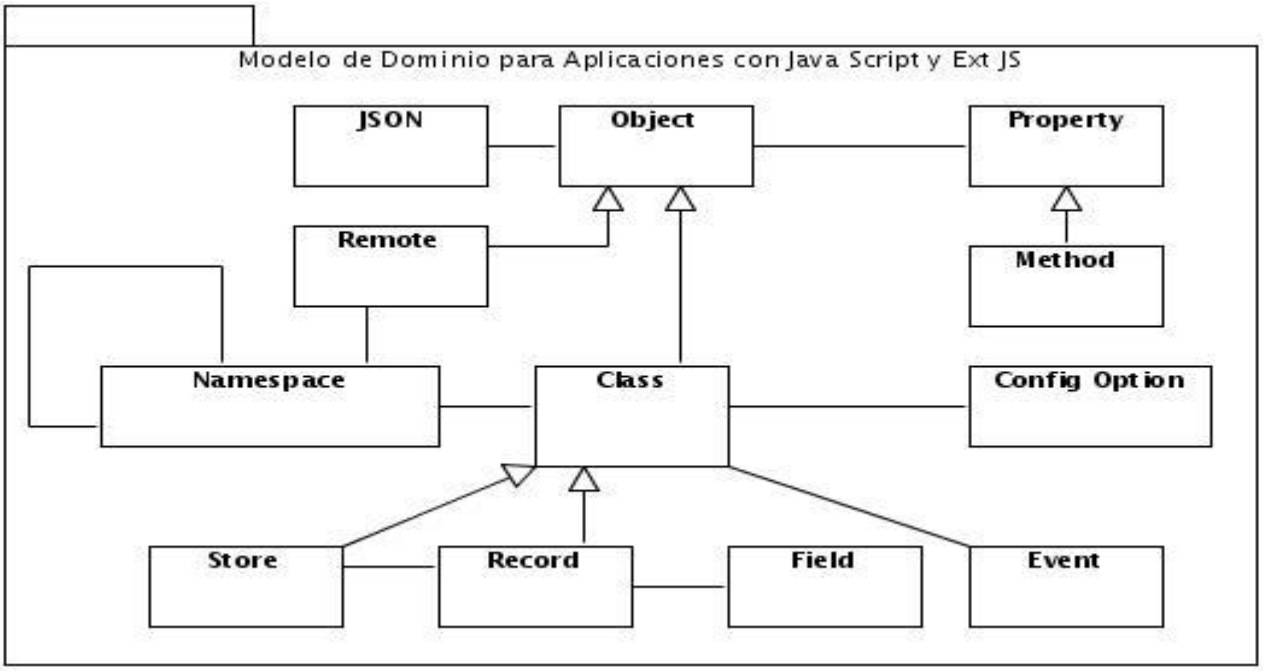


Figura No9. Modelo del Dominio para Aplicaciones con Java Script y Ext JS.

La entidades que se identifican en el Modelo de Dominio son las siguientes: JSon, Object, Remote, Namespace, Class, Property, Method, Config Option, Store, Record, Field y Event. Este modelo se traduce de la siguiente manera: un JSon (Notación de Java Script Orientada a Objetos) tiene objetos que están compuestos por las propiedades de cada componente de Ext.js; los Remote que sirven para marcar interfaces cuyos métodos se van a concebir vía RPC (Remote Protocole Call) y las clases JavaScript, son objetos y los métodos que se utilicen van a constituir las propiedades de cada componente de Ext.js. Los Remote además tienen asociados Namespace, los cuales se utilizan con el fin de empaquetar de manera lógica y organizada cada uno de los componentes desarrollados en el centro. Las entidades Namespace pueden tener asociadas otras Namespace y también clases Java Script, estas clases están compuestas por opciones de configuración y por eventos, la primera representan las opciones asociadas a un componente de Ext.js y la segunda representa cada una de los eventos asociados a un componente Java Script. Las entidades Store y Record son clases, el primero representa la fuente o almacén de datos que usará un componente de Ext JS, y la segunda está compuesta por varios Record que constituyen una entidad de este almacén de datos, por lo que al ser una entidad de ese almacén se compone por campos que caracterizan esa entidad. A continuación

se realiza la descripción de las entidades que conforman el Modelo de Dominio, incluyendo además el nombre del estereotipo propuesto y la metaclase a la que extiende en la especificación del perfil.

Descripción de las entidades.

JSon. Representa la notación de Java Script orientada a Objetos. **Estereotipo**: json. **Metaclase UML a la que amplía:** Class.

Remote. Sirve para marcar interfaces cuyos métodos se van a concebir vía RPC (Remote Protocole Call). **Estereotipo**: remote. **Metaclase UML a la que amplía:** Class.

Namespace. Se utiliza con el fin de empaquetar de manera lógica y organizada cada uno de los componentes desarrollados en el centro, ya que Java Script no incorpora de forma nativa los espacios de nombres. **Estereotipo**: js namespace. **Metaclase UML a la que amplía:** Package.

Store. Representa la fuente o almacén de datos que usará el componente. **Estereotipo**: store. **Metaclase UML a la que amplía:** Class.

Record. Representa una entidad del almacén de datos. **Estereotipo**: record. **Metaclase UML a la que amplía:** Class.

Field. Representa un campo o columna de la entidad. **Estereotipo**: field. **Metaclase UML a la que amplía:** Attribute.

Event. Representa cada una de los eventos asociados con un componente en Java Script, lo que serían funciones que se ejecutan en determinada acción. **Estereotipo**: event. **Metaclase UML a la que amplía:** Method.

Property. Representa las propiedades de cada componente de Ext.js. **Estereotipo**: property. **Metaclase UML a la que amplía:** Attribute.

Config Option. Representa las opciones de configuración asociado a un componente de Ext.js. **Estereotipo**: config option. **Metaclase UML a la que amplía:** Attribute.

Class. Se usará para representar una clase en JavaScript, lo que en la realidad corresponde con la definición del prototipo de la función constructora, dado que el concepto de clase propiamente no

existe en la versión 3.x de ECMA Script. **Estereotipo**: js class. **Metaclase UML a la que amplía:** Class.

Especificación del perfil.

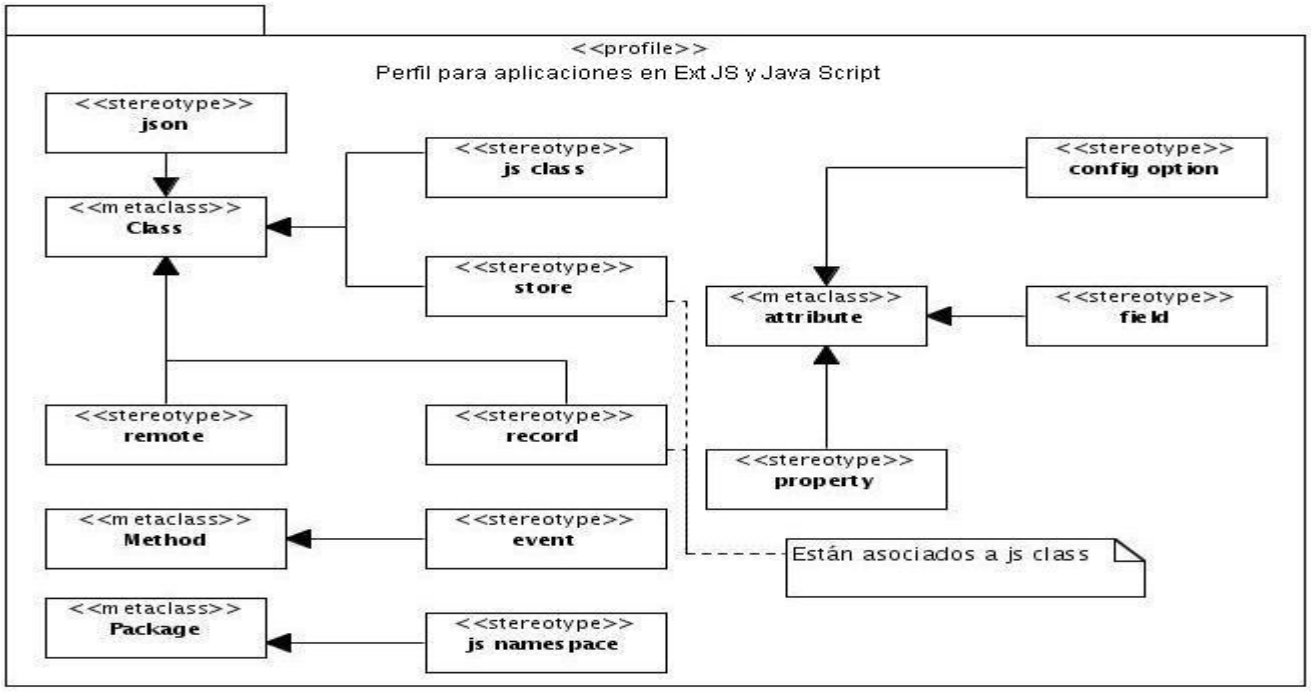


Figura No10. Especificación del perfil para aplicaciones en Ext JS y Java Script.

En esta especificación del perfil se definen un grupo de estereotipos a partir del Modelo de Dominio, los cuales extienden a metaclases formalizando el lenguaje de modelado. Cada metaclase a la que extiendan los estereotipos significa que cuando se usa un elemento del tipo del estereotipo especificado en los modelos UML, se está asegurando que ese elemento sigue las mismas reglas semánticas que un elemento del tipo de la metaclase a la que extiende, ya que se ha ampliado el metamodelo, pero en ningún caso se ha modificado su semántica. Los estereotipo json, js class, store, record y remote extienden a la metaclase Class. Los estereotipos config option, field y property extienden a la metaclase Attribute. El estereotipo event extiende a la metaclase Method y el estereotipo js namespace extiende a la metaclase Package. Este perfil tiene como restricción que los estereotipos store y record están asociados a las clases Java Script. A continuación se define la semántica para cada uno de los estereotipos propuestos.

Semántica de los estereotipos.

 **Estereotipo: <<jsclass>>**

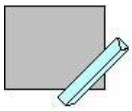
Metaclase UML a la que amplía: Class

Semántica del estereotipo: Este estereotipo representa una clase Java Script orientada a componentes con sus interfaces de comunicación definidas explícitamente a través de los servicios que esta provee.

 **Estereotipo: <<event>>**

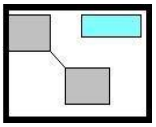
Metaclase UML a la que amplía: Method

Semántica del estereotipo: Este estereotipo representa una acción externa que es lanzada para aquellos componentes interesados en capturarla, es un complemento más en el comportamiento de un componente.

 **Estereotipo: <<config option>>**

Metaclase UML a la que amplía: Attribute

Semántica del estereotipo: Este estereotipo representa los atributos de configuración con los cuales un componente puede ser inicialmente declarado.

 **Estereotipo: <<js namespace>>**

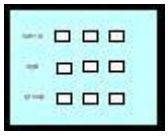
Metaclase UML a la que amplía: Class

Semántica del estereotipo: Este estereotipo representa la estructura lógica a la que pueden pertenecer uno o más componentes del sistema.

Estereotipo: <<store>>

Metaclase UML a la que amplía: Class

Semántica del estereotipo: Este estereotipo representa el almacén de datos que se le puede asignar a un componente que tenga concebida en su definición dicha propiedad de almacenamiento de datos.



Estereotipo: <<record>>

Meta clase(s) UML a la que amplia: Class

Semántica del estereotipo: Representa una entidad del almacén de datos.

Estereotipo: <<field>>

Metaclase UML a la que amplía: Attribute

Semántica del estereotipo**:** Este estereotipo representa un campo de una entidad que pertenece a un almacén de datos, habitualmente se encontrará representado en formularios.

Estereotipo: <<json>>

Metaclase UML a la que amplía: Class

Semántica del estereotipo: Este estereotipo es una notación que se utiliza para el intercambio de datos en una aplicación cliente-servidor. Es un formato de comunicación que facilita la independencia del cliente y del servidor.

Estereotipo: <<remote>>

Metaclase UML a la que amplía: Class

Semántica del estereotipo: Este estereotipo se aplicará a clases interfaz y servirán para marcar interfaces cuyos métodos se van a concebir vía RPC.

Estereotipo: <<property>>

Metaclase UML a la que amplía: Attribute

Semántica del estereotipo: Este estereotipo se usará con el fin de representar las propiedades y elementos que describen cada componente Ext.js.

Ejemplo de uso en un modelo

Ej.: Se utiliza el estereotipo <<js class>> en todas las clases del diseño dejando bien definido en dicho modelo a los desarrolladores que cada uno de estos componentes se implementarán explícitamente como clases de Java Script. Se utiliza el estereotipo <<record>> en la clase ¨RegisteredUser¨ para resaltar al desarrollador que esta clase es de tipo Ext.Data.Record. Se utiliza el estereotipo <<store>> para indicar que la clase debe ser de tipo Ext.Data.Store.

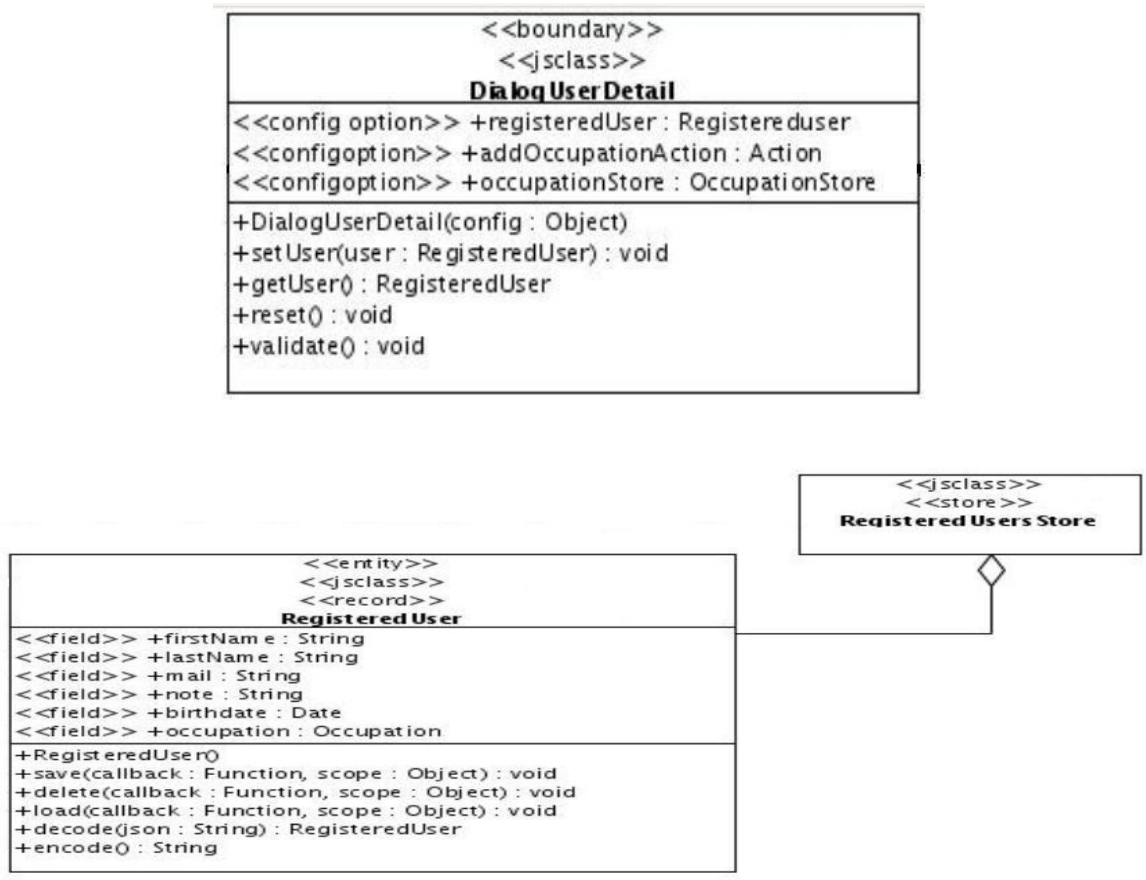


Figura No11. Ejemplo de uso del modelo.

2.5. Perfil para el Modelo de Implementación en la Línea de Soluciones Integrales (DATEC).

El objetivo de este perfil es identificar los componentes físicos de la implementación para que puedan comprenderse y gestionarse mejor, además explica cómo organizar e integrar los componentes a implementar basándose en las especificaciones de diseño. El Modelo de Implementación representa la composición física de la implementación en términos de subsistemas de implementación, y elementos de implementación.

Modelo de Dominio de la disciplina de Implementación.

En el diagrama se muestran las estructuras que conforman el Modelo de Dominio de la disciplina de Implementación, así como las relaciones entre ellas permitiendo conocer la arquitectura y organización de los elementos dentro del proyecto.

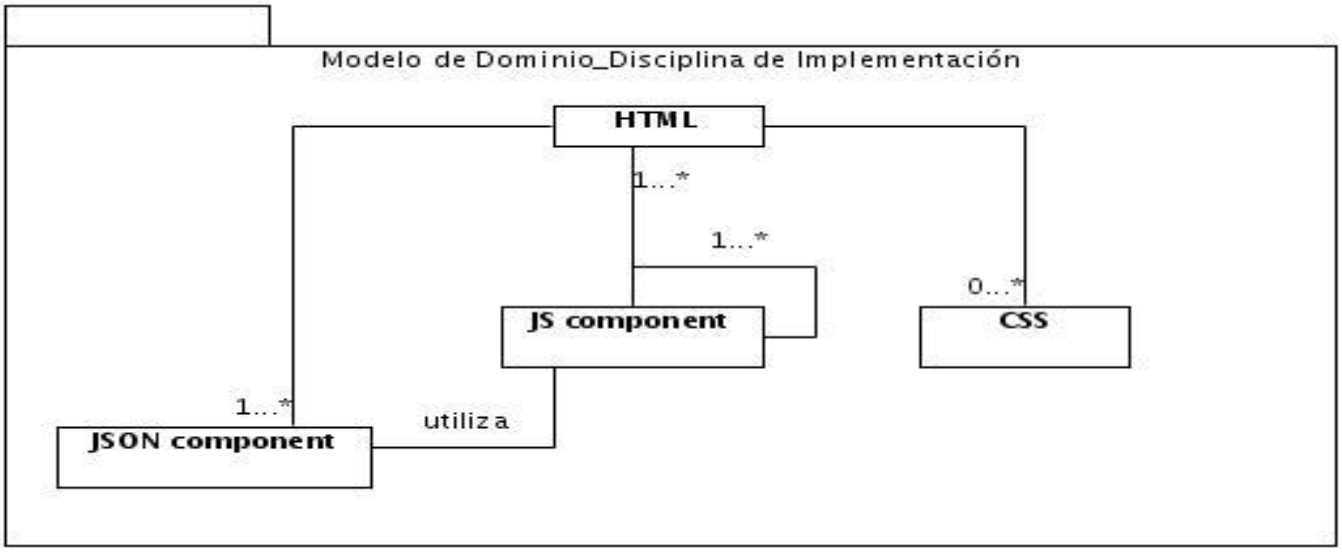


Figura No12. Modelo de Dominio para la disciplina de implementación.

En este Modelo de Dominio se definen los componentes HTML, CSS, JSON component y JS component. El componente HTML que representa cualquier archivo con código fuente en Java Script, está compuesto por uno o varios JSON component, que representan un formato sencillo para intercambiar datos, además tiene asociado varios JS component indicando cualquier archivo con código fuente en Java Script. El componente HTML además puede o no, tener varios CSS, este componente indica la tecnología que permite controlar la presentación de los documentos en la web. El componente JS component puede a su vez tener asociado otros JS component además de que utiliza

también un formato sencillo para intercambiar datos. A continuación se muestra un concepto más amplio de cada una de las entidades pertenecientes a este Modelo de Dominio.

Descripción de las entidades.

JS component. Representa cualquier archivo con código fuente en Java Script. Es la parte del sistema sustituible, casi independiente e importante que desempeña una función clara en el contexto de una arquitectura bien definida. Un componente cumple los requisitos de la realización de un conjunto de interfaces y proporciona, además, dicha realización. **Estereotipo**: js component. **Metaclase UML a la que amplía:** Component.

JSON component. Representa un formato sencillo para intercambiar datos. Consiste básicamente en un arreglo asociativo de JavaScript que se utiliza para incluir información del objeto. Ofrece 2 grandes ventajas: es muy fácil de leer en JavaScript y puede reducir el tamaño en bytes de la respuesta del servidor. **Estereotipo**: json component. **Metaclase UML a la que amplía:** Component.

CSS. Representa la tecnología que permite controlar la presentación de los documentos en la web, se pueden especificar muchos atributos de los elementos que conforman una página web:(color del texto, márgenes, el tipo de letras, tamaño y color, posicionar elementos). CSS funciona a base de reglas, las hojas de estilo están compuestas por una o más de esas reglas la regla tiene dos partes: un selector y la declaración. A su vez la declaración está compuesta por una propiedad y el valor que se le asigne. **Estereotipo**: css. **Metaclase UML a la que amplía:** Component.

HTML. Representa un lenguaje de marcado predominante para la elaboración de páginas web. Se usa para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. HTML también puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir un script, el cual puede afectar el comportamiento de navegadores web y otros procesadores de HTML. **Estereotipo**: HTML. **Metaclase UML a la que amplía:** Component.

Especificación del perfil.

En esta especificación del perfil para la implementación, se muestra la relación que existe entre los estereotipos definidos a partir del Modelo de Dominio, los cuales extienden a determinadas metaclases. Cuando un estereotipo extiende a una metaclase significa que cuando se usa un elemento del tipo del estereotipo especificado en los modelos UML, se está asegurando que ese elemento sigue

las mismas reglas semánticas que un elemento del tipo de la metaclase a la que extiende, ya que se ha ampliado el metamodelo, pero en ningún caso se ha reformado su semántica. En esta especificación del perfil se observan los estereotipos js component, css, json component y html que extienden a la metaclase Component, mientras que el estereotipo merge extiende a la metaclase Relation. A continuación se describe la especificación de los estereotipos propuestos.

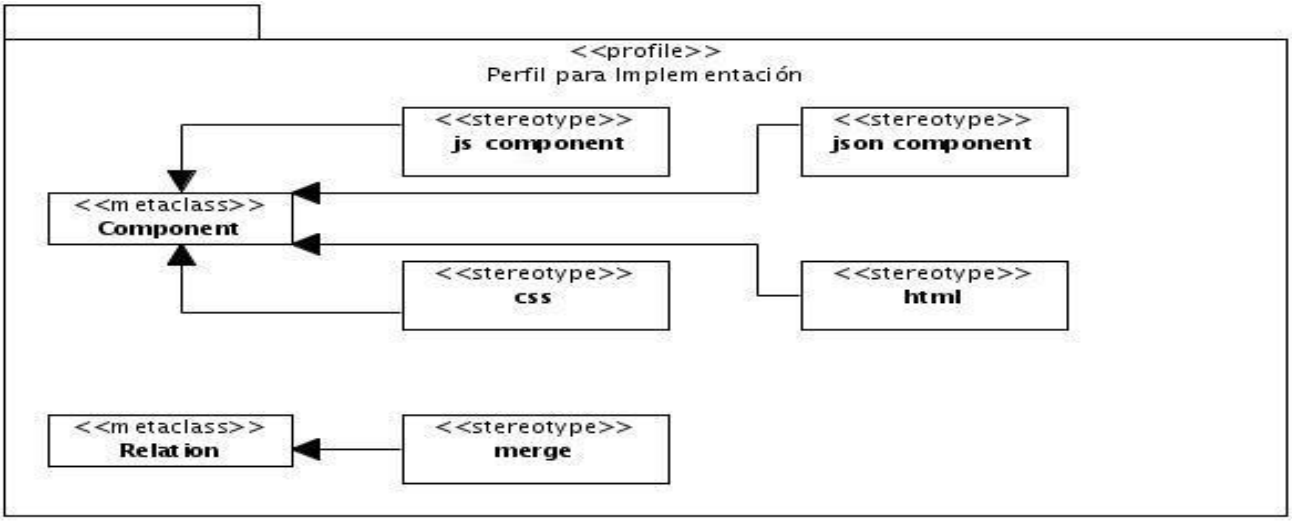


Figura No13. Especificación del perfil para la disciplina de implementación.

Especificación de los Estereotipos.

Estereotipo: <<js component>>

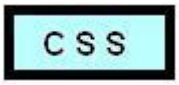
Metaclase UML a la que amplía: Component

Semántica del estereotipo: Representa componentes en Java Script, los cuales pueden depender de otros componentes y además utilizar JSON component.

Estereotipo: <<json component>>

Metaclase UML a la que amplía: Component

Semántica del estereotipo: Representa un archivo de texto cuya estructura corresponde a un JSON bien formado y está en correspondencia con el estándar que define JSON. En el entorno tecnológico de DATEC, algunos JSON son usados como archivos de configuración para las aplicaciones, particularmente en la capa de presentación.



Estereotipo: <<css>>

Metaclase UML a la que amplía: Component

Semántica del estereotipo: Representa el archivo físico, durante la implementación.

Estereotipo: <<html>>

Metaclase UML a la que amplía: Component

Semántica del estereotipo: Este estereotipo representa un archivo físico y además el lenguaje de marcado que describe la estructura de la página web y está orientado completamente a la web.

Estereotipo: <<merge>>

Metaclase UML a la que amplía: Component

Semántica del estereotipo: Representa una relación de mezcla entre varios componentes Java Script. Nota: el orden de la mezcla está en correspondencia con las dependencias topológicamente ordenadas de los js component.

Ejemplo de uso en un modelo.

En este ejemplo se describe cómo los elementos del modelo de diseño, como las clases, se implementan en términos de componentes, ficheros de código fuente, ejecutables, entre otros. Describe también cómo se organizan los componentes de acuerdo a los mecanismos de estructuración y modularización disponibles en el entorno de implementación y lenguaje o lenguajes de implementación empleados, y cómo dependen los componentes unos de otros.

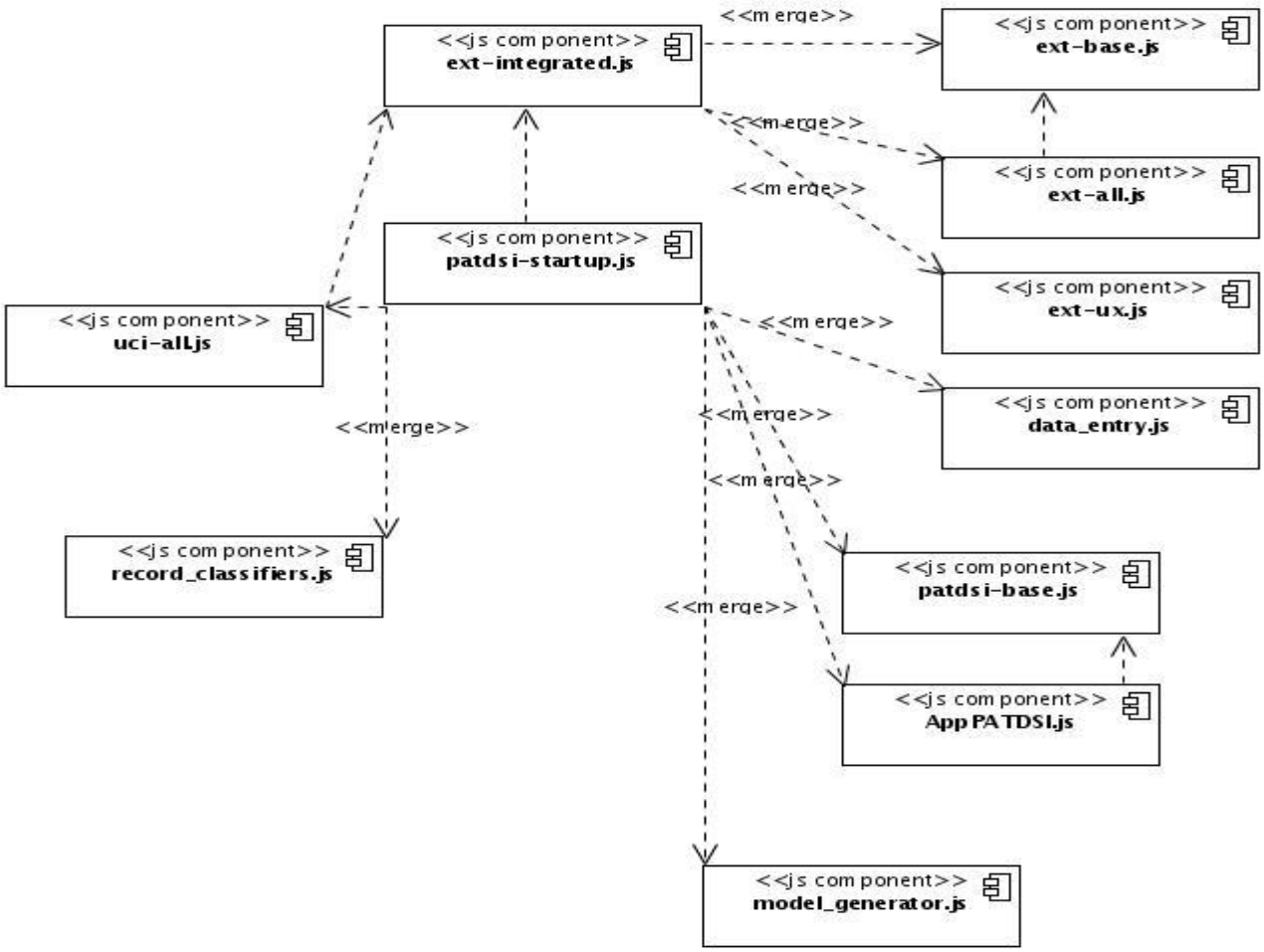


Figura No14. Ejemplo de uso del perfil para la disciplina de implementación.

Conclusiones parciales.

A partir de las necesidades de la línea Soluciones Integrales, se llevó a cabo la descripción del perfil de UML para representar las decisiones de análisis y diseño de los productos que la conforman. En esta descripción, se confeccionaron plantillas las cuales están contenidas por el Modelo de Dominio para el Análisis, Diseño e Implementación, la especificación del perfil y de las entidades que integran los distintos modelos de dominio, La extensión contiene además los necesarios estereotipos, valores etiquetados y restricciones para modelar de manera formal los elementos utilizados durante las etapas de análisis, arquitectura y desarrollo de la línea Soluciones Integrales de DATEC.

Capítulo 3: Verificación del perfil propuesto.

Introducción.

En este capítulo, se realiza un estudio relacionado al método Delphi, el cual será empleado con el objetivo de verificar la propuesta del perfil de UML elaborada en el capítulo 2. Luego del estudio de este método se selecciona un grupo de expertos que no son más que personas con el conocimiento necesario para determinar si la investigación realizada se aproxima a la realidad y a través de su criterio, facilitarán la verificación de la propuesta y darán un resultado negativo o positivo de la misma, estableciendo si el perfil de UML creado, está en condiciones de ser aplicado o no. Además la correcta elaboración del mismo, se puede comprobar a partir de lo establecido por la superestructura de UML, descrito en el capítulo anterior, ya que para la realización de este perfil, se siguieron correctamente los pasos que indica el paquete UML Profile 2.0, definido por la OMG.

3.1. Métodos generales de prospectiva.

Se define la prospectiva como: el conjunto de tentativas sistemáticas para observar a largo plazo el futuro de la ciencia, la tecnología, la economía y la sociedad con el propósito de identificar las tecnologías emergentes que probablemente produzcan los mayores beneficios económicos y sociales. Los métodos de prospectiva estudian el futuro en lo que se refiere a la evolución de los factores del entorno tecno-socio-económico y las interacciones entre estos factores. Estos métodos permiten que las organizaciones consigan desarrollar sus planes estratégicos con la seguridad de que se van a conseguir los objetivos a largo plazo. Entre los métodos generales de prospectiva se encuentran los siguientes:

Métodos de expertos: Se basan 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 que se tendrán en el futuro. Estos métodos utilizan como fuente de información un grupo de personas a las que se supone un conocimiento elevado de la materia que se va a tratar.

Métodos extrapolativos: En este método se proyectan hacia el futuro los datos de evolución que se tienen del pasado. Para ello se recopila la información histórica disponible y se buscan posibles tendencias o ciclos evolutivos. Estos proporcionarán los posibles entornos futuros.

Métodos de correlación: En éstos se intenta ver qué factores están implicados en un desarrollo y en qué grado influyen. Teniendo esto presente se determina cuál es la posible línea evolutiva que van a seguir todos estos factores. (15)

Para verificar el perfil propuesto se aplicará el método de expertos o método Delphi debido a que este presenta un grupo de ventajas que favorecen a que se realice con calidad la evaluación del perfil. Una de las ventajas más importantes que presenta la aplicación de los métodos expertos es que la información disponible está siempre más contrastada que aquella de la que dispone el participante mejor preparado, es decir, que la del experto más versado en el tema. Esta afirmación se basa en la idea de que varias cabezas son mejor que una. Otra ventaja consiste en que el número de factores que es considerado por un grupo, es mayor que el que podría ser tenido en cuenta por una sola persona. Cada experto podrá aportar a la discusión general la idea que tiene sobre el tema debatido desde su área de conocimiento. (16)

3.1.2. Condiciones para aplicar métodos de expertos.

- ✓ No existen datos históricos con los que trabajar. Un caso típico de esta situación es la previsión de implantación de nuevas tecnologías.
- ✓ El impacto de los factores externos tiene más influencia en la evolución que el de los internos. Así, la aparición de una legislación favorable y reguladora y el apoyo por parte de algunas empresas a determinadas tecnologías pueden provocar un gran desarrollo de estas que de otra manera hubiese sido más lento.
- ✓ Las consideraciones éticas o morales dominan sobre los económicas y tecnológicas en un proceso evolutivo. En este caso, una tecnología puede haber dificultado su desarrollo si este provoca un alto rechazo en la sociedad (Un ejemplo se tiene en la tecnología genética, que ve dificultado su avance por los problemas morales que implica la posibilidad de manipulación del genotipo). (16)

3.2. Método Delphi.

El método Delphi pretende extraer y maximizar las ventajas que presentan los métodos basados en grupos de expertos y minimizar sus inconvenientes. Para ello se aprovecha la sinergia del debate en el grupo y se eliminan las interacciones sociales indeseables que existen dentro del mismo. De esta

forma se espera obtener un consenso lo más fiable posible del panel de expertos. Este método presenta tres características fundamentales:

Anonimato: Durante un Delphi, ningún experto conoce la identidad de los otros que componen el grupo de debate. Esto tiene una serie de aspectos positivos, como son: impide la posibilidad de que un miembro del grupo sea influenciado por la reputación de otro de los miembros o por el peso que supone oponerse a la mayoría. La única influencia posible es la de la congruencia de los argumentos. Además permite que un miembro pueda cambiar sus opiniones sin que eso suponga una pérdida de imagen y el experto puede defender sus argumentos con la tranquilidad que da saber que en caso de que sean erróneos, su equivocación no va a ser conocida por los otros expertos.

Iteración y realimentación controlada: La iteración se consigue al presentar varias veces el mismo cuestionario. Como, además, se van presentando los resultados obtenidos con 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. (17)

3.3. Procedimiento a seguir.

El procedimiento a seguir para la verificación de la propuesta se centrará en las siguientes fases:

- ✓ **Fase preliminar**: Se delimita el contexto, los objetivos, el diseño, los elementos básicos del trabajo y la selección de los expertos.
- ✓ **Fase exploratoria**: Elaboración y aplicación de los cuestionarios a los expertos seleccionados en la fase anterior.
- ✓ **Fase final**: Análisis de los resultados y presentación de la información. (18)

3.4. Proceso de selección de los expertos.

Se entiende por experto a una persona reconocida como una fuente confiable de un tema, técnica o habilidad. De forma más general se puede decir que un experto es una persona con un conocimiento amplio en un área particular del conocimiento. Los expertos gracias a su educación, experiencia y

trabajos realizados, llegan a alcanzar un nivel más alto de conocimiento sobre un tema que excede el nivel de conocimiento de otra persona. (19)

Se debe tener en cuenta que ninguno de los expertos seleccionados conoce las identidades ni respuestas de los otros expertos, para de esta manera lograr que cada uno defienda sus ideas.

3.5. Particularidades para la selección de expertos.

Se seleccionaron 9 expertos, de las cuales se observaron el siguiente grupo de características, las cuales favorecen a la obtención de resultados con calidad y que las opiniones brindadas sean confiables y válidas para el objetivo propuesto.

- Graduado del nivel superior.
- Tener conocimiento sobre análisis, arquitectura o desarrollo de acuerdo al rol en el que se desempeñan (Específicamente estos roles, ya que son estos los que interactúan en la propuesta de perfil).
- Ser parte de la Universidad de las Ciencias Informáticas y conocer el modelo de producción de la universidad.
- Tener como mínimo 4 años de experiencia en proyectos de software.
- Tener como mínimo 2 trabajos publicados en eventos que validen su rol, relacionados con análisis, arquitectura o desarrollo.

A los expertos seleccionados se les realizó un cuestionario (Anexo #6), con el objetivo de conocer sus particularidades para saber si podían ser parte de la verificación de la propuesta, o sea si cumplían con los requisitos necesarios, por lo que en este cuestionario se recogen datos personales y conocimientos adquiridos de cada uno de ellos.

De los 9 expertos encuestados, solamente 7 fueron seleccionados para continuar con la verificación de la propuesta, debido a que dos de ellos tenían dos o menos trabajos publicados, lo cual impidió que fueran avalados como expertos de esta investigación.

A los expertos que fueron seleccionados se les aplicó otra encuesta que valida la competencia de los mismos.

3.6. Competencia de los expertos.

A partir de los resultados obtenidos de enviar el primer cuestionario a los especialistas, se lleva a cabo un análisis cuantitativo para medir la competencia de los expertos seleccionados. Para ello el método Delphi propone determinar el coeficiente de competencia mediante la fórmula: K= ½ (kc + ka), donde kc es el coeficiente de conocimientos y ka es el coeficiente de argumentación. Para calcular el coeficiente de conocimientos y el coeficiente de argumentación, Delphi propone dos tablas, en las cuales el experto emite una autoevaluación acerca de su conocimiento a partir del tema investigado.

kc = Se obtiene mediante una tabla que recoge una autoevaluación del posible experto, esta tabla contiene una escala del 0 al 10, en la que el experto señala con una X su grado de competencia sobre los temas sometidos a consideración.

Valor de kc.

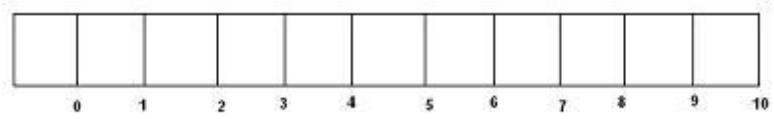


Figura No15. Tabla para calcular el valor de kc.

La marca de los expertos es n y el coeficiente de conocimiento se calcula a través de la fórmula:

kc = n * 0,1

Los resultados obtenidos de kc se muestran en la siguiente tabla:

Expertos	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Kc	0.6	0.8	0.8	0.9	0.8	0.7	0.7

Ka = Se obtiene después de analizar los resultados de la Pregunta 2 de la encuesta No.1, donde el experto debe marcar, según su criterio, su grado de competencia sobre la siguiente tabla patrón:

Valor de ka.

FUENTES DE ARGUMENTACION	Grado de influencia de cada una de las fuentes en sus criterios.		
	A (alto)	M (medio)	B (bajo)
Análisis teóricos realizados por usted			
Su experiencia obtenida			
Trabajos de autores nacionales			
Trabajos de autores extranjeros			
Su propio conocimiento del estado del problema en el extranjero			
Su intuición			
Totales			

Figura No16. Tabla para calcular el valor de ka.

Las marcas de los expertos se convierten en puntos, los valores que propone Delphi mostrados en la siguiente tabla:

FUENTES DE ARGUMENTACION	Grado de influencia de cada una de las fuentes en sus criterios.		
	A (alto)	M (medio)	B (bajo)
Análisis teóricos realizados por usted	0.3	0.2	0.1
Su experiencia obtenida	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 estado del problema en el extranjero	0.05	0.05	0.05
Su intuición	0.05	0.05	0.05
Totales	1.0	0.8	0.5

Figura No17. Valores de la tabla para calcular el valor de ka.

Los aspectos que influyen sobre el nivel de argumentación o fundamentación del tema de investigación permiten calcular el coeficiente de argumentación de cada experto a partir de la siguiente fórmula:

ka = a ni (n1 + n2 + n3 + n4+ n5 + n6)

a ni: marca correspondiente a la fuente de argumentación i.

Los resultados obtenidos de ka se muestran en la siguiente tabla:

Expertos	**1**	**2**	**3**	**4**	**5**	**6**	**7**
ka	0.8	0.8	0.7	0.9	0.9	0.5	0.8

A partir de los resultados obtenidos se procede a determinar el coeficiente de competencia K, a partir del coeficiente de conocimiento (Kc) y el coeficiente de argumentación (Ka). El coeficiente de competencia es quien finalmente determina qué experto se toma en consideración para trabajar en esta investigación.

Después de aplicar la fórmula: K = 0,5 (Kc + Ka) y obtenidos los resultados, estos se valoran de la siguiente manera:

- Si 0.8 <= k <=1.0, el coeficiente de competencia es alto
- Si 0.5 <= k < 0.8, el coeficiente de competencia es medio.
- Si k <0.5 el coeficiente de competencia es bajo.

Aquellos expertos cuyos resultados fueron el coeficiente de competencia medio y alto son los seleccionados para verificar la propuesta realizada. De los 7 expertos a los que se les aplicó la encuesta de autoevaluación, todos obtuvieron coeficiente de competencia medio y alto, por lo tanto todos son seleccionados para continuar con la verificación del perfil. A continuación se muestran los resultados obtenidos.

Expertos	**1**	**2**	**3**	**4**	**5**	**6**	**7**
K	0.6	0.8	0.7	0.9	0.8	0.6	0.7
Coeficiente	Medio	Alto	Medio	Alto	Alto	Medio	Medio

Figura No18. Resultados obtenidos de K.

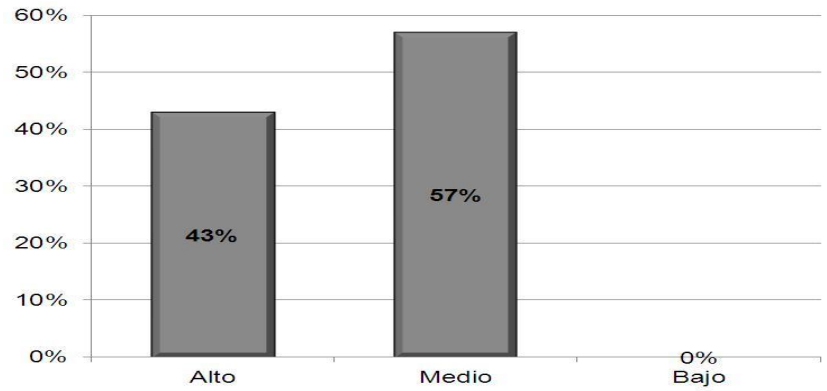


Figura No19. Grado de competencia de los expertos.

3.7. Confección del cuestionario.

Luego de seleccionar a los expertos cuyo coeficiente de competencia coincide con medio y alto se procede a elaborar la encuesta necesaria para realizar la evaluación de la propuesta, teniendo en cuenta aspectos como la elaboración de preguntas claras, precisas e independientes, tanto cuantitativas para calcular promedios, como cualitativas para la justificación de las opiniones. Se visitó a cada uno personalmente, el autor del trabajo le explicó punto por punto a cada experto los objetivos y resultados de la propuesta, luego se les dio la encuesta y el trabajo de diploma con un plazo de una semana de tiempo para entregarla.

3.8. Análisis de los resultados del cuestionario.

Después de haber realizado un análisis de los resultados del cuestionario enviado a los expertos con el fin de verificar la propuesta de perfil se concluye que:

En la pregunta, qué relevancia se le otorga a la investigación realizada el 71% respondió alta, mientras que el 29% respondió media.

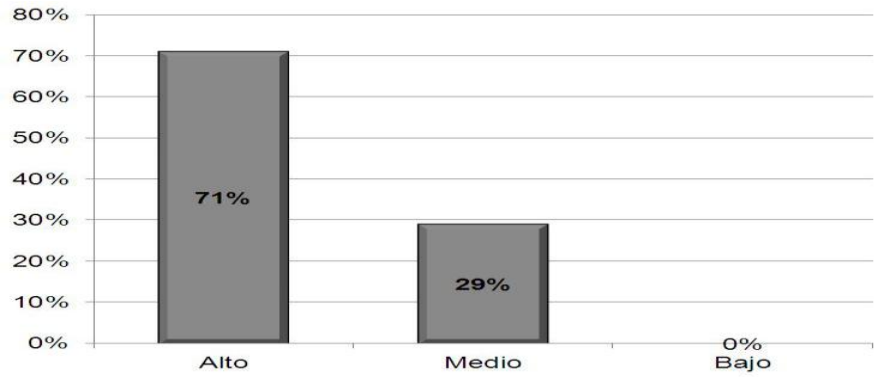


Figura No20. Relevancia otorgada a la investigación.

En la pregunta de si el mecanismo de extensión que se seleccionó para proporcionarle solución al problema planteado, es el más indicado, el 86% respondió que si, sin embargo el 14%, especificó que cualquiera de los dos descritos pudo haber resuelto dicho problema.

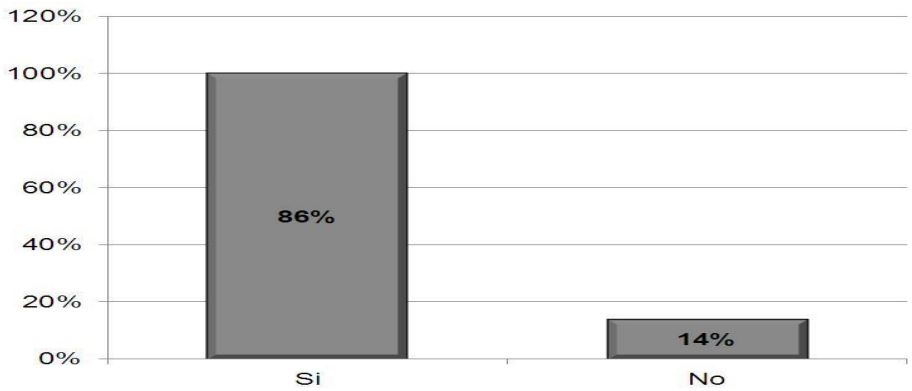


Figura No21. Mecanismo de extensión bien seleccionado.

En la pregunta de si es necesaria la implantación del perfil definido para propiciar el entendimiento y la comunicación entre los grupos de Análisis, Arquitectura y Desarrollo en el Centro de Tecnología y Gestión de Datos, el 86% respondió que si, mientras que el 14% respondió que no es necesaria.

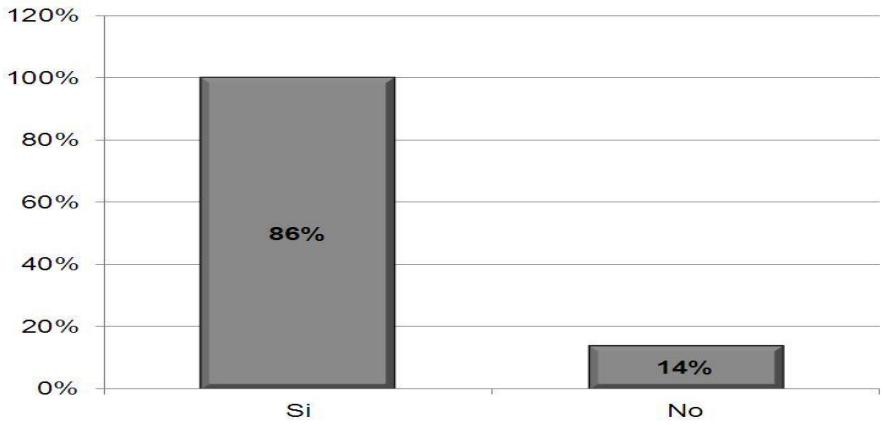


Figura No22. Necesidad de la implantación del Perfil.

En la pregunta de si el perfil sirve de directriz a la modelación para las vistas lógica y de implementación, el 100% respondió que si.

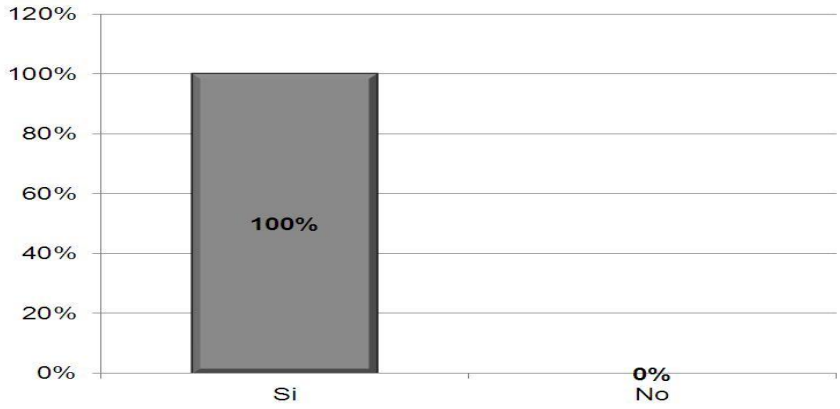


Figura No23. Sirve de directriz a la modelación para las vistas lógica y de implementación.

En la pregunta de si el uso del perfil reportará beneficios al formalizar elementos del modelado durante la etapa de análisis y diseño permitiendo su traducción precisa en implementación, el 100% respondió que si.

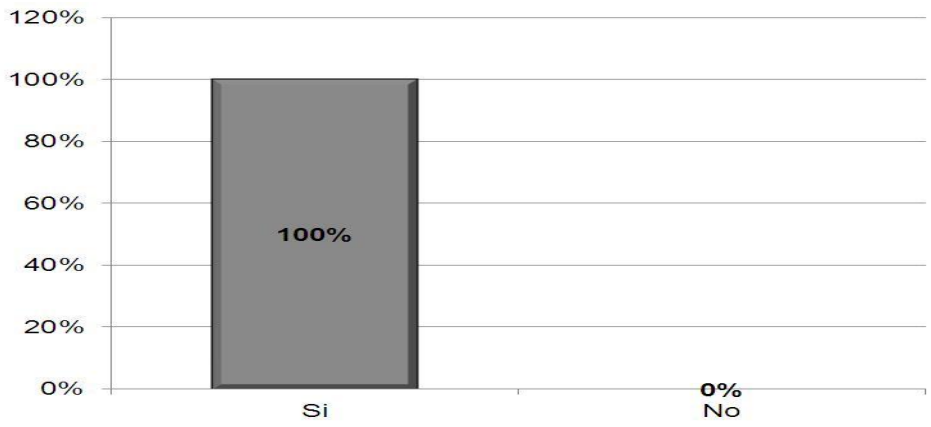


Figura No24. Beneficios al formalizar elementos del modelado.

En la pregunta de si el perfil de UML es entendible y está bien elaborado, el 100% respondió que si.

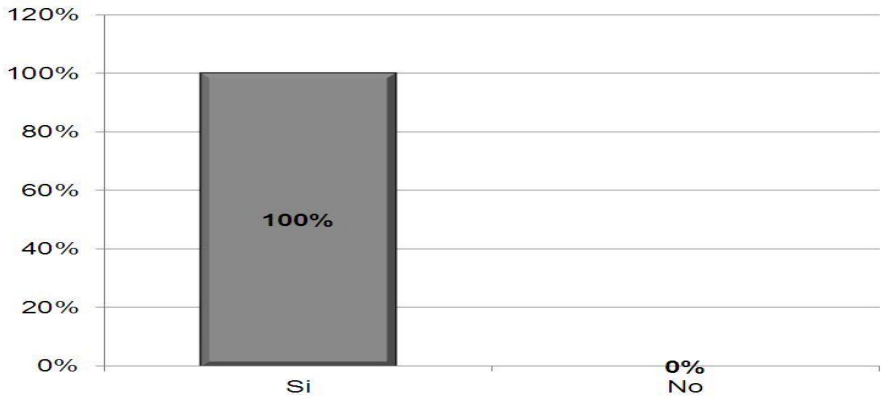


Figura No25. Perfil bien elaborado.

En la pregunta de si la propuesta es tan compleja como para que no se use, el 100% respondió que no.

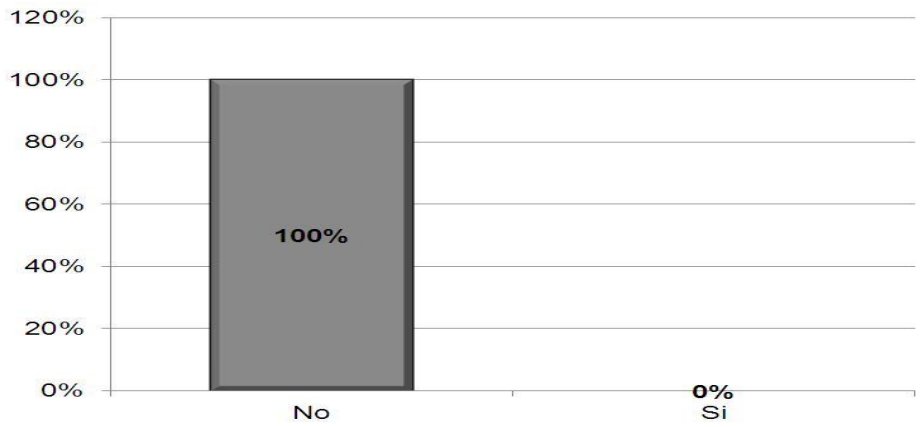


Figura No26. Propuesta del perfil compleja.

En la pregunta, qué grado de completitud se le confiere a esta propuesta de perfil, el 72% respondió que alta, mientras que el 28% respondió que media.

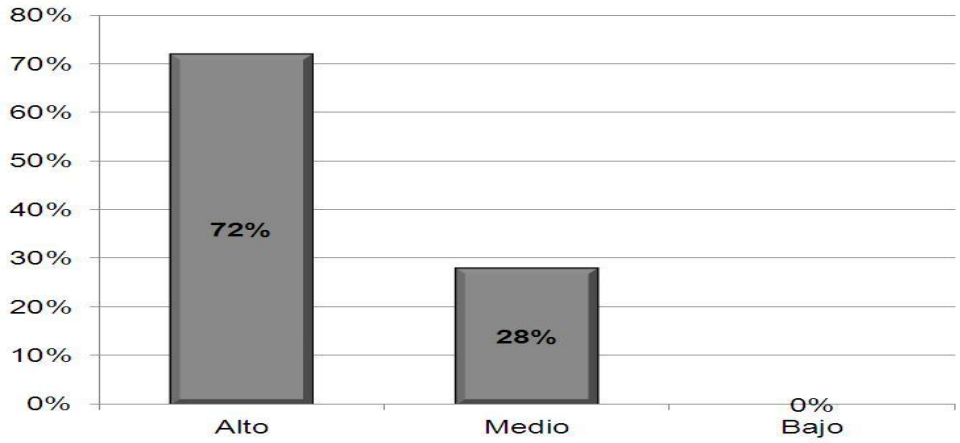


Figura No27. Grado de completitud.

En la pregunta de si se pueden presentar dificultades durante la aplicación del perfil definido, el 42% respondió que si, mientras que el 58% respondió que no.

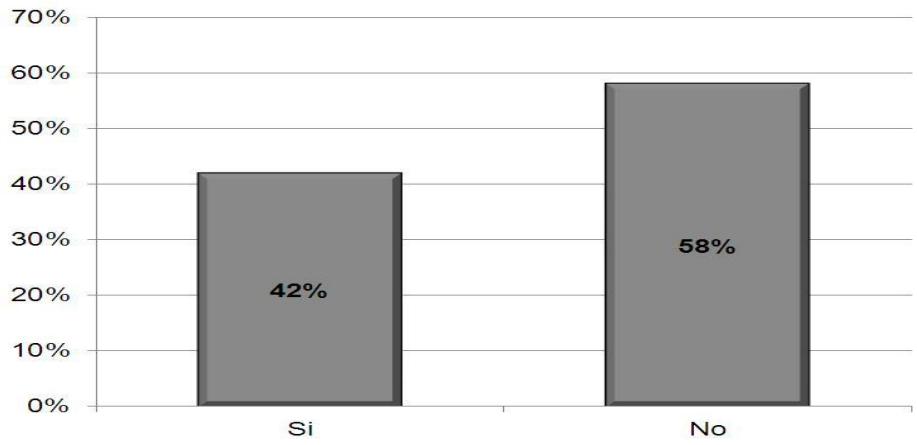


Figura No28. Dificultades durante la aplicación.

Algunas de las dificultades identificadas por le panel de expertos fueron:

- ✓ La reinscripción de estándares.
- ✓ La capacitación a los miembros del proyecto, ya que requiere del dominio de los patrones y tecnologías soportados por el perfil por parte de los analistas.
- ✓ La resistencia al cambio.
- ✓ La escasa familiarización con los estereotipos especificados.
- ✓ La asimilación correcta del uso del perfil por parte de los miembros del equipo.

En la pregunta de si considera que están correctamente identificados los patrones de interfaz de usuario de los sistemas de gestión que están presentes en el modelo de análisis, el 100% de los encuestados respondió que si.

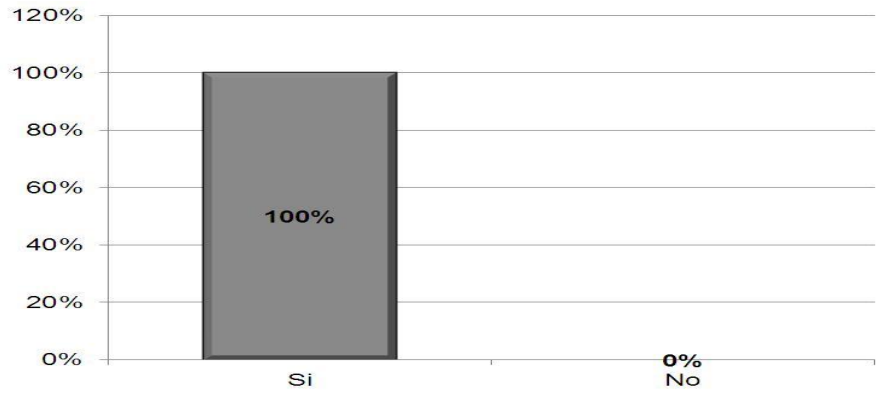


Figura No29. Patrones de interfaz de usuario correctamente identificados.

En la pregunta de si considera que están correctamente formalizadas las prácticas arquitectónicas y los principios y patrones de diseño en el Modelo de Diseño, el 100% respondió que si.

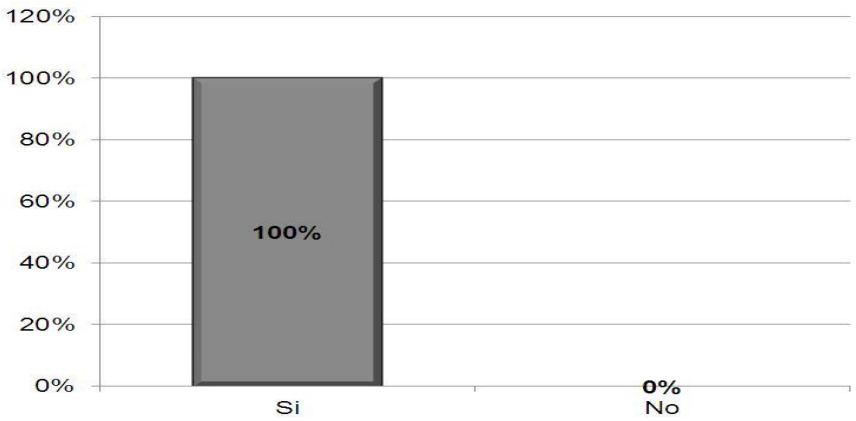


Figura No30. Prácticas arquitectónicas, principios y patrones de diseño en el Modelo de Diseño correctamente formalizados.

En la pregunta, exponga su opinión sobre la propuesta de forma general, las opiniones fueron satisfactorias por lo que la propuesta tuvo un 100% de aceptación.

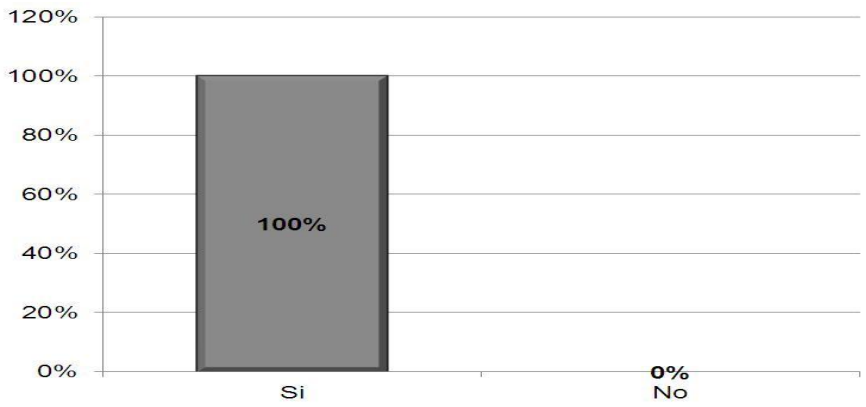


Figura No31. Opinión sobre la propuesta de forma general.

Las opiniones generales sobre la propuesta fueron las siguientes:

- ✓ La propuesta de perfil, resuelve un problema puntual que ayuda a la estandarización del proceso de desarrollo de software, reduciendo los tiempos para la confección de un producto y la calidad del mismo, ya que propicia una correcta comunicación y entendimiento entre los diferentes grupos.

- ✓ Proporciona un mecanismo de modelado que permitirá acelerar el proceso de desarrollo de la línea Soluciones Integrales en DATEC. Proporciona además una mejor comprensión por parte de los stakeholders.

- ✓ Propuesta interesante y adecuada al estilo y requerimientos de desarrollo de la línea Soluciones Integrales y para todo aquel que desarrolle con estas tecnologías.

- ✓ Muy acertada la propuesta pues estandariza y brinda una identidad a los productos del centro, desde el punto de vista del lenguaje UML y favorece a la comunicación entre los grupos de trabajo.

- ✓ Brinda un aporte significativo al sensible tema de la comunicación entre las distintas fases y roles implicados en el desarrollo de aplicaciones Informáticas. La propuesta es necesaria y simple de usar, además de que es muy atractiva ya que define un conjunto de plantillas para el modelo de análisis, diseño e implementación para la línea Soluciones Integrales, que formalizan elementos del análisis y el diseño y permiten la traducción precisa en implementación.

- ✓ La implantación de este perfil acarreará beneficios en el desarrollo de la línea. El mismo posibilitará enfocar mejor el análisis y el diseño a las particularidades de la línea de desarrollo, permitiendo mejorar y facilitar la comunicación del equipo de desarrollo, además propone un perfil de investigación, en el que se puede investigar más para el beneficio, no solo de la línea sino de toda la facultad.

- ✓ La propuesta ha analizado de manera profunda las características y potencialidades de los perfiles UML, obteniendo un vocabulario capaz de lograr una comunicación fluida y libre de ambigüedades.

Conclusiones parciales.

En este capítulo se manejaron las gestiones para la selección del panel de expertos. La buena clasificación de los mismos garantiza la calidad de las opiniones expresadas. Se coleccionaron datos cualitativos y cuantitativos que garantizaron la validez de la propuesta y la verificación de la misma, diseñada mediante encuestas a expertos tuvo un resultado exitoso ya que la mayoría de los especialistas emitieron una valoración satisfactoria, lo cual implica una buena admisión de la misma.

Conclusiones

Una vez concluida la investigación se arriba a las siguientes conclusiones:

- Con el estudio de los principales elementos teóricos, se determinó que los perfiles UML, constituyen el mecanismo que soluciona la problemática de esta investigación, pues define una serie de recursos para extender y adaptar las metaclases de un metamodelo cualquiera a las necesidades concretas de un dominio de aplicación. Este mecanismo no pretende modificar la semántica de UML, sino sólo particularizar algunos de sus conceptos, además pueden integrarse conceptos de la teoría y metodología de la línea Soluciones Integrales con los mecanismos de extensión que ofrece UML 2.0. También se observó que el modelado de aplicaciones tiene cada vez mayor protagonismo frente a la codificación, y el papel de los perfiles UML es fundamental en la definición de modelos, por lo que el perfil se acopla con facilidad a MDA y MDD, lo que ofrece la posibilidad de generar de manera automática la traducción precisa en implementación a partir del modelo conceptual.

- A partir de las necesidades de la línea Soluciones Integrales, se llevó a cabo la descripción del perfil de UML para representar las decisiones de análisis y diseño de los productos que la conforman. En esta descripción, se confeccionaron plantillas las cuales están contenidas por el Modelo de Dominio para el Análisis, Diseño e Implementación, la especificación del perfil y de las entidades que integran los distintos modelos de dominio. La extensión contiene además los necesarios estereotipos, valores etiquetados y restricciones para modelar de manera formal los elementos de análisis, arquitectura y desarrollo de la línea Soluciones Integrales de DATEC.

- A partir del estudio del método Delphi se manejaron las gestiones para la selección de un panel de expertos. La buena clasificación de los mismos garantizó la calidad de las opiniones expresadas. Se coleccionaron datos cualitativos y cuantitativos que garantizaron la validez de la propuesta y la verificación de la misma, diseñada mediante encuestas a expertos tuvo un resultado exitoso ya que la mayoría de los especialistas emitieron una valoración satisfactoria, lo cual implica una buena admisión de la propuesta.

Recomendaciones

Para este trabajo se recomienda lo siguiente:

- Aplicar de forma práctica la propuesta de perfil de UML en los proyectos de la Línea Soluciones Integrales del Centro de Tecnología de Gestión de Datos y realizar un análisis de los resultados.

Bibliografía

1. Montilva, Jonás A. Desarrollo de Software Basado en Líneas de productos de Software. Universidad de Los Andes. Facultad de Ingeniería. Departamento de Computación. Mérida, Venezuela. Noviembre, 2003.

2. Bravo, M. de L. y Arrieta, J. El método Delphi. Su implementación en una estrategia didáctica. Revista Iberoamericana de Educación. Cuba-España. 1-10.

3. Vale, Anelis Pereira. Un perfil UML 2.0 para el modelado de planes del entrenamiento deportivo. Revista Avanzada Científica Vol. 14 No. 1 Año 2011 Matanzas, Cuba.

4. Bautista, Osire. Lenguaje de Modelado Unificado (UML). Julio, 2009, REPÚBLICA BOLIVARIANA DE VENEZUELA, [Online] Disponible en :

   http://www.buenastareas.com/ensayos/Lenguaje-De-Modelado-Unificado/194570.html

5. López, José Roberto. Conceptos Básicos De UML. [Online] febrero, 2010. Disponible en:

   http://www.buenastareas.com/temas/caracteristicas-e-importancia-del-lenguaje-modelado-unificado/220

6. Booch G., Rumbaugh J., Jacobson I. El Lenguaje Unificado de Modelado. Addison-Wesley, Madrid, 1999.

7. Hernández, Enrique. El Lenguaje Unificado de Modelado (UML), España.

8. G. Booch, J. Rumbaugh y I. Jacobson, "El Lenguaje Unificado de Modelado", Addison Wesley, 1999

9. Ajila, Samuel A. Change Management: Modeling Software Product Lines Evolution. Carleton University, Canada.

10. Fuentes, Lidia y Vallecillo, Antonio. Una Introducción a los Perfiles UML. Depto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga. Málaga, España.

11. Anacleto, Valerio Adrián. [Online]. Disponible en http://www.ebooktoyou.net/ebook/uml-2.0-pdf.php

12. Alva, María Elena. XMI: XML Metadata Interchange. Universidad de Oviedo.

13. Conejero, Hernández, Pedrero. Definición de un Perfil UML para el Aspecto de Notificación en Entornos Distribuidos CORBA. Quercus Software Engineering Group, Departamento de Informática, Universidad de Extremadura, España.

14. Meta Object Facility (MOF) Core Specification. OMG Available Specification, Versión 2.0, Enero, 2006.

15. Pastrana, José L.; Pimentel, Ernesto; Katrib Miguel. Un Perfil UML para la definición de Componentes Inteligentes. ETSI Informática, Universidad de Málaga, Campus de teatinos s/n, Málaga, España. Facultad de Matemática y Computación, Universidad de La Habana, San Lázaro y L, Vedado, Ciudad de la Habana, Cuba.

16. Johnston, Simón. Perfil de UML 2.0 para servicios de software.

17. Alonso, Evelyn Menéndez. Herramientas CASE proceso desarrollo Software. [Online]. Disponible en http://www.plusformacion.com/Recursos/r/Herramientas-CASE-para-proceso-desarrollo-Software#herramiena

18. OMG Unified Modeling Language (OMG UML), Superstructure. Versión 2.2. Febrero 2009.[Online]. Disponible en http://www.omg.org/spec/UML/20080501/Superstructure.cmof

19. CSS Plantillas. [Online]. Disponible en www.cssplantillas.es.tl

20. Bas, Enric. Prospectiva. Como Usar El Pensamiento Sobre El Futuro. 2002.

21. Miranda, Hugo. Administración de operaciones. [Online]. Disponible en

    http://adminoperaciones.blogspot.com/2008_04_01_archive.html

22. GTIC: Grupo de Tecnología de la Información y las Comunicaciones. Método Delphi. [Online].2010. Disponible en: http://www.gtic.ssr.upm.es/encuestas/delphi.htm#A1.1.2

23. Oñate Martínez, N., Ramos Morales, L. y Díaz Armesto, A. [1990]. Utilización del Método Delphi en la pronosticación: una experiencia inicial. La Habana: Instituto de Investigaciones Económicas de la Junta Central de Planificación.

24. Perito ó experto. [online]. Disponible en http://es.scribd.com/doc/51611781/Perito-o-experto

25. OMG Unified Modeling Language**TM** (OMG UML), Infrastructure. Versión 2.2. Febrero, 2009.

26. Crnkovic, Ivica y Larsson, Magnus. Building Reliable Component-Based Software Systems. 2002, Boston. London.

27. Brown, Marcelo Pardo. METAMODELOS. 2008.

28. Potencier, Fabien y Zaninotto, François. Symfony 1.2, la guía definitiva, Diciembre, 2008 [Online] Disponible en http://www.librosweb.es/symfony_1_2

Referencia bibliográfica

1. Montilva, Jonás A. Desarrollo de Software Basado en Líneas de productos de Software. Universidad de Los Andes. Facultad de Ingeniería. Departamento de Computación. Mérida, Venezuela. Noviembre, 2003.

2. Bautista, Osire. Lenguaje de Modelado Unificado (UML).

3. López, José Roberto. Conceptos Básicos De UML. [Online] febrero, 2010. Disponible en: http://www.buenastareas.com/temas/caracteristicas-e-importancia-del-lenguaje-modelado-unificado/220

4. Booch G., Rumbaugh J., Jacobson I. El Lenguaje Unificado de Modelado. Addison-Wesley, Madrid, 1999.

5. Hernández, Enrique. El Lenguaje Unificado de Modelado (UML), España.

6. Fuentes, Lidia y Vallecillo, Antonio. Una Introducción a los Perfiles UML. Depto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga. Málaga, España.

7. Anacleto, Valerio Adrián. [Online]. Disponible en http://www.ebooktoyou.net/ebook/uml-2.0-pdf.php

8. Alva, María Elena. XMI: XML Metadata Interchange. Universidad de Oviedo.

9. Conejero, Hernández, Pedrero. Definición de un Perfil UML para el Aspecto de Notificación en Entornos Distribuidos CORBA. Quercus Software Engineering Group, Departamento de Informática, Universidad de Extremadura, España.

10. Pastrana, José L.; Pimentel, Ernesto; Katrib Miguel. Un Perfil UML para la definición de Componentes Inteligentes. ETSI Informática, Universidad de Málaga, Campus de teatinos s/n, Málaga, España. Facultad de Matemática y Computación, Universidad de La Habana, San Lázaro y L, Vedado, Ciudad de la Habana, Cuba.

11. Johnston, Simón. Perfil de UML 2.0 para servicios de software.

12. Alonso, Evelyn Menéndez. Herramientas CASE proceso desarrollo Software. [Online]. Disponible en http://www.plusformacion.com/Recursos/r/Herramientas-CASE-para-proceso-desarrollo-Software#herramiena

13. OMG Unified Modeling Language (OMG UML), Superstructure. Versión 2.2. Febrero 2009,[Online]. Disponible en http://www.omg.org/spec/UML/20080501/Superstructure.cmof

14. CSS Plantillas. [Online]. Disponible en www.cssplantillas.es.tl

15. Bas, Enric. Prospectiva. Como Usar El Pensamiento Sobre El Futuro. 2002

16. Miranda, Hugo. Administración de operaciones. [Online].

    Disponible en http://adminoperaciones.blogspot.com/2008_04_01_archive.html

17. GTIC: Grupo de Tecnología de la Información y las Comunicaciones. Método Delphi. [Online].2010. Disponible en: http://www.gtic.ssr.upm.es/encuestas/delphi.htm#A1.1.2

18. Oñate Martínez, N., Ramos Morales, L. y Díaz Armesto, A. [1990]. Utilización del Método Delphi en la pronosticación: una experiencia inicial. La Habana: Instituto de Investigaciones Económicas de la Junta Central de Planificación.

19. Perito ó experto. [online]. Disponible en http://es.scribd.com/doc/51611781/Perito-o-experto