UNIVERSIDAD CATÓLICA ANDRÉS BELLO
VICERRECTORADO ACADÉMICO
ESTUDIOS DE POSTGRADO
ÁREA DE INGENIERÍA
Postgrado en Sistemas de Información

APLICACIÓN DE LA INGENIERIA INVERSA DE SOFTWARE SOBRE EL SISTEMA DE APROVISIONAMIENTO DE LA RED BANDA ANCHA DE CANTV

Presentado por:
Janeth Coromoto PEREZ ANGEL

Como un requisito parcial para obtener el titulo de
Especialista en Sistemas de Información

Profesor Guía:

Paola SAPUTELLI

Caracas, Octubre 2007

DEDICATORIA

En la vida son muchas las personas que se ponen en nuestro camino para ayudarnos a crecer e inculcarnos las cosas buenas, pero antes que nada no estaríamos en donde nos encontramos en estos momentos si no fuera gracias a Dios, en lo personal él es el ser que me da fuerzas para seguir adelante, en cada paso de este caminar llamado “Vida”.

Claro no puede faltar el apoyo moral e incondicional de mis padres y hermanos, por todos los esfuerzos para ayudarme a llegar hasta donde he querido, sus enseñanzas vida, su amor invaluable y su paciencia, por eso este nuevo logro se los dedico a ustedes.

A mi Amor, por reafirmarme que las metas hay que comenzarlas y concluirlas, que a veces no son sencillas o simples las etapas para lograrlo, pero que siempre es posible si uno se lo propone, para así lograr lo que uno quiere en la vida, también por estar allí en cada uno de mis pasos, por ser de ahora en adelante mi compañero amado en este sueño que hemos construido juntos.

No podía faltar mis Amigos de carrera de esta especialización, Daniel y Carla, aunque no nos vamos a poder graduar juntos y nuestros trabajos y obligaciones no nos permiten compartir tanto como lo hacíamos, les dedico este trabajo que es el fruto de la meta que nos propusimos al conocernos o reencontrarnos y que al fin de mi parte llega a su fin.

AGRADECIMIENTOS

Hoy al finalizar esta etapa propuesta, agradezco a la Universidad Católica Andrés Bello por haberme ofrecido una oportunidad de hacer este estudio de cuarto nivel en sus instalaciones, a todos los profesores que aquí laboran por brindarme sus enseñanzas y colaboración en todo momento.

Le doy gracias a CANTV, la empresa de la que formo parte desde hace mucho tiempo, que me a brindado un sin fin de enseñanzas y en particular a mi jefe Ing. Alvaro Carvajal y a mis compañeros del Departamento Daniel, Andrea y Manuel por darme la oportunidad de tomar en estudio un proyecto interno para el desarrollo de este trabajo.

A la Lic. Paola Saputelli, compañera de trabajo y amiga, por darme su apoyo sin dudar en la tutoría de este Trabajo Especial de Grado, por su ayuda incondicional y por su gran valor personal y profesional que la hacen destacar siempre, espero haber cumplido con tus expectativas.

A los diversos compañeros de clases, los cuales a pesar de ser de carreras profesionales diversas siempre existió unión y solidaridad, lo cual nos permitió ser un buen grupo y lograr todos esta meta que nos propusimos al empezar esta especialización, en particular a Daniel Acuña, Carla Iglesias, Juan Carlos Palacios y Mery Orellana.

Por ultimo doy gracias nuevamente a Dios, a mis padres y hermanos y a mi Amor por estar apoyándome incondicionalmente siempre en cada unas de mis metas y proyectos de vida.

RESUMEN

Este trabajo de grado tiene como objetivo primordial la aplicación de la Ingeniería Inversa de Software como solución a la obtención de las especificaciones de diseño de software partiendo del código fuente de un software de documentación escasa o inexistente, generando como resultado un documento con las abstracciones de procesamientos, datos e interfaz, las cuales, fueron generadas en un estándar de modelado universal para ser entendido por cualquier experto del área, basadas en el uso de una herramienta CASE (Ingeniería de Software Asistida por Computadora) gratuita.

El ámbito de trabajo es la Corporación CANTV específicamente en la Coordinación de Aprovisionamiento de Servicios, basados en uno de los software críticos bajo su responsabilidad y que es de vital importancia para la empresa, el cual, es el sistema de aprovisionamiento de la red de banda ancha, que tiene como propósito la configuración de los equipos de red claves para prestar el servicio de banda ancha y la auditoria de todos los clientes configurados en esta plataforma para velar por la integridad del servicio prestado.

En el aspecto conceptual, se hace referencia a todos los conceptos que se consideraron esenciales para comprender lo desarrollado en este trabajo, a nivel de procedimiento que fue definido para el desarrollo de este trabajo esta orientado a la definición y desarrollo de dos variables claves, una de ellas es el Modelado de Sistemas de Software, la cual, fue llevada a cabo con la aplicación del Lenguaje Unificado de Modelado (UML, Unified Modeling Lenguaje) como base práctica para mostrar las diversas abstracciones obtenidas de la Ingeniería Inversa, la otra es de Sistemas y Confidencialidad, en la cual se plantea los aspectos que van a ser tocados del sistema, limitaciones por parte de la empresa y la confidencialidad de la información dada en este trabajo. Por la naturaleza a la que se orientó este trabajo se considera un proyecto factible y a nivel de procedimiento formal se siguieron las premisas y definiciones propuestas para la aplicación de la Ingeniería Inversa.

El enfoque final de la aplicación de la Ingeniería Inversa para el sistema propuesto, esta expuesto básicamente en la reestructuración del código fuente y la obtención de los tres niveles de abstracción requeridos, los cuales son: procesamiento, datos e interfaz, a su vez por cada una de estas abstracciones se encuentra el desarrollo de los diversos diagramas UML que aplican en cada caso.

Como conclusión final y general se tiene que la aplicación de la Ingeniería Inversa no es de amplio conocimiento general, pero los resultados obtenidos con ella demuestran la fuerza e importancia que es para el ámbito de desarrollo de software, de igual manera el conocimiento de estándares de modelado de sistemas como UML y el uso de herramientas CASE como bases de software para realizarlos. De igual manera, lo importante del manejo de documentación en los desarrollos de sistema que permite retrasmitir conocimiento, realizar estimaciones efectivas y manejar reingeniería o cambios por nuevos requerimientos o continuidad de las aplicaciones.

Con este trabajo finalmente se logro solventar las necesidades planteadas por una empresa y dar una pauta para formalizar e introducir estándares de especificaciones de software ya desarrollados y los nuevos por realizar, así como hacer conciencia de la importancia del manejo de documentación de software.

INTRODUCCIÓN

Como cualquier sistema, un sistema de información es parte de un suprasistema que lo contiene, esto es, una organización o empresa. Si se adopta un enfoque de "caja negra", los sistemas de información reciben los datos de entrada, los procesan y producen la salida. Todo el proceso se lleva a cabo en un entorno formado por personas u otras instituciones que ejercen su influencia sobre la organización.

Desde una perspectiva de negocios, se puede definir a los sistemas de información como: "una solución administrativa y organizacional, basada en la tecnología de la información, a un desafío planteado por el entorno" (Laudon y Laudon, 1997). Esta definición enfatiza la naturaleza administrativa y organizacional de los sistemas de información.

Se puede acotar entonces que el empleo de sistemas de información automatizados que sirvan de apoyo a tareas operativas y regulares ha sido una práctica constante, los cuales permiten a las empresas usar productos ofertados por el mercado para sus soluciones o crear desarrollos propios que se ajusten a sus necesidades especificas y en muchos casos lograr obtener un ahorro significativo en soluciones de software y en efectividad de la operación. Nuestro país no escapa de esta realidad, ya que se ha experimentado un crecimiento significativo de este comportamiento, así como una constante dinámica de cambio.

Un elemento esencial, en cualquier sistema de información, es seguir una ingeniería de software apropiada que permita realizar un optimo desarrollo de software, logrando obtener un buen análisis, diseño y desarrollo, arrojando como producto final un sistema que cumpla con las necesidades para lo que fue creado, así como que exista la debida documentación del mismo, la cual permite la correcta implantación, operación y mantenimiento cuando sea necesario, sin depender directamente del mismo grupo de proyecto inicial.

En las organizaciones existe una continua dinámica de personal y de actividades debido a como se ha venido desarrollado el área de sistemas, esto hace que las empresas deban tomar sus provisiones y no enlazar los desarrollos y proyectos a un grupo de personas sin ser capaces de continuar los mismos con el mismo personal, esto solo es posible con el uso de metodologías universales para el desarrollo de software.

Como respuesta a esta necesidad de poseer especificaciones de software universales, los investigadores Booch, Rumbaugh y Jacobson (Rumbaugh, J.; Jacobson I. et al., 1999b) unieron sus trabajos en la materia, dando como resultado la primera versión del Lenguaje Unificado de Modelado (UML, Unified Modeling Lenguaje). Posteriormente este trabajo fue enriquecido con aportaciones de un gran número de colaboradores que se unieron al grupo inicial, logrando la adopción de UML como un estándar de OMG (Object Manamengt Group) en Noviembre de 1997 (OMG, 2005), a partir de esta fecha se han venido desarrollando diversas herramientas de Ingeniería del Software Asistida por Computador (CASE, Computer Aided Software Systems Engineering) que dan herramientas graficas de fácil uso para implementar este lenguaje estandarizado.

Todo sistema de información se construye para solucionar problemas presentes cuando se decide su construcción. Un cambio en las condiciones del entorno organizacional puede producir un cambio en los objetivos de la organización y, en consecuencia, las reglas de la organización también se modifican. Muchos sistemas en funcionamiento, deben someterse a un mantenimiento apremiante, constante y, normalmente, indocumentado, que desemboca en una degradación de la aplicación y, por ende, en un servicio deficiente para el usuario.

Las necesidades de evolución y adaptación a los nuevos requerimientos tecnológicos y de negocios empujan a los sistemas a una nueva situación que no se puede alcanzar con el mantenimiento clásico. En una primera aproximación, la solución a estos problemas se puede lograr por uno de dos caminos: un *nuevo*

desarrollo que incorpore las nuevas tecnologías y funcionalidades, o la aplicación heredado (Rodríguez, A. et al., 2001).

Por lo antes mencionado, la comunidad científica involucrada con el desarrollo de software en los últimos años se ha visto en la necesidad de reconocer la importancia que tiene el mantenimiento de software, sobretodo en el ámbito económico, con algunos estudios realizados se a buscando la forma de llamar la atención de la etapa de mantenimiento en el ciclo de vida de un producto de software, ya que en este ciclo, esta etapa se a considerado que es el recurso mas consumible (Piattini, M. et al. 2000).

De los trabajos realizados por parte de esta comunidad científica han surgido iniciativas, reuniones, publicaciones, etc., dedicadas a abordar este asunto. Dentro de esta propuesta se incluyen el desarrollo de mejores métodos y técnicas de ingeniería inversa y reingeniería (Piattini, M. et al. 2000).

La ingeniería inversa y, en particular, la recuperación de la arquitectura, ayuda a la extracción de representaciones de más alto nivel de sistemas heredados, y ofrece soporte a los ingenieros para la evaluación, el mantenimiento y el desarrollo de sistemas software a gran escala. A fin de producir estas vistas arquitectónicas, las herramientas de ingeniería inversa son capaces de procesar distintos tipos de artefactos software disponibles, tales y como pueden ser el código fuente, escenarios de funcionamiento, documentación, información de dominio y conocimiento de los expertos (Chikofsky, E. et al. 1990).

Las técnicas de ingeniería inversa son utilizadas para la comprensión de la estructura y el comportamiento de un sistema software del que no se dispone documentación, o que en caso de poseerla está obsoleta o es insuficiente (Briand, L. et al. 2003). En la ingeniería inversa, no se pretende exclusivamente recuperar especificaciones a más alto nivel del sistema heredado. Además de este tipo de información, también se pretende obtener documentación, información que resulta

importante a la hora de analizar este tipo de sistemas software y mejorar su comprensión y mantenimiento (Riva, C. et al. 2002).

Esta información, en muchas ocasiones, bien se ha perdido con el tiempo o incluso es inexistente, como ocurre con muchos sistemas actuales, que debido a ciclos de desarrollo rápidos merman el desarrollo de documentación asociada, lo que hace que su posterior etapa de mantenimiento sea más difícil y costosa (Di Lucca, G. et al. 2001).

Se puede reflejar de lo antes expuesto, un caso específico de nuestro entorno, es la CANTV (Compañía Anónima Nación de Teléfonos de Venezuela) una empresa de telecomunicaciones que realiza desarrollos de software propios para dar soluciones a necesidades específicas de su operación. Por su tamaño y diversidad de áreas existen ciertos desarrollos que no cumplen con las características antes mencionadas. Esto se evidencia en sistemas donde se han realizado análisis, diseño y desarrollo que no cuentan con su respectiva documentación, y si esta existe, no cumple con ningún estándar de desarrollo de software.

La Coordinación de Aprovisionamiento de Servicios de la Gerencia de Ingeniería, Desarrollo y Construcción TI/SI presenta esta necesidad, debido a que esta unidad maneja sistemas críticos para el negocio de la corporación y no cuentan con la documentación respectiva de sus procesos, la ausencia de este material conlleva a que algún cambio de estructura interna, movimiento de personal, estimación de nuevos requerimientos o modificaciones sobre el sistema sean tareas difíciles debido a la falta de información y la ausencia de una visión completa y precisa de los sistemas manejados. Es requerido como objetivo del departamento realizar la documentación de los procesos manejados basados en un estándar de desarrollo de software efectivo y preciso que permita el completo conocimiento de los componentes del sistema a través de la aplicación de la ingeniería inversa de software.

CAPÍTULO I – MARCO TEÓRICO

En este capitulo se hace un recorrido por los fundamentos de *Sistemas*, luego se da mas detalle de la *Ingeniería del Software* y los elementos que la componen (procesos, modelos, métodos y herramientas) a su vez se detalla cada uno de estos elementos de manera de conocer el conjunto de opciones disponibles para el desarrollo de software, una vez que se esta claro en estos conceptos se profundiza en lo que se conoce como la *Ingeniería Inversa de Software*, la cual es conocida por formar parte de los pasos para aplicar reingeniería de procesos, pero ella por si sola es un proceso consistente que puede ser aplicada para generar especificaciones de software partiendo de un código fuente y para culminar se plantea el conjunto de conceptos que abarca el *Lenguaje de Modelado Unificado*, conocido mundialmente como UML, el cual es el lenguaje de modelado estándar actualmente utilizado en el desarrollo de software y apoya directamente a el método de software orientado a objetos.

1.1 SISTEMAS

Se puede empezar por revisar este conjunto de definiciones ya desarrolladas, a continuación:

- Por Bestalanffy: "Un sistema es un conjunto de unidades en interrelación." (Bestalanffy, 1998).
- Por Saussure: "Sistema es una totalidad organizada, hecha de elementos solidarios que no pueden ser definidos más que los unos con relación a los otros en función de su lugar en esa totalidad." (Saussure, 1931).
- Por Bunge: Sistema Σ es una terna ordenada [C(Σ), E(Σ), S(Σ)] en la que:
  - C(Σ) (composición de Σ) representa el conjunto de partes de Σ.
  - E(Σ) (entorno o medio ambiente de Σ es el conjunto de aquellos elementos que, sin pertenecer a C(Σ), actúan sobre sus componentes o están sometidos a su influencia.
  - S(Σ) (estructura de Σ) es el conjunto de relaciones y vínculos de los elementos de C(Σ) entre sí o bien con los miembros del entorno E(Σ). (Bunge, 1979).

- Según la IEEE: "Sistema es un todo integrado, aunque compuesto de estructuras diversas, interactuantes y especializadas. Cualquier sistema tiene un número de objetivos, y los pesos asignados a cada uno de ellos pueden variar ampliamente de un sistema a otro. Un sistema ejecuta una función imposible de realizar por una cualquiera de las partes individuales. La complejidad de la combinación está implícita." (IEEE, 1997).

Resumiendo, de las definiciones se pueden extraer unos aspectos fundamentales del concepto sistema:

- La existencia de elementos diversos e interconectados.
- El carácter de unidad global del conjunto.
- La existencia de objetivos asociados al mismo.
- La integración del conjunto en un entorno.

Los sistemas están compuestos por los siguientes elementos: Los *componentes* del sistema, las *relaciones* entre ellos, que determinan la estructura del sistema, el *objetivo* del sistema, el *entorno* del sistema que es aquello que lo rodea, dentro del cual esta ubicado y los limites del sistema que es la frontera entre lo que es el sistema y lo que constituye el entorno. En muchos sistemas la salida influye en el estado del mismo y esto es conocido como retroalimentación. (Piattini, 1996).

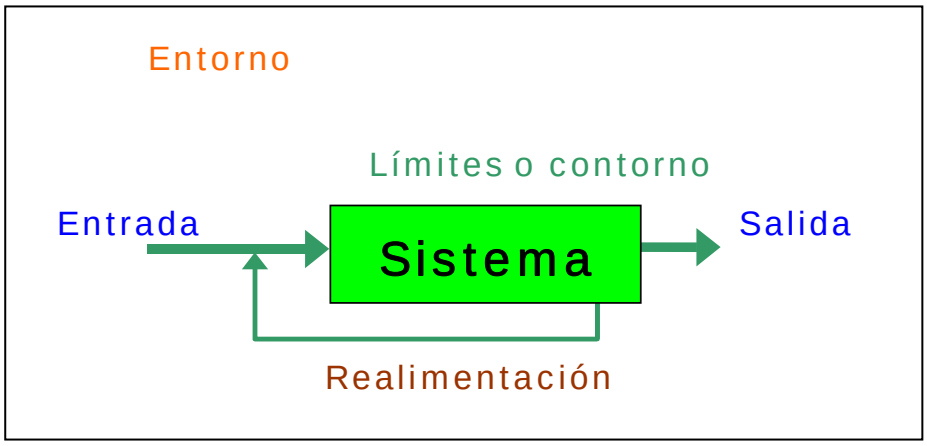

Figura 1. Concepto de sistema

Fuente: (Ros, 2005)

1.1.1 SISTEMAS DE INFORMACIÓN

Un sistema de información se puede definir como un conjunto de funciones o componentes interrelacionados que forman un todo, es decir, obtiene, procesa, almacena y distribuye información para apoyar la toma de decisiones y el control en una organización. Igualmente apoya la coordinación, análisis de problemas, visualización de aspectos complejos, entre otros.

Un sistema de información contiene información de sus procesos y su entorno. Como actividades básicas producen la información que se necesita: entrada, procesamiento y salida. La retroalimentación consiste en entradas devueltas para ser evaluadas y perfeccionadas.

Es un sistema que sirve para proporcionar la información necesaria a la organización o empresa, donde y cuando se necesite, esta información puede ser usada para las operaciones de dicha empresa y para las actividades de dirección y control correspondientes (decisiones) para desempeñar su actividad de acuerdo a su estrategia de negocio.

Un sistema de información se puede definir como el conjunto de funciones y procedimientos encaminados a la captación, desarrollo, recuperación, almacenamiento, etc., de información en el seno de una organización. (Wikipedia, 2005)

1.1.2 SISTEMAS BASADOS EN COMPUTADORAS

El sistema de información existe siempre, esté mecanizado o no. Ahora bien, el computador interviene para mejorar todos los factores que determinan el valor de un sistema de información (eficacia, bajo costo, calidad, etc.), o a veces incluso para hacerlos simplemente posibles. La complejidad del mundo empresarial actual hace que el computador sea hoy día un elemento prácticamente imprescindible para casi cualquier sistema de información, de cualquier empresa y organismo.

El computador puede participar de manera efectiva en casi todos los flujos de información que se producen en la empresa: información económica, laboral, de materias y de productos. Por estas razones, cuando se habla de sistemas de información en la empresa, se entiende a veces sistemas de información basados en computadora, cuando en realidad son dos conceptos distintos. (Piattini, 1996).

Los principales objetivos de los sistemas de información basados en computadora son:

- Simplificar o *realizar automáticamente* procesos que tradicionalmente se realizaban de forma manual: por ejemplo, pueden realizar los reiterativos cálculos contables sin errores en las operaciones y con una rapidez inigualable, o permiten el uso de métodos matemáticos inabordables sin estas herramientas, o controlan el proceso de fabricación, evitando errores y mejorando la velocidad y precisión de medida.
- Deben proporcionar información que *facilite la toma de decisiones*, dejando libertad plena a los usuarios. Se incrementa de esta manera la calidad de la función directiva, al apoyar cada decisión en una base sólida de información debidamente presentada y procesada. La información debe llegar siempre a aquellas personas para las que sea relevante, y a nadie más (pero tampoco a nadie menos). La información se debe proporcionar de manera concisa, evitando generar grandes listados imposibles de analizar. (Pressman, 2002)

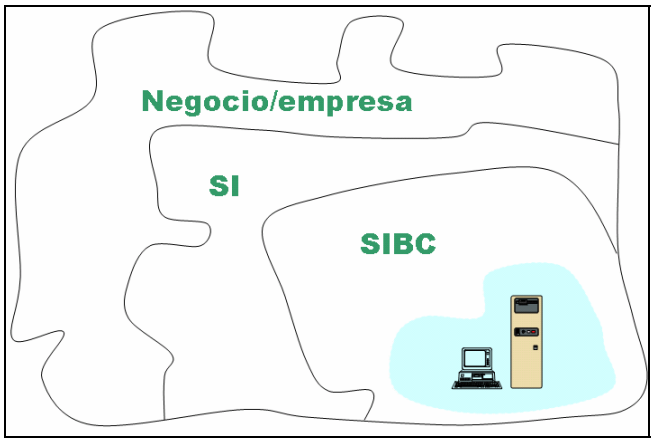

Figura 2. Organización, sistemas de información (SI) y SI basados en computadoras (SIBC)
Fuente: (Ros, 2005)

1.1.3 SISTEMAS DE APROVISIONAMIENTO DE SERVICIOS

No existe una definición especifica de este tipo de sistema, en base a la definición de sistema ya especificada anteriormente: “es un conjunto de funciones o componentes interrelacionados que forman un todo, es decir, obtiene, procesa, almacena y distribuye información para apoyar la toma de decisiones y el control en una organización. Igualmente apoya la coordinación, análisis de problemas, visualización de aspectos complejos, entre otros” (Wikipedia, 2005) y también definimos la palabra aprovisionamiento que según el diccionario de la real academia española se define como: “acción y efecto de aprovisionar”, al consultar el concepto de aprovisionar tenemos: “abastecer” y al buscar la definición de abastecer es: “Proveer de bastimentos, víveres u otras cosas necesarias”. (Real academia española, 2005).

Haciendo una unión de ambos conceptos, orientando los mismos al entorno de sistema de información, se puede decir, que son sistemas de información con el objetivo de ejecutar un conjunto de actividades o tareas en su mayoría operativas, las cuales pueden estar claramente definidas y automatizadas para aprovisionar una plataforma de servicio, adelantando así tareas rutinarias que mantengan la plataforma consistente y poder realizar transacciones de cara al cliente mas cortas y efectivas, logrando así hacer mas optimo los procesos que se manejan.

1.2 INGENIERÍA DEL SOFTWARE

Antes de dar cualquier concepto acerca de la ingeniería del software se debe empezar por definir que es un software de computadora, entre tantos escritos acerca del tema, podemos acotar esta definición a que el software de computadora es el producto que diseñan y construyen los ingenieros de software. Esto abarca programas que se ejecutan dentro de una computadora de cualquier tamaño y arquitectura, documentos que comprenden formularios virtuales o impresos y datos que combinan números y texto y también incluyen representaciones de información de audio, video e imágenes. (Bauer, F. 1972).

De lo antes comentado podemos descomponer esta definición en sus tres bases:

- *Instrucciones* que, cuando se ejecutan, proporcionan la funcionalidad deseada.
- *Estructuras de Datos* que facilitan a las instrucciones manipular adecuadamente la información.
- *Documentos* que describen el desarrollo, uso, instalación y mantenimiento de los programas

El software debe ser realizado por ingenieros de software, ya que la importancia del mismo es tal que afecta muy de cerca cualquier aspecto de nuestra vida y esta muy extendido en el comercio, cultura y en nuestras actividades cotidianas. Para construir un software de computadora debe ser como construir cualquier otro producto satisfactorio, aplicando procesos que conducen a un resultado de alta calidad que satisface las necesidades de la gente que usara el producto y para ello se deben aplicar un enfoque de la ingeniería de software.

Desde un ingeniero de software el producto obtenido son los programas, documentos y los datos que configuran al software, pero desde el punto de vista del usuario, el producto es la información resultante que hace de algún modo el mundo mejor a los usuarios.

Una vez que se esta claro en lo que se conoce por software de computadora, se puede entrar a revisar algunas definiciones de Ingeniería del Software, por diversos autores, entre las cuales se tienen:

- La Ingeniería del Software trata del establecimiento y uso de principios sólidos de ingeniería, orientados a obtener software económico que sea fiable y trabaje de manera eficiente en maquinas reales. (Bauer, 1972).
- Ingeniería del Software: (1) La aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, la operación y el mantenimiento del software; es decir, la aplicación de la ingeniería del software: (2) El estudio de enfoques como en (1). (IEEE, 1993).

- Es una disciplina que comprende todos los aspectos de la producción de software desde las etapas iniciales de la especificación del sistema, hasta el mantenimiento de este después que se utiliza. (Sommerville, 2002).

En esta última definición se tocan dos aspectos importantes, uno es referente a la *disciplina de ingeniería*: en donde los ingenieros hacen que las cosas funcionen. Aplican teorías, métodos y herramientas donde sea conveniente, pero las utiliza de forma selectiva y siempre tratando de descubrir soluciones a los problemas, aun cuando no existan teorías y métodos aplicables para resolverlos. Los ingenieros también tienen que trabajan con restricciones financieras y organizacionales, por lo que buscan soluciones teniendo en cuenta estas restricciones.

El otro punto de esta definición es que se tocan *todos los aspectos de la producción de software*: la ingeniería del software no solo comprende los procesos técnicos de desarrollo de software, sino también actividades como la administración de proyectos de software y el desarrollo de herramientas, métodos y teorías de apoyo a la producción de software. (Pressman, 2002).

Otro enfoque dado a la ingeniería del software es definirla como una tecnología estratificada, que se compone de un conjunto de capas que son: proceso, modelos, métodos, herramientas y todos sustentados en un enfoque de calidad, cada una de estas capas se especifica a continuación:

1.1.1 PROCESOS DE SOFTWARE

Los procesos de software se puede definir como un conjunto de actividades cuya meta es el desarrollo o evolución del software, entre las actividades fundamentales y comunes de cualquier proceso de este tipo tenemos: especificación, desarrollo, validación y evolución del software.

Los procesos de software son complejos y, como en todo proceso intelectual, se basa en el juicio humano. Se puede decir que no existe un proceso de software

"ideal", en las organizaciones existen muchos enfoques para mejorarlos, la mayoría están basados en la búsqueda de la estandarización, reduciendo con ella la diversidad de los procesos de la organización, también, con su aplicación trae consigo la mejora de la comunicación, reducir el tiempo de capacitación y hacer que la ayuda al proceso automatizado sea mas económica, esta estandarización también es el primer paso para introducir nuevos métodos, técnicas y buenas practicas de ingeniería del software.

1.2.2 MODELOS DE PROCESOS DE SOFTWARE

Los estándares establecen los diferentes procesos implicados a la hora de desarrollar y mantener un sistema desde que surge la idea o necesidad de desarrollar las aplicaciones hasta que éstas se retiran de producción. Sin embargo, ningún estándar impone un modelo de procesos concreto (modelo de ciclo de vida) ni cómo realizar las diferentes actividades incluidas en cada proceso, por lo que cada empresa deberá utilizar los métodos, técnicas y herramientas que considere oportuno.

Por su naturaleza, los modelos son simplificaciones; por lo tanto, un modelo de procesos del software es una simplificación o abstracción de un proceso real. Se puede definir un modelo de procesos del software como una representación abstracta de alto nivel de un proceso software.

Cada modelo es una descripción de un proceso software que se representa desde una perspectiva particular, por lo que solo provee información parcial acerca de ese proceso. Estos modelos se pueden categorizar a gran escala de dos maneras los no evolutivos (no maneja la iteración ni la naturaleza evolutiva del software) y los evolutivos (son iterativos y permite desarrollos de versiones de software mas completas), a continuación se describen brevemente los aspectos más relevantes de los modelos más conocidos.

1.2.2.1 MODELO LINEAL SECUENCIAL

Este modelo fue desarrollado entre 1960 – 1980, esta basado en el modelo en cascada de Winston Royce, por ello es conocido también como modelo de cascada o ciclo de vida clásico. Este modelo se caracteriza por no pasar de una etapa a otra sin que se hayan concluido las actividades de la etapa anterior, se considera que su mayor debilidad es el manejo del esquema de cambios, esto se debe a que la mayoría de las veces los usuarios no exponen todos sus requisitos al inicio del sistema y para que el cliente pueda ver un bosquejo real de lo que será el sistema debe tener paciencia, ya que se vera cuando ya se este concluyendo el mismo, las etapas de este modelos son: análisis (análisis y definición de requerimientos), diseño (diseño de sistemas y de software), código (implementación y prueba de unidades), prueba (integración y prueba del sistema) y mantenimiento (operación y mantenimiento).

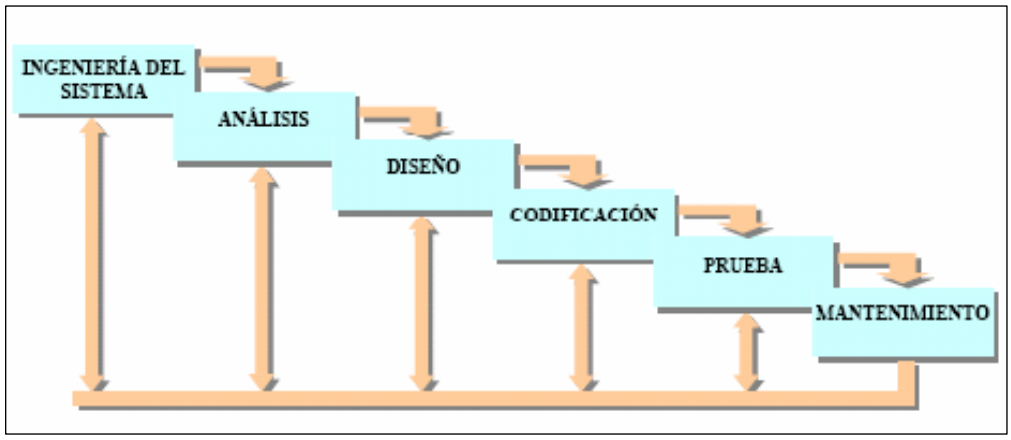

Figura 3. Modelo lineal secuencial o modelo en cascada

Fuente: (Pressman, 2002)

1.2.2.2 MODELO DE CONSTRUCCIÓN DE PROTOTIPOS

Este modelo permite trabajar con sistemas en donde el cliente no esta claro en los requisitos que requiere, es por ello, que el ingeniero de software se basa en el uso de maquetas (diseño rápido) para lograr capturar todas las necesidades del cliente. Este modelo se basa en la recolección de requisitos previamente con el cliente, basado en estas interacciones entre ambos se realiza un diseño rápido de lo que podría ser el sistema (prototipo) sin importar detalles de optimización sino

mas de funcionabilidad, para en progresivas interacciones con el cliente se logren recopilar todos los requisitos expuestos en la maqueta y una vez aprobada la maqueta por el cliente se procede a su desarrollo.

Este modelo no modifica a gran escala lo planteado por el modelo lineal secuencial, sino que refuerza la debilidad que posee esté, que es el manejo de cambios, ya que se reduce el riesgo de construir productos que no satisfagan las necesidades de los usuarios, lo que implica directamente una reducción de costo y un aumento en la probabilidad de éxito. Este modelo se caracteriza por ser barato y de rápida elaboración, en la fase de construcción se basa en la interfaz de usuario. Una vez culminado el prototipo o maqueta con los requerimientos del cliente es el momento donde se comienza a realizar el producto de ingeniería del software como tal con un alto grado de aprobación del usuario final.

1.2.2.3 MODELO DE DESARROLLO RÁPIDO DE APLICACIONES

El modelo de desarrollo rápido de aplicaciones (RAD, Rapid Application Development) se obtiene combinando el prototipado con las técnicas de cuarta generación. Estas herramientas incrementan la potencia de las herramientas de diseño de prototipos, proporcionando todo lo necesario para construir un prototipo y convertirlo en una aplicación funcional. Se construyen aplicaciones, con herramientas RAD, empezando por el diseño de la interfaz de usuario. Se ensamblan componentes, como botones, menús, ventanas, etc. centrándonos más en lo que hace el programa que en cómo lo hace, aplica así un ciclo de vida extremadamente corto. Se muestra la aplicación al usuario, se recoge su opinión y se introducen las modificaciones en la aplicación. Se repite el proceso hasta que el usuario final está satisfecho.

1.2.2.4 MODELO EN ESPIRAL

En este modelo, que proporciona las ventajas del modelo de ciclo de vida en cascada y el prototipado, el software se va desarrollando en una serie de versiones incrementales. A pesar de ser un modelo relativamente reciente (Boehm, 1988) y de no haber sido utilizado ampliamente, es un modelo realista para el desarrollo de sistemas software complejos.

El modelo espiral para la ingeniería de software ha sido desarrollado para cubrir las mejores características tanto del ciclo de vida clásico, como de la creación de prototipos, añadiendo al mismo tiempo un nuevo elemento: el análisis de riesgo.

El modelo representado mediante la espiral define una serie de actividades o regiones de tareas, generalmente se distinguen entre tres y seis:

1. Comunicación con el cliente: tareas requeridas para establecer la comunicación entre el desarrollador y el cliente.
2. Planificación: tareas para definir recursos, tiempo y otras informaciones relacionadas con el proyecto.
3. Análisis de riesgos:
    a. Identificar riesgos (presupuesto, retraso en el tiempo,...).
    b. Estimación de riesgos: para cada riesgo identificarla probabilidad de que ocurra y las consecuencias.
    c. Evaluación de riesgos: establecer niveles de referencia que permitan, llegado el caso, interrumpir el proyecto. Por ejemplo, si superamos un determinado costo o duración.
    d. Gestión de riesgos: supervisar el desarrollo del proyecto de forma que se detecten los riesgos tan pronto como aparezcan y minimizar daños.
4. Ingeniería: tareas para desarrollar la aplicación o prototipo.
5. Construcción y adaptación: tareas para construir, probar e instalar el sistema o prototipo y proporcionar soporte al usuario (documentación).
6. Evaluación del cliente: tareas para obtener la reacción del cliente al mostrarle el software o prototipo desarrollado.

En cada región existen una serie de tareas que se adaptan a las características del proyecto. Para proyectos pequeños el número de tareas y su formalidad será bajo, para grandes será mayor. Cuando empieza este proceso evolutivo, el equipo de ingeniería del software gira alrededor de la espiral en la dirección de las agujas del reloj, comenzando por el centro. En la primera iteración se definen los

requisitos del sistema y se realiza la planificación inicial. Se analizan los riesgos y se desarrolla una versión o prototipo del sistema. El cliente la evalúa y con sus comentarios se refinan los requisitos, se reajusta la planificación, etc.

En cada iteración del modelo, el análisis de riesgos permitirá decidir si seguimos con el desarrollo o si abortamos el proyecto. Con el desarrollo de prototipos, estos se utilizaban para decidir los requisitos, y al final se debía desarrollar el sistema final. Aquí, en cada iteración tenemos una versión del producto, operativa, cada vez más completa.

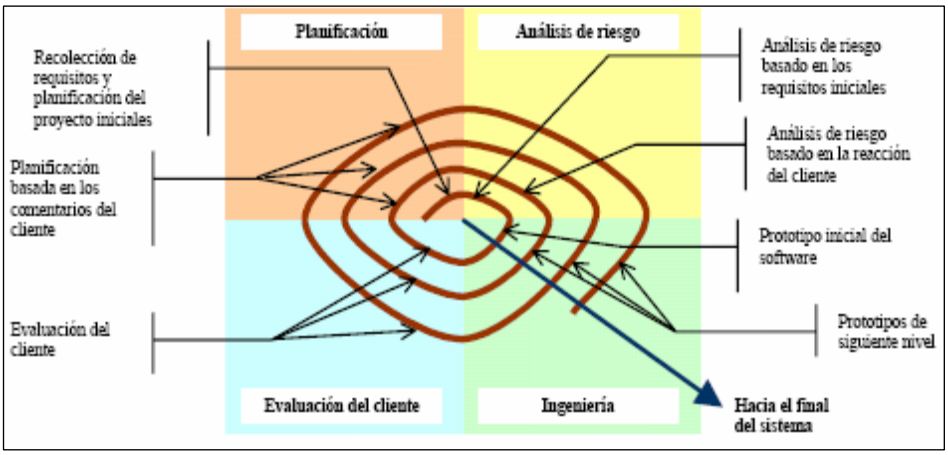

Figura 4. Modelo espiral

Fuente: (Pressman, 2002)

Además del modelo de prototipo como método evolutivo se tienen también los listados a continuación (Pressman, 2002):

- El modelo incremental.
- El modelo espiral WINWIN (Victoria & Victoria).
- El modelo de desarrollo concurrente.

1.2.2.4 MODELO DE ENSAMBLAJE DE COMPONENTES

Incorpora muchas de las características del Modelo Espiral. Es evolutivo por naturaleza y exige un enfoque interactivo para la creación del software. Sin embargo, el modelo ensamblador de componentes aplica para un marco de trabajo técnico del tipo orientado a objetos, por ende este modelo absorbe muchas

de las propiedades que caracterizan a este paradigma tales como el encapsulamiento, clases, reutilización, etc.

Para llevarlo a cabo, en primer lugar se identifica las clases candidatas examinando los datos que se van a manejar por parte de la aplicación y las operaciones a realizar sobre estos datos. Si estas clases han sido creadas por programas anteriores se almacenan en una biblioteca de clases o depósito. Se determina cuáles de ellas ya existen a fin de reutilizarlas. En caso de que exista alguna que no esté diseñada, se aplican los métodos orientados a objetos. Este proceso se inicia en el estado de Análisis de Riesgos del Modelo Espiral y se inserta en el estado de Construcción de Ingeniería del mismo.

En resumen, este modelo se basa en ir consolidando con la construcción de cada sistema una biblioteca de componentes (clases/objetos), cuando se va a construir un nuevo sistema, se debe hacer el proceso de definir los objetos del sistema, dentro de este paso se debe buscar en la librería de objetos, construir los que no existen, agregarlos en la biblioteca y ensamblar los objetos de la forma mas adecuada. La metodología busca que sea evolutiva pasando por una fase de planificación, análisis de riesgos, ingeniería, construcción y adaptación, evaluación del cliente y repetir estas fases de tal forma que las primeras iteraciones desarrollan los conceptos, al avanzar se desarrollan los nuevos componentes, luego se busca mejorarlos y finalmente se les da mantenimiento.

Dentro de este tipo de desarrollo se tiene el *proceso unificado de software* que representa un número de modelos basados en componentes que han sido puestos en la industria. Se utiliza el Lenguaje de Modelado Unificado (UML) para la definición de componentes y las interfaces que los conectara. Este proceso es la combinación del desarrollo incremental e iterativo y a su vez aplica un enfoque basado en escenario (punto de vista del usuario). (Pressman, 2002).

1.2.2.5 MODELO DE MÉTODOS FORMALES

Son modelos que aplican un proceso de desarrollo en donde enfocan su conjunto de actividades en lograr una especificación matemática del software, aplicando para esto, notación rigurosa y matemática. Hasta el momento no hay un enfoque formal claramente establecido, pero ofrecen producir software libre sin defectos, por aplicar matemática correcta a cada fase del proceso logrando eliminar defectos en cada una de las etapas de construcción del software. (Pressman, 2002).

1.1.2.6 TÉCNICAS DE CUARTA GENERACIÓN

El término técnicas de cuarta generación (T4G) abarca un amplio espectro de herramientas de software que tienen algo en común: Todas facilitan al ingeniero del software la especificación de algunas características del software de alto nivel y con estas las herramientas general el código automáticamente en base a las especificaciones dadas.

Un entorno de desarrollo de software que soporte el paradigma de T4G suele incluir algunas de las siguientes herramientas:

- Lenguajes no procedimentales para acceso a bases de datos. Son lenguajes de alto nivel (generalmente SQL o derivados) que nos permiten añadir, modificar, buscar o suprimir información de una base de datos sin preocuparnos de los detalles relativos a su organización física.
- Generadores de código, a partir de una especificación de requisitos.
- Generadores de pantallas y entornos gráficos.
- Generadores de informes.
- Herramientas que permiten disminuir la duración de las fases del ciclo de vida. Al igual que en otros paradigmas de desarrollo del software, usando técnicas 4G:
    - El proceso comienza con la recolección de requisitos. En el caso ideal serían traducidos directamente aun prototipo operativo, pero aquí vuelve a surgir la dificultad de expresar al comienzo todos los requisitos: sigue siendo necesario el diálogo cliente/desarrollador.

- El siguiente paso sería la realización del diseño.
- A continuación, en la fase de implementación podríamos centrarnos en la representación de los resultados porque la herramienta se encarga de generar automáticamente el código necesario.
- Podríamos pensar en saltarnos la prueba (código automático, entonces libre de errores), sin embargo debemos verificar que hemos realizado la especificación adecuadamente y que el producto funciona y entregárselo al usuario para que compruebe el producto.

Cuando surgieron las T4G muchos pensaron en que podría llegarse al momento en el que desaparecieran los programadores, pues cualquier usuario con unos conocimientos básicos podría crearse sus propias aplicaciones. Sin embargo esto no ha sido así, ya que:

- Las herramientas 4G no son mucho más fáciles de usar que los lenguajes de tercera generación. Disfrazan la codificación por especificación, pero se siguen necesitando conocimientos técnicos.
- El código fuente que generan es ineficiente.
- Se aplican casi exclusivamente al software de gestión. (Pressman, 2002).

1.2.3 MÉTODOS DE SOFTWARE

Los métodos de software indican como construir técnicamente el software, es por ello que pueden definirse como enfoques estructurados para el desarrollo de software cuyo propósito es facilitar la producción de software de alta calidad de una forma costeable, para ello se incluyen modelos de sistemas, notaciones, reglas, sugerencias de diseño y guías de procesos; entre las tareas que abarca cualquier método a emplear se tiene: análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento.

Métodos como *Análisis Estructurado* (DeMarco, T. 1978) y el método de desarrollo de Jackson *JSD* (Jackson, M. 1983) fueron los primeros desarrollados en los años 70. Estos métodos intentaron identificar los componentes funcionales básicos de

un sistema de tal forma que los métodos orientados a funciones aun se utilizan ampliamente. En los años 80 y 90, estos métodos orientados a funciones fueron complementados por métodos *Orientados a Objeto*, como los propuestos por (Booch, G. 1994) y Rumbaugh (Rumbaugh, J. 1991). Estos diferentes enfoques se han integrado a un solo enfoque unificado, basado en Lenguaje Unificado de Modelado (UML, Unified Modeling Lenguaje) (Fowler, M. y Scott, K. 1997; Booch G.; Rumbaugh J. *et al.* 1999; Rumbaugh, J.; Jacobson I. *et al.* 1999a, 1999b).

ENFOQUE	TIPO DE SISTEMA	FORMALIDAD
ESTRUCTURADOS Orientados a procesos Orientados a datos Jerárquicos No jerárquicos Mixtos	GESTIÓN (Sistema de información)	NO FORMAL
OO	TIEMPO REAL	FORMAL

Figura 5. Métodos de software
Fuente: (Sommerville, 2002)

Todos estos métodos se basan en la idea de modelos gráficos de desarrollo de un sistema y el uso de estos modelos como un sistema de especificación y diseño. Los métodos incluyen una variedad de componentes diferentes: descripción de los modelos de sistema, reglas, recomendaciones y guías del proceso.

No existe un método ideal y métodos diferentes tienen distintas áreas donde son aplicables. Por ejemplo, los métodos orientados a objetos a menudo son apropiados para sistemas interactivos, pero no para sistemas de tiempo real con requerimientos severos. (Sommerville, 2002)

1.2.4 HERRAMIENTAS DE SOFTWARE

Las herramientas de la Ingeniería de Software (ISW) proporcionan un enfoque automático o semi-automático para el proceso y para los métodos, la mas común en la ISW es uso de un sistema de soporte para el desarrollo del software llamado

Ingeniería del Software Asistida por Computadora (CASE) que se detalla brevemente a continuación. (Pressman, 2002).

1.2.4.1 CASE (INGENIERÍA DE SOFTWARE ASISTIDA POR COMPUTADORA)

Es el nombre que se le da al software que se utiliza para ayudar a las actividades del proceso del software como la ingeniería de requerimientos, el diseño, el desarrollo de programas y pruebas. Se puede decir, que CASE es la unión de las herramientas automáticas de software y las metodologías de desarrollo de software formales.

El uso de la tecnología CASE al automatizar actividades rutinarias mejora la calidad y productividad del software, el limite de esta tecnología se a restringido por la creatividad, un elemento clave para la ingeniería del software y también el manejo de las interacciones entre los equipos de trabajo que tampoco es una tarea fácil de automatizar.

Para lograr el uso de estas herramientas, es necesario realizar una integración hombre y computadora mediante estándares de interfaz que se están volviendo cada vez más comunes a lo largo y ancho de toda la industria. Para facilitar la integración de los usuarios con las herramientas, de las herramientas entre si, de las herramientas con los datos y de los datos con otros datos, se diseña así un modelo de referencia de integración. (Sommerville, 2002).

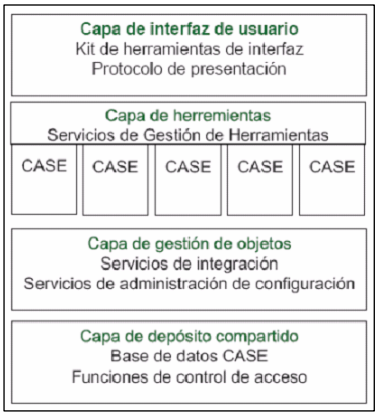

Figura 6. Modelo de referencia de integración

Fuente: (Sommerville, 2002)

1.2.4.2 CLASIFICACIÓN DE CASE

La clasificación de CASE ayuda a comprender los diferentes tipos de herramientas CASE y su importancia en la ayuda de las actividades del proceso de software. Existen varios enfoques para clasificarlas, pero hay tres aspectos posibles:

1. *Perspectiva funcional* en la que las herramientas CASE se clasifican de acuerdo a su función especifica.
2. Una *perspectiva de proceso* en la que las herramientas se clasifican de acuerdo con su ayuda a las actividades del proceso.
3. Una *perspectiva de integración* en la que las herramientas CASE se clasifican de acuerdo con la forma en que están organizadas en unidades integradas, las cuales proporcionan ayuda o mas actividades del proceso.

Para mayor detalle de cada una de las clasificaciones expuestas consultar al autor (Sommerville, 2002).

1.3 INGENIERÍA INVERSA DE SOFTWARE

El termino “ingeniería inversa” tiene sus orígenes en el mundo del hardware. Una cierta compañía desensambla un producto del hardware competitivo en un esfuerzo por comprender los “secretos” de diseño y fabricación de su competidor. Este trabajo no seria necesario si se obtuvieran las especificaciones de diseño y fabricación del competidor. Pero estos documentos son privados, y es por ello que la compañía debe efectuar lo que se conoce como ingeniería inversa para conocer el diseño y fabricación de la competencia. En esencia, una ingeniería inversa con éxito da lugar a una o más especificaciones de diseño y fabricación para el producto, mediante el examen de ejemplos reales de ese producto. (Pressman, 2002).

La ingeniería inversa del software se basa en el mismo principio aplicado en el caso del hardware. Sin embargo, en muchos casos, el software al que hay que hacer una ingeniería inversa no es un competidor, mas bien, es el propio trabajo de la compañía (con frecuencia, efectuado hace muchos años). Los “secretos” que

hay que comprender resultan incomprensibles porque no se llego a desarrollar nunca una especificación.

Es por ello, que la ingeniería inversa del software se define como el proceso consistente para analizar un programa en un esfuerzo por crear una representación del programa con un nivel de abstracción mas elevado que el código fuente. La ingeniería inversa es un proceso de recuperación de diseño. Las herramientas de ingeniería inversa extraen información acerca de los datos, arquitectura y diseño de procedimientos de un programa ya existente.

Llevando este concepto a un ejemplo practico, la ingeniería inversa invoca una imagen de “ranura mágica”. Se inserta un listado de código no estructurado y no documentado por la ranura (código fuente puro), y por el otro lado sale la documentación completa del software analizado. Lamentablemente, la ranura mágica no existe. La ingeniería inversa puede extraer la información de diseño a partir del código fuente, pero el nivel de abstracción, la completitud de la documentación, el grado con el cual trabajan al mismo tiempo las herramientas y el analista humano, y la dirección del proceso son sumamente variables. Analizando estos elementos variables tenemos:

- El *nivel de abstracción* de un proceso de ingeniería inversa y las herramientas que se utilicen para realizarlo aluden a la sofisticación de la información de diseño que se puede extraer del código fuente. Idealmente, el nivel de abstracción seria lo mas alto posible. Esto es, el proceso de ingeniería inversa debería ser capaz de derivar sus representaciones de diseño de procedimientos (con un bajo nivel de abstracción); y la información de programas de estructuras de datos (un nivel de abstracción ligeramente mas elevado); modelos de flujos de datos y de control (un nivel de abstracción relativamente alto); y modelos de entidades y de relaciones (un elevado nivel de abstracción). A medida que crece el nivel de abstracción se proporciona al ingeniero del software información que le permitirá una comprensión más sencilla de estos programas.

- La *completitud* de un proceso de ingeniería inversa alude al nivel de detalle que se proporciona en un determinado nivel de abstracción. En la mayoría de los casos, la completitud decrece a medida que aumente el nivel de abstracción. Por ejemplo, dado un listado del código fuente, es relativamente sencillo desarrollar una representación de diseño de procedimientos completo. También se pueden derivar sencillas representaciones del flujo de datos o un programa de transición de estados. La completitud mejora en proporción directa con la cantidad de análisis efectuado por la persona que efectúe la ingeniería inversa.

- La *interactividad* alude al grado con el cual el ser humano "se integra" con unas herramientas automatizadas para crear un proceso de ingeniería inversa efectivo. En la mayoría de los casos, a medida que crece el nivel de abstracción, la interactividad debe de incrementarse o bien la complejidad se vera reducida.

- Si la *direccionabilidad* del proceso de ingeniería inversa es monodireccional, toda la información extraída del código fuente se proporcionara a la ingeniería del software que podrá entonces utilizarla durante la actividad de mantenimiento. Si la dirección es bidireccional, entonces la información que se suministrara a una herramienta de la reingeniería que intentara reestructurar o regenerar el viejo programa.

Es por esto, que el núcleo de la ingeniería inversa es una actividad denominada extracción de abstracciones. El ingeniero tiene que evaluar el viejo programa y a partir del código fuente (que suele no estar documentado) tiene que extraer una especificación significativa del procesamiento que se realiza, la interfaz de usuario que se aplica, y las estructuras de datos de programa o de la base de datos que se utiliza.

1.3.1 INGENIERÍA INVERSA PARA COMPRENDER EL PROCESAMIENTO

La primera actividad real de la ingeniería inversa comienza con un intento de comprender y después extraer abstracciones de procedimientos representadas por el código fuente. Para comprender las abstracciones de procedimientos, se analiza el código en distintos niveles de abstracción, sistema, programa, modulo, trama y sentencia.

Por ende, la funcionalidad general de todo el sistema debe ser algo perfectamente comprendido antes de que se produzca un trabajo de ingeniería inversa mas detallado. Así se establece un contexto para su posterior análisis, y se proporcionan ideas generales acerca de los problemas de interoperabilidad entre aplicaciones dentro del sistema. Cada uno de los programas de que consta el sistema de aplicaciones representara una abstracción funcional con un elevado nivel de detalle. Un diagrama de bloques, que represente la iteración entre estas abstracciones funcionales, se creara también. Los módulos efectúan cada uno de ellos una subfunción, y representan una abstracción de procedimientos definida. Se crean alternativas de procesamiento para cada uno de los modelos. En algunos casos, ya existen especificaciones de sistema, programa, y modulo. Cuando tal cosa ocurre, se revisan las especificaciones para apreciar si se ajustan al código existente.

Las cosas se vuelven mas complicadas cuando se considera el código que reside en el interior de un modulo. El ingeniero busca secciones de código que representen tramas de procedimientos genéricos. En casi todos los módulos, existe una sección de código que prepara los datos para su procesamiento, una sección diferente de código que efectúa el procesamiento, y otra sección de código que prepara los resultados del procesamiento para exportarlos de ese modulo.

En el interior de cada una de estas secciones, se encuentran tramas más pequeñas (por ejemplo: suele producirse una verificación de los datos y una

comprobación de alcances dentro de la sección de código que prepara los datos para su procesamiento).

Para grandes sistemas, la ingeniería inversa suele efectuarse mediante el uso de un enfoque semiautomático. Se utilizan herramientas CASE (por ejemplo: para "analizar" la semántica del código existente). La salida de este proceso se pasa entonces a unas herramientas de reestructuración y de ingeniería progresiva que complementaran el proceso de reingeniería. (Pressman, 2002).

1.3.2 INGENIERÍA INVERSA PARA COMPRENDER LOS DATOS

La ingeniería inversa de datos suele producirse en distintos niveles de abstracción. En el nivel de programa, es frecuente que sea preciso realizar una ingeniería inversa de las estructuras de datos de programa internas, como parte de un esfuerzo global de reingeniería. En el nivel del sistema, es frecuente que se efectúe una reingeniería. En el nivel del sistema, es frecuente que se efectúe una reingeniería de las estructuras globales de datos (por ejemplo: archivos, bases de datos) para ajustarlas a los nuevos paradigmas de gestión de bases de datos (por ejemplo: el paso de unos archivos planos a unos sistemas de bases de datos relacionales u orientados a objetos). La ingeniería inversa de las estructuras de datos globales actuales establece el escenario para la introducción de una nueva base de datos que abarque todo el sistema. Existen dos estructuras de datos que se pueden definir y son (Pressman, 2002):

- *Estructuras de datos internas:* En la ingeniería inversa se centra en la definición de clases de objetos, esto se logra examinando el código del programa en un intento de agrupar variables de programas que estén relacionadas, según (Breuer, P. y Lano, K. 1991) sugieren el siguiente enfoque:
  - Identificador de los indicadores y estructuras de datos locales dentro del programa (archivos, base de datos, etc.),
  - Definición de las relaciones entre indicadores y estructuras de datos locales y las estructuras de datos globales.

- Toda variable que represente una matriz o un archivo, listar todas las variables adicionales que tengan relación con ella.

- *Estructura de base de datos:* las bases de datos permiten definir objetos de datos, y apoyan los métodos de establecer relaciones entre objetos, para definir este modelo de datos según (Premerlani, 1994), se emplean los siguientes datos:
  - Construcción de un modelo de objeto inicial.
  - Determinación de los candidatos a claves.
  - Refinamiento de las claves provisionales
  - Definición de las generalizaciones.
  - Descubrimiento de las asociaciones

1.3.3 INGENIERÍA INVERSA PARA INTERFACES DE USUARIO

Para realizar reingeniería inversa de una interfaz de usuario ya existente, es preciso especificar la estructura y comportamiento de la interfaz. El autor Merlo (Merlo, E. *et al*. 1993) propone la elaboración de tres preguntas:

- ¿Cuáles son las acciones básicas que deber procesar la interfaz?
- ¿Cuál es la descripción compacta de la respuesta de comportamiento del sistema a estas acciones?
- ¿Qué concepto de equivalencia de interfaces es relevante en este caso?

La notación de modelado de comportamiento puede proporcionar una forma de desarrollar las respuestas de las dos primeras preguntas. Este modelo se logra con la observación de la manifestación externa de la interfaz existente y seguido del código se extrae la información adicional para completar el modelo.

Con el último punto pueden existir ciertas diferencias con la interfaz real, se puede colocar que el comienzo de una acción se activa con el ratón y realmente se activa con un comando distinto.

1.4 LENGUAJE DE MODELADO UNIFICADO (UML)

1.4.1 UML

El Lenguaje de Modelación Unificado, conocido por sus siglas en ingles como UML (Unified Modeling Language), es un lenguaje de modelación común en la industria y que permite la especificación, visualización, construcción y documentación de los elementos que constituyen un sistema de software orientado a objetos u otros sistemas que no estén relacionados con la creación de software (OMG, 2005). UML, representa la unificación de los métodos Booch (Booch, G. 1994), OMT (Object Modeling Technique) (Rumbaugh, J. 1995) y OOSE (Object-Oriented Software Engineering) (Jacobson, I. 1992). El lenguaje de Modelación Unificado es un estándar para construir modelos.

Jacobson define el lenguaje UML como: “un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir“ (Rumbaugh, J.; Jacobson I. et al., 1999b).

El lenguaje UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar un trabajo que finalmente beneficia a un usuario externo. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar.

UML no es un lenguaje de programación. Las herramientas pueden ofrecer generadores de código de UML para una gran variedad de lenguaje de programación, así como construir modelos por ingeniería inversa a partir de programas existentes.

UML es un lenguaje de propósito general para el modelado orientado a objetos. UML es también un lenguaje de modelamiento visual que permite una abstracción del sistema y sus componentes. Existían diversos métodos y técnicas orientadas a objetos, con muchos aspectos en común pero utilizando distintas notaciones, se presentaban inconvenientes para el aprendizaje, aplicación, construcción y uso de herramientas, etc., además de pugnas entre enfoques, lo que genero la creación del UML como estándar para el modelamiento de sistemas de software principalmente, pero con posibilidades de ser aplicado a todo tipo de proyecto (Rumbaugh, J.; Jacobson I. *et al*., 1999b).

1.4.2 OBJETIVOS DE UML

Entre los principales objetivos de UML se tienen:

- Proveer semántica y notación suficiente para dirigirse a una amplia variedad de problemas de modelación.
- Proveer semántica para dirigirse a futuros problemas de modelación (tecnológicos y distribuidos).
- Proveer semántica para facilitar el intercambio entre una gran variedad de herramientas del mercado orientado a objeto.
- Proveer semántica para especificar interfaces necesarias en el desarrollo del sistema.
- Proveer mecanismos de extensibilidad para proyectos individuales que puedan extender el metamodelo, a través del uso de estereotipos, valores etiquetados y restricciones. (OMG, 2005).
- UML es un lenguaje de modelado de propósito general que pueden usar todos los modeladores. No tiene propietario y está basado en el común acuerdo de gran parte de la comunidad informática.
- UML no pretende ser un método de desarrollo completo. No incluye un proceso de desarrollo paso a paso. UML incluye todos los conceptos que se consideran necesarios para utilizar un proceso moderno iterativo, basado en construir una sólida arquitectura para resolver requisitos dirigidos por casos de uso.

- Ser tan simple como sea posible pero manteniendo la capacidad de modelar toda la gama de sistemas que se necesita construir. UML necesita ser lo suficientemente expresivo para manejar todos los conceptos que se originan en un sistema moderno, tales como la concurrencia y distribución, así como también los mecanismos de la ingeniería de software, como son la encapsulación y componentes.
- Ser un lenguaje universal, como cualquier lenguaje de propósito general.
- Imponer un estándar mundial.

1.4.3 ÁREAS CONCEPTUALES DE UML

Los conceptos y modelos de UML pueden agruparse en las siguientes áreas conceptuales:

- *Estructura estática:* Cualquier modelo preciso debe primero definir su universo, esto es, los conceptos clave de la aplicación, sus propiedades internas, y las relaciones entre cada una de ellas. Este conjunto de construcciones es la estructura estática. Los conceptos de la aplicación son modelados como clases, cada una de las cuales describe un conjunto de objetos que almacenan información y se comunican para implementar un comportamiento. La información que almacena es modelada como atributos; La estructura estática se expresa con diagramas de clases y puede usarse para generar la mayoría de las declaraciones de estructuras de datos en un programa.
- *Comportamiento dinámico:* Hay dos formas de modelar el comportamiento, una es la historia de la vida de un objeto y la forma como interactúa con el resto del mundo y la otra es por los patrones de comunicación de un conjunto de objetos conectados, es decir la forma en que interactúan entre sí. La visión de un objeto aislado es una maquina de estados, muestra la forma en que el objeto responde a los eventos en función de su estado actual. La visión de la interacción de los objetos se representa con los enlaces entre objetos junto con el flujo de mensajes y los enlaces entre ellos. Este punto de vista unifica la estructura de los datos, el control de flujo y el flujo de datos.
- *Construcciones de implementación:* Los modelos UML tienen significado para el análisis lógico y para la implementación física. Un componente es una parte

física reemplazable de un sistema y es capaz de responder a las peticiones descritas por un conjunto de interfaces. Un nodo es un recurso computacional que define una localización durante la ejecución de un sistema. Puede contener componentes y objetos.

- *Organización del modelo:* La información del modelo debe ser dividida en piezas coherentes, para que los equipos puedan trabajar en las diferentes partes de forma concurrente. El conocimiento humano requiere que se organice el contenido del modelo en paquetes de tamaño modesto. Los paquetes son unidades organizativas, jerárquicas y de propósito general de los modelos de UML. Pueden usarse para almacenamiento, control de acceso, gestión de la configuración y construcción de bibliotecas que contengan fragmentos de código reutilizable.
- *Mecanismos de extensión:* UML tiene una limitada capacidad de extensión pero que es suficiente para la mayoría de las extensiones que se requieren, sin la necesidad de un cambio en el lenguaje básico. Un estereotipo es una nueva clase de elemento de modelado con la misma estructura que un elemento existente pero con restricciones adicionales.

En la siguiente tabla (Ver Tabla 1) se muestra la coherencia de las áreas previamente descritas con los nueve modelos propuestos por UML:

Área	Vista	Diagramas	Conceptos Principales
Estructural	Vista Estática	Diagrama de Clases	Clase, asociación, generalización, dependencia, realización, interfaz.
	Vista de Casos de Uso	Diagramas de Casos de Uso	Caso de Uso, Actor, asociación, extensión, generalización.
	Vista de Implementación	Diagramas de Componentes	Componente, interfaz, dependencia, realización.
	Vista de Despliegue	Diagramas de Despliegue	Nodo, componente, dependencia, localización.
Dinámica	Vista de Estados de máquina	Diagramas de Estados	Estado, evento, transición, acción.
	Vista de actividad	Diagramas de Actividad	Estado, actividad, transición, determinación, división, unión.
	Vista de interacción	Diagramas de Secuencia	Interacción, objeto, mensaje, activación.
		Diagramas de Colaboración	Colaboración, interacción, rol de colaboración, mensaje.

Administración o Gestión de modelo	Vista de Gestión de modelo	Diagramas de Clases	Paquete, subsistema, modelo.
Extensión de UML	Todas	Todos	Restricción, estereotipo, valores, etiquetados.

Tabla 1. *Estructura completa de UML*

Fuente: (Pérez, 2002)

Ahora describiremos un poco cada uno de los diagramas que contiene cada área expuesta anteriormente:

- *Diagrama de Casos de Uso:* modela la funcionalidad del sistema agrupándola en descripciones de acciones ejecutadas por un sistema para obtener un resultado.
- *Diagrama de Clases:* muestra las clases (descripciones de objetos que comparten características comunes) que componen el sistema y cómo se relacionan entre sí.
- *Diagrama de Objetos:* muestra una serie de objetos (instancias de las clases) y sus relaciones.
- *Diagrama de Secuencia:* enfatiza la interacción entre los objetos y los mensajes que intercambian entre sí junto con el orden temporal de los mismos.
- *Diagrama de Colaboración:* igualmente, muestra la interacción entre los objetos resaltando la organización estructural de los objetos en lugar del orden de los mensajes intercambiados.
- *Diagrama de Estados:* modela el comportamiento de acuerdo con eventos.
- *Diagrama de Actividades:* simplifica el Diagrama de Estados modelando el comportamiento mediante flujos de actividades.
- *Diagrama de Componentes:* muestra la organización y las dependencias entre un conjunto de componentes.
- *Diagrama de Despliegue:* muestra los dispositivos que se encuentran en un sistema y su distribución en el mismo. ((Rumbaugh, J.; Jacobson I. et al., 1999b) y Pérez at el. 2002).

Ciertos conceptos están asociados a UML: vistas, diagramas, elementos del modelo y mecanismos generales.

Las *vistas* muestran diferentes aspectos de los sistemas que son modelados. Una vista no es un gráfico, es una abstracción que consiste en una serie de diagramas. Definiendo una serie de vistas, cada una mostrando un aspecto particular del sistema, puede ser construida una imagen completa del sistema.

Los *Diagramas* son los gráficos que describen los contenidos de las vistas. UML tiene nueve tipos diferentes de diagramas, que son usados en combinación para proporcionar diferentes vistas de un sistema.

Los diagramas de secuencia, colaboración, estado y actividad, están asociados al modelo de comportamiento. Los diagramas de ejecución y componentes están asociados al modelo de implementación. Los diagramas de objetos y clases están asociados al modelo estructural y el diagrama de casos de uso está asociado al modelo funcional. A continuación se muestran los modelos y sus diagramas de asociados. (Soriano, 2000).

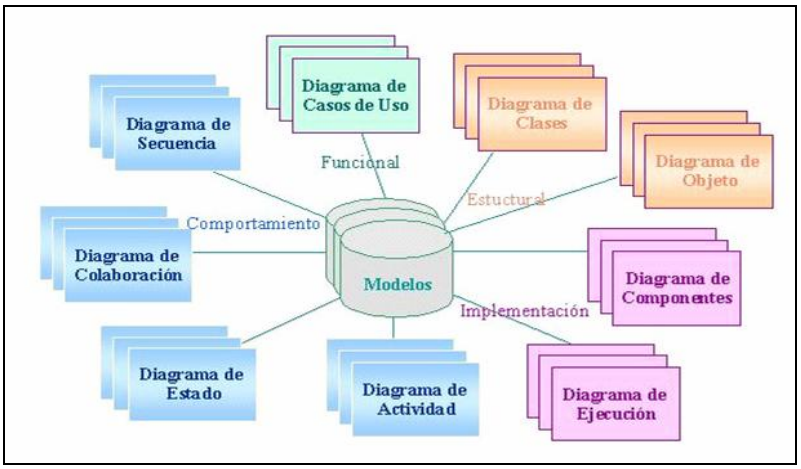

Figura 7. Modelos y diagramas UML

Fuente: (Soriano, 2002)

Los *Elementos del modelo* representan los conceptos utilizados en los diagramas, tales como clases, objetos, mensajes, relaciones de asociación, dependencia,

generalización. Un elemento del modelo puede ser utilizado en diferentes tipos de diagramas, pero siempre tiene el mismo significado y símbolo.

Los *mecanismos generales* proporcionan comentarios extras, información o semántica acerca de un elemento del modelo, ellos proporcionan también mecanismos de extensión para adaptar o extender UML a un método, proceso, organización o usuario específico.

UML es importante porque hoy en día representa un estándar de la industria OMG (Object Management Group), es un lenguaje de modelación completo, la mayoría de organizaciones y vendedores de herramientas ya lo han adoptado y soporta todas las fases del proceso de desarrollo de software.

CAPÍTULO II – MÉTODO

En este capítulo se utilizan los conceptos del marco teórico para explicar el procedimiento de cómo se va a realizar el trabajo. Este capítulo contiene los aspectos relacionados con el método, consta del planteamiento del problema, las variables de diseño, la descripción del tipo de intervención, el diseño de la intervención, los instrumentos a usar y el procedimiento a realizar.

2.1 PROBLEMA

La investigación y el desarrollo de técnicas y métodos de ingeniería del software son constantes y suelen suponer interesantes avances en la resolución de problemas de desarrollo del software. Sin embargo, es habitual que en la práctica diaria profesional no se incluya prácticamente ninguna de las recomendaciones más elementales de la ingeniería del software. En efecto, es común que el desarrollo de software parezca mas al descontrol del cuento de “si los programadores fueran albañiles…” (Novatita, 1996).

De hecho, las evaluaciones de los procesos productivos de software realizadas a raíz de los procesos de software, confirman que el desarrollo de software suele estar básicamente en estado caótico. Y no solo en, como uno podría pensar, pequeñas organizaciones de un país como el nuestro, sino también en empresas adjudicatarias de interés en EE.UU. o Japón.

Aunque los medios de evaluación suelen ser rígidos, revelan en la practica despreocupación de los responsables de las organizaciones de software por la mejora de la calidad de los procesos de trabajo o de sus productos: bien sea por contar con una situación interna de la empresa poco propicia, porque el ambiente de mercado no fomenta la preocupación por estos temas y solo se enfocan en resultados de momento o bien por su poco interés real en la búsqueda de la calidad.

Uno de los problemas más críticos y silenciosos en el desarrollo de software es conseguir productos desarrollados, ya operativo y con poca o inexistente documentación, que apoye los procesos de ingeniería de software. Esta falta de documentación conlleva a que exista ausencia de manuales y un esquema completo de funcionamiento, lo cual dificulta realizar de manera optima la estimación de mejoras razonables en los procesos, aplicar cambios con su respectiva medición de impacto real y dar continuidad a aplicaciones o nuevos requerimientos del sistema.

La falta de documentación implica directamente debilidades en el manejo del conocimiento, el cual queda sectorizado solo entre el grupo de proyecto que desarrolla el software y cada uno manejando su área de alcance, esto conlleva a que se tenga una fuerte dependencia de personas especificas para llevar a cabo las tareas, trayendo como secuela que ningún otro grupo o miembro que no este involucrado desde el inicio, no pueda estar relacionado directamente a la actividad, no pueda asumir de manera rápida esas labores en caso de alguna eventualidad o ausencia por no contar con la especificación de software correspondiente al proceso que se va a trabajar y puede desconocer a profundidad.

Haciendo caso puntual en el marco de referencia de Venezuela, existen en el país muchas empresas que manejan desarrollo de software, unas especializadas y otras que aunque su área de producción precisamente no es el desarrollo de software, adoptaron esta tarea como propia para resolver necesidades internas y brindar mejoras en sus productos y servicios, utilizando para ello productos de software, que en muchos casos son desarrollos internos.

Una empresa nacional que a asumido esta es CANTV (Compañía Anónima Nación de Teléfonos de Venezuela), una empresa de telecomunicaciones venezolana, la cual, se especializa en las telecomunicaciones pero realiza desarrollos de software propios para dar soluciones a necesidades especificas de

su operación, por su tamaño y diversidad de áreas existen ciertos desarrollos que no cumplen con las características antes mencionadas, esto se evidencia en sistemas que están operativos y que no cuentan con su respectiva documentación y si existe no cumple con ningún estándar de desarrollo de software.

Un producto de software que vive esta realidad dentro de esta corporación, es el Sistema de Aprovisionamiento de la Red de Banda Ancha, el cual, es el software crítico de uno de los servicios de valor agregado que ofrece la corporación a sus clientes. Es conocido como el producto Acceso Banda Ancha (ABA) (Ver Anexo A), el cual ofrece a sus clientes una experiencia de navegación, que se traduce en beneficios como mayor velocidad y disponibilidad.

Para poder prestar este servicio, dentro de la corporación existen diversas unidades, cada una con tareas especificas que integradas permiten otorgar el servicio definido. Este producto esta compuesto por un conjunto de soluciones de software desarrollados por la Coordinación de Aprovisionamiento de Servicio de la Gerencia de Ingeniería, Desarrollo y Construcción TI/SI, los cuales se encargan entre otras cosas de recibir y procesar los diversos requerimientos que involucren a la plataforma, bien sea nuevos servicios, mejoras o ajustes.

Debido al auge e incremento desproporcionado del servicio en sus inicios, trajo como consecuencias que en el proceso de desarrollo de software usado para desarrollar el sistema de aprovisionamiento de la Red de Banda Ancha no se realizara ningún tipo de documentación del sistema y no se maneja ningún tipo de estándar de especificaciones de software, que son los problemas críticos y mas comunes que mencionamos anteriormente.

Adicionalmente, actualmente a nivel externo dentro de la corporación se tiene un problema de desconocimiento y la visión de una Caja Negra (Aquel elemento que es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno.), nadie

sabe exactamente que se realiza, quien, ni como, y actualmente no se puede ofrecer una documentación que siga un estándar de especificación de software que permita mostrar los procesos del sistema abiertamente.

Por lo antes mencionado, surge la necesidad de generar la documentación necesaria del sistema de Aprovisionamiento de la Red Banda Ancha CANTV con el uso de un lenguaje estándar para la especificaciones de software que permita, especificar, construir y documentar las entidades que forman un sistema, logrando así tener una base documental de todos los procesos manejados, de manera de poder realizar mejores estimaciones (nuevos requerimientos, cambios, mejoras) y lograr la distribución unificada del conocimiento a nivel interno y externo del departamento.

Para dar respuesta a la necesidad de obtener especificaciones de software partiendo de software ya desarrollado (código fuente), se plantea el uso de la ingeniería inversa, la cual plantea justamente la resolución de esta necesidad y permite generar modelos haciendo uso de algún método de ingeniería de software, los detalles teóricos están descritos en las sesiones 1.2 y 1.3 del Capítulo 1.

2.2 OBJETIVO

Este trabajo especial de grado tiene como principal objetivo la realización de la ingeniería inversa del sistema actual de Aprovisionamiento de la Red Banda Ancha de la corporación CANTV, obteniendo de esta, la documentación del sistema que representara la especificación de software estándar del mismo.

Para alcanzar la realización de esta documentación basados en la ingeniería inversa, se debe:

- Aplicar el enfoque unificado, basado en UML (Lenguaje de Modelado Unificado) en UML como método de software.
- Utilizar alguna herramienta CASE como herramienta de software que apoye el enfoque unificado.

- Para la ingeniería inversa se debe extraen información acerca de los datos, arquitectura y diseño de procedimientos del sistema.

2.3 DEFINICIÓN DE VARIABLES

Las variables o parámetros de diseño relacionadas con el trabajo a realizar son dos (2): variables de modelado y variables de sistema y confidencialidad.

2.3.1 VARIABLES DE MODELADO DE SISTEMAS DE SOFTWARE

Las variables de modelado de sistemas de software se refieren a los términos relacionados con el enfoque unificado UML (Unified Modeling Language) como método de software a usar para especificaciones de software. UML contiene un conjunto de diagramas, de los cuales debe conocer su contexto, notación, que maneja y su utilidad, los detalles de los mismos se pueden consultar en la sesión 1.4 del capitulo 1.

Entre las variables tenemos:

- Diagramas para el modelo estático.
- Diagramas para el comportamiento.
- Diagramas de aplicación.

2.3.2 VARIABLES DE SISTEMA Y CONFIDENCIALIDAD

Solo se contemplarán los procesos de sistema y de clientes, no se tocaran aspectos corporativos de cuidado, ni de mercadeo del servicio por lo delicado de los mismos y de mantener una confidencialidad de la información. Se tratara de solo enfocar las especificaciones necesarias que le permitan a cualquier integrante del departamento o de la corporación poder asumir la continuidad de cualquier desarrollo realizado o lograr realizar estimaciones o nuevas ideas sin afectar lo que ya esta en producción y operativo. La definición del enfoque de sistema a manejar esta en la sección 1.1.

Entre las variables tenemos:

- Principios de sistema de aprovisionamiento.
- Aspectos técnicos y de mercado sobre el producto.
- Seguridad y privacidad de la información obtenida.

2.4 TIPO DE INTERVENCIÓN

El tipo de intervención es el tipo de trabajo que se va a realizar. El trabajo que se presenta es un diseño de proyecto factible que muestra la aplicación de la ingeniería inversa sobre un software para generar las especificaciones de datos, arquitectura y diseño de procedimientos de un programa ya existente (código fuente). Estas especificaciones de software generadas por la aplicación de la ingeniería inversa, son generadas a partir de:

- Enfocados en el desarrollo basado en componentes como modelo de software a seguir. Detalles en la sección 1.2.2.3 y 1.2.2 del capitulo 1.
- Utilización del método orientado a objetos y el enfoque unificado con UML a través de la aplicación de sus diagramas UML, explicados en la sección 1.4, ver también sección 1.2.3, capítulo 1.
- Utilización de una herramienta CASE para realizar los diagramas UML explicadas en la sección 1.2.4 capitulo 1.
- Aplicación de las técnicas y principios de la ingeniería inversa explicado en la sección 1.3, capítulo 1.

2.5 DISEÑO DE INTERVENCIÓN

En esta sección se describe el diseño de la intervención como los pasos a seguir para lograr aplicar la ingeniería inversa al sistema y obtener sus especificaciones de software.

El diseño de la intervención consiste en el seguimiento de los pasos para la ingeniería inversa y el uso de algunos modelos que sugiere UML, para la realización del trabajo se define los siguientes pasos a realizar:

Paso 1: Realizar una reestructuración del código fuente no estructurado para generar el código fuente estructurado.

Paso 2: Realizar la extracción de abstracciones, obteniendo así el procesamiento o funcionabilidad, interfaz y base de datos, esto se realiza con la aplicación de los diagramas de UML (Diagrama de Casos de Usos, Diagrama de Componentes, Diagrama de Clases, Diagrama de Secuencia y Diagrama de Actividades) y obtener de esto la especificación inicial.
Paso 3: Se refina, simplifica y agrupa estas especificaciones elaboradas con UML en una especificación final.

2.6 INSTRUMENTOS

En esta sección se describen los instrumentos de software y de hardware que se utilizaron para la realización del trabajo.

2.6.1. INSTRUMENTOS DE SOFTWARE PARA EL DESARROLLO DEL SISTEMA

En el diseño explicado en la sección 2.5, se mencionaron tres (3) pasos a realizar. Para cada paso se utilizará uno o varios instrumentos que permitieron la ejecución del mismo.

Para el paso 1, se realizó una revisión del software, de manera de estructurar el código fuente, de forma que solamente contenga construcciones de programación estructurada, este trabajo se realizo en conjunto con los desarrolladores del sistema, para ello se dispuso de conexiones remotas a el código fuente con el uso de la herramienta Putty (es un cliente SSH, Telnet, rlogin, y TCP raw con licencia libre) y con acceso a la Red Privada Virtual (VPN, Virtual Point Network) de la red de CANTV, también se contó con Microsoft Office Word para la realización de las especificaciones y observaciones obtenidas.

Para el paso 2, se evaluaron diversas herramientas CASE existentes en mercado actual, entre las que más destacan: Visio Profesional, Rational Rose y software gratuito que permiten hacer modelos UML (Poseidon, ArgoUML, etc.). Para este

trabajo se escogió ArgoUML-0.22, elaborado por: http://www.tigris.org/ que es una organización de software Open Source, lo que hace que este software sea gratuito y código abierto, hecho en Java e instalable de manera local en el computador sin requerir recursos adicionales de los ya existentes en los lineamientos corporativos de los computadores de la empresa.

En esta decisión también tuvo importancia la exigencia por parte de la gerencia de llevar a cabo alguna especificación de software estándar basada en alguna herramienta CASE que no representara ningún gasto adicional a la empresa y generara la documentación necesaria para dar soporte de cualquier aspecto de la aplicación.

Para lo que se basa este trabajo, el punto clave, además de los mencionados, es que la herramienta escogida permita la construcción de los diversos diagramas de UML y use la sintaxis definida en la guía de “OMG Unified Modeling Language Specification” (OMG, 2005).

Pare el paso 3, una vez realizados los diagramas UML en la herramienta CASE escogida, se realiza una documentación que centralice las especificaciones obtenidas con estos diagramas, para ello se realizo una nueva revisión de lo que se especifico en los mismos, con lo que realiza el sistema, se ajusto cualquier detalle y se obtuvo las especificaciones finales del sistema, esto es lo desarrollado en el capitulo 3, para el cual se uso de Microsoft Office Word.

Luego de terminado el proceso de ingeniería, se propone la entrega de este Trabajo Especial de Grado como el documento de especificaciones finales obtenidos a la corporación CANTV, particularmente la Coordinación de Aprovisionamiento del Servicios.

2.6.2. INSTRUMENTOS DE HARDWARE PARA EL DESARROLLO DEL SISTEMA

Se requirió una computadora que sus especificaciones soporten la herramienta CASE escogida instalada, que tenga mínimo 256 MB de memoria RAM y un

espacio de disco duro suficiente para instalar la herramienta CASE y manejar documentos de gran tamaño.

2.7 PROCEDIMIENTO

El procedimiento que se siguió para realizar el Trabajo Especial de Grado es fácil de describir en función de lo mencionado en las sesiones anteriores de este capítulo.

El procedimiento consistió en:

- Realizar un estudio detallado del código fuente de la aplicación con ayuda de los desarrolladores, del cual se obtuvo un instructivo completo del sistema de aprovisionamiento.
- Usar algunos de los diagramas de UML con la sintaxis descrita en la guía de “OMG Unified Modeling Language Specification” (OMG, 2005), específicamente los diagramas que permitieron describir procesamiento (Diagrama de Casos de Usos y Diagrama de Componentes), datos (Diagrama de Clases) e interfaz (Diagrama de Secuencia y Diagrama de Actividades).
- Realizar los pasos descritos en el diseño de la intervención, sección 2.5 de este capítulo.
- Usar las herramientas descritas en la sección 2.6 de este capítulo.

CAPÍTULO III – APLICACIÓN DE LA INGENIERÍA INVERSA DE SOFTWARE

Este capítulo contiene la aplicación de la ingeniería inversa de software sobre el Sistema de Aprovisionamiento de la Red de Banda Ancha de CANTV, a continuación se muestra el desarrollo de todas las actividades propuestas a consecuencia de lo descrito en el capitulo 2.

3.1 REESTRUCTURACIÓN DEL CODIGO FUENTE

Esta actividad se llevo acabo realizando un estudio y ajuste de todo el código fuente existente con el objetivo de levantar toda una documentación que permitiera generar la especificación completa de lo que ese código contenía y de lo que el sistema conlleva como un todo y sus partes.

Se utiliza el estándar corporativo para la realización de los instructivos de los programas, permitiendo generar todas las especificaciones de los mismos de una manera organizada y estructurada manejada dentro de la Corporación CANTV.

El resultado de esta es el documento denominado: "*Administración de Plataforma de Aprovisionamiento ABA (Provisioning-combo)"* (Ver Anexo B), el cual represento la base del resto del trabajo a mostrar en este capitulo y el cumplimiento de la reestructuración del código fuente.

3.2 ESPECIFICACIONES UML PARA LA ABSTRACCIÓN DEL PROCESAMIENTO

En base a la metodología de la ingeniería inversa de software esta actividad se debe enfocar en la toma de información de dos puntos claves:

- Abstracción funcional, que permita conocer el funcionamiento general del sistema.
- Abstracción de procedimientos del código fuente, tomando como base el sistema, programas, módulos y secciones del código fuente, para saber como esta compuesto el sistema.

La compresión del código fuente que es una de las actividades claves se desarrollo en el punto 3.1 de este capitulo, por ende se tomo como base para trabajar con un nivel de abstracción superior que permitiese formalizar las funcionalidades generales del sistema en lenguaje manejado por el cliente, este nivel requerido fue llevado a cabo con el diseño del diagrama de UML que permite modelar la parte funcional de los sistemas, el cual, es el Diagrama de Casos de Usos, este fue llevado a cabo y expuesto en la sección 3.2.1 de este capitulo.

Posteriormente, se requería obtener el modelo que representara el conjunto de componentes, estructuras y recursos involucrados en el cual se pudiese representar como es el ambiente de funcionamiento donde esta conviviendo el sistema, para ello, basados en lo estudiado en la ingeniería inversa se debe diseñar un Diagrama de Bloques, en UML el que permite modelar como es la vista de implementación del sistema es el Diagrama de Componentes o Ejecución, el cual esta expuesto en la sección 3.2.2 de este capitulo.

En función de la metodología de ingeniería inversa, se concluye, que con el uso de los dos diagramas UML mencionados anteriormente, se logra obtener la abstracción de procesamiento necesaria para la comprensión y documentación de este aspecto del sistema. A continuación el desarrollo de ambos diagramas.

3.2.1 DIAGRAMAS DE CASOS DE USOS DE PROVISIONING-COMBO

Este modelo describe las funcionalidades del sistema según la perspectiva del usuario. Los diagramas de Casos de Uso nos permiten especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. Estos se utilizan para ilustrar los requerimientos del sistema al mostrar como reacciona una respuesta a eventos que se producen en el mismo.

Las actividades definidas para realizar el diagrama son:

- Identificación de los Actores y Casos de Uso Principales.
- Refinamiento de los Casos de Uso Principales.

3.2.1.1 IDENTIFICACIÓN DE LOS ACTORES Y CASOS DE USO PRINCIPALES

Se identifican en un primer nivel los actores que interactúan con el caso de uso general (Ver Figura 8), en el caso del sistema definido existe un único actor definido a continuación:

- *Crontab del Servidor de Aprovisionamiento:* Es la aplicación que permite programar tareas que se ejecutarán en un momento determinado del tiempo de manera automática. El sistema Linux (y cualquier UNIX en general) comprueba regularmente, guiándose por el evento de reloj del sistema, si existe alguna tarea programada para ejecutarse y, en caso afirmativo, la ejecuta sin necesidad de que nadie (ningún usuario) lo haga explícitamente.

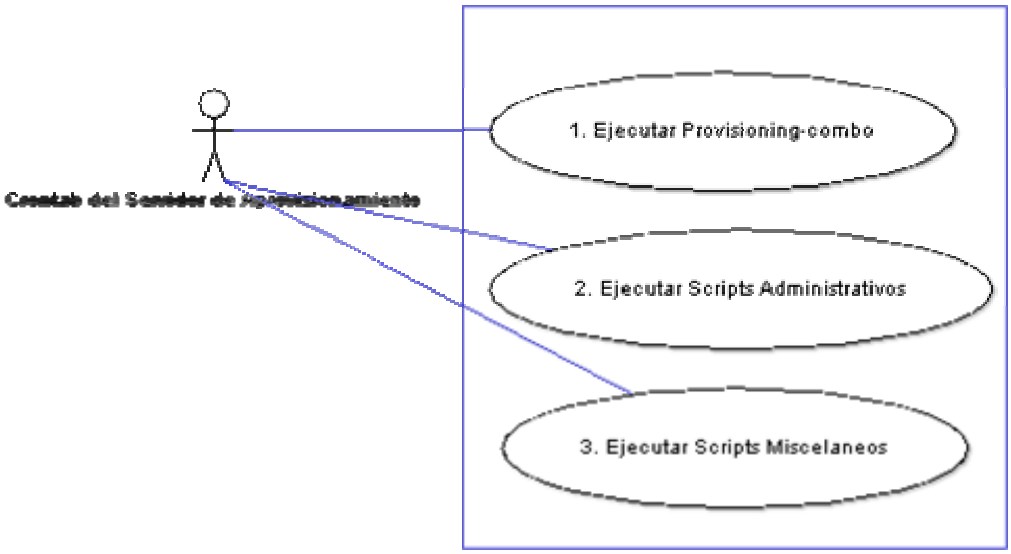

Figura 8. Actor y Casos de Usos Principales
Fuente: Elaborado por el autor (2007)

Así como el actor, en la Figura 8 se pueden observar los Casos de Uso principales, los cuales se describen brevemente a continuación:

1.- **Ejecutar Provisioning-combo:** Este caso de uso engloba todas aquellas actividades necesarias para poder realizar la auditoria de gestión sobre los equipos de aprovisionamiento de ABA.

2.- **Ejecutar Scripts Administrativos:** Este caso de uso engloba todas aquellas actividades adicionales que permiten el mantenimiento, respaldo, obtención de información requerida, etc. que garanticen la operatividad del servicio del aprovisionamiento de ABA.

3.- **Ejecutar Scripts Miscelaneos:** Este caso de uso engloba todas aquellas actividades adicionales que deben de ejecutarse previamente para garantizar el correcto funcionamiento del aprovisionamiento de ABA.

3.2.1.2 REFINAMIENTO DE LOS CASOS DE USO PRINCIPALES

Una vez definidos los casos de usos principales, se quiere modelar las diferentes situaciones de interacción que se presentan en cada uno de ellos, y de esta manera obtener un modelo de casos de uso que represente todos los escenarios posibles que se pueden presentar en la aplicación. En el modelo de Casos de Uso existen dos formas de representar las interacciones, las cuales son:

- *Relación de extensión (Extend):* Relación entre dos casos de uso. Esta relación es una asociación que describe un curso alterno opcional de otro caso de uso base.
- *Relación de Uso (Use o Include):* Relación entre dos casos de uso. Es una asociación que relaciona cursos fuertemente acoplados que permite aislar partes del caso de uso base. Una relación *include* es una relación que define una instancia de un Caso de Uso como un curso obligatorio de otro Caso de Uso.

A continuación se presenta el refinamiento de cada uno de los casos de uso principales, especificando la relación que existe entre cada uno de ellos con el caso base.

1.- Ejecutar Provisioning-combo: (Ver Figura 8)

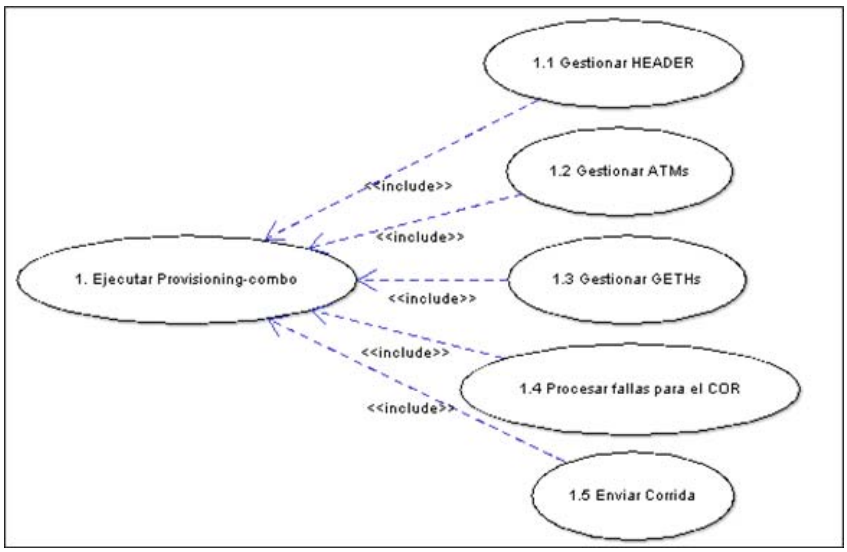

*Figura 9***.** Refinamiento del Caso de Uso "Ejecutar Provisioning-combo"

Fuente: Elaborado por el autor (2007)

A continuación se muestra el refinamiento a detalle de los Casos de Usos críticos mostrados en la Figura 9:

1.1 Gestionar HEADER: (Ver Figura 10)

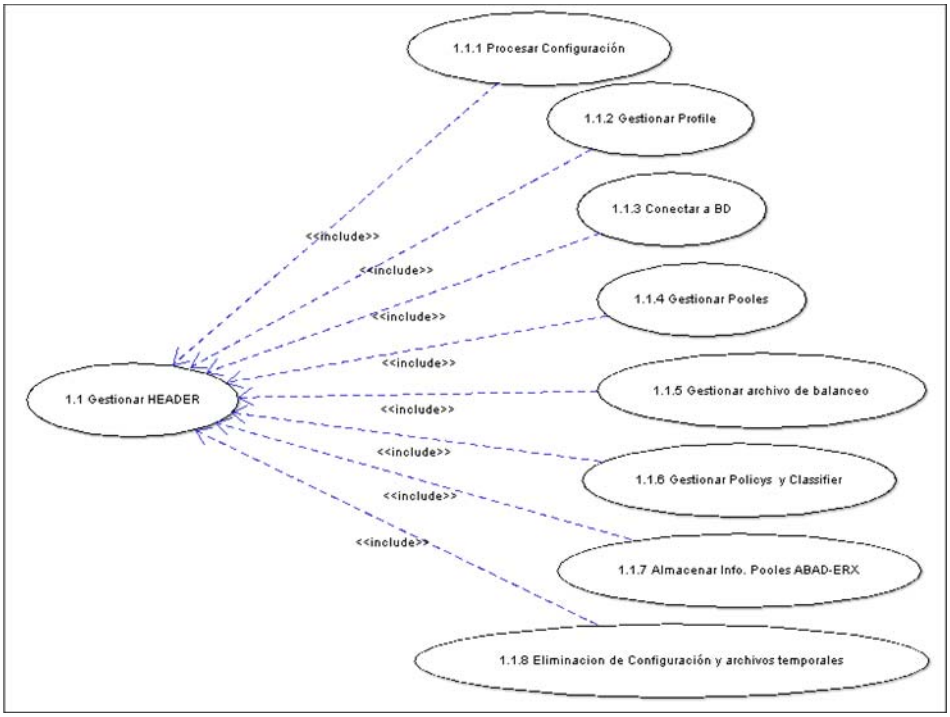

Figura 10. Refinamiento del Caso de Uso "Gestionar HEADER"

Fuente: Elaborado por el autor (2007)

1.1.1 Procesar Configuración: (Ver Figura 11)

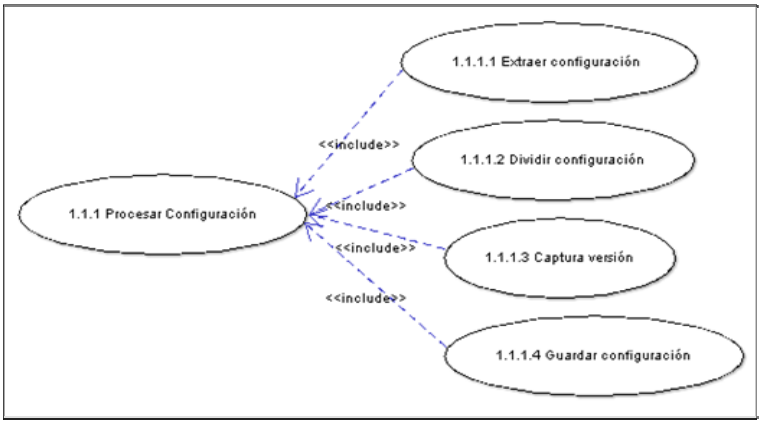

Figura 11. Refinamiento del Caso de Uso "Procesar Configuración"

Fuente: Elaborado por el autor (2007)

1.1.2 Gestionar Profile: (Ver Figura 12)

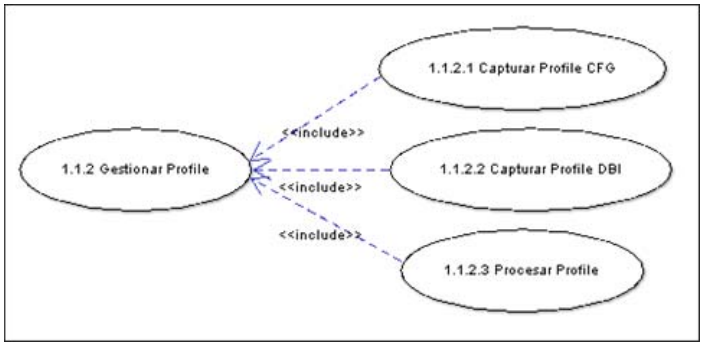

Figura 12. Refinamiento del Caso de Uso "Gestionar Profile"

Fuente: Elaborado por el autor (2007)

1.1.4 Gestionar Pooles: (Ver Figura 13)

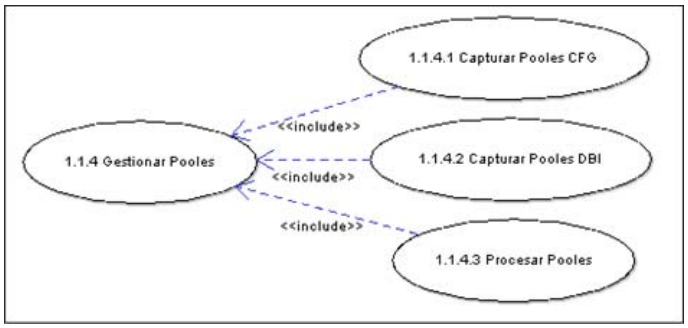

Figura 13. Refinamiento del Caso de Uso "Gestionar Pooles"

Fuente: Elaborado por el autor (2007)

1.1.5 Gestionar Archivo de Balanceo: (Ver Figura 14)

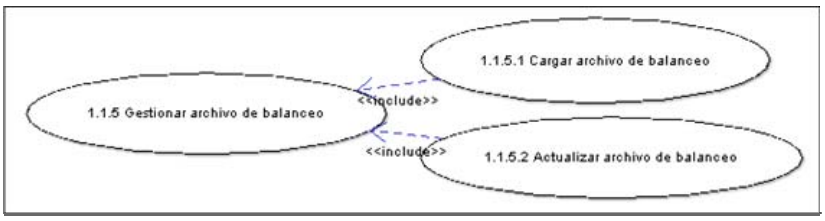

Figura 14. Refinamiento del Caso de Uso "Gestionar Archivo de Balanceo"

Fuente: Elaborado por el autor (2007)

1.1.6 Gestionar Archivo de Balanceo: (Ver Figura 15)

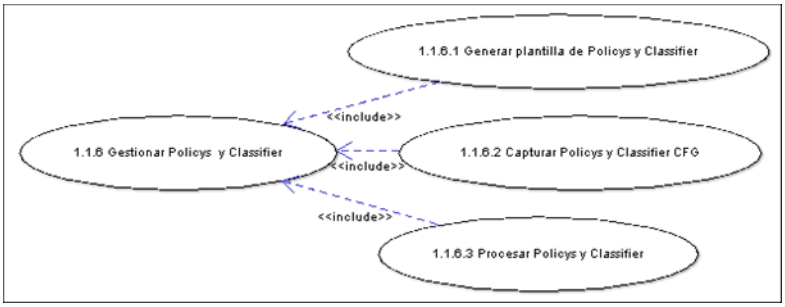

Figura 15. Refinamiento del Caso de Uso "Gestionar Policys y Classifier"

Fuente: Elaborado por el autor (2007)

1.2 Gestionar ATMs: (Ver Figura 16)

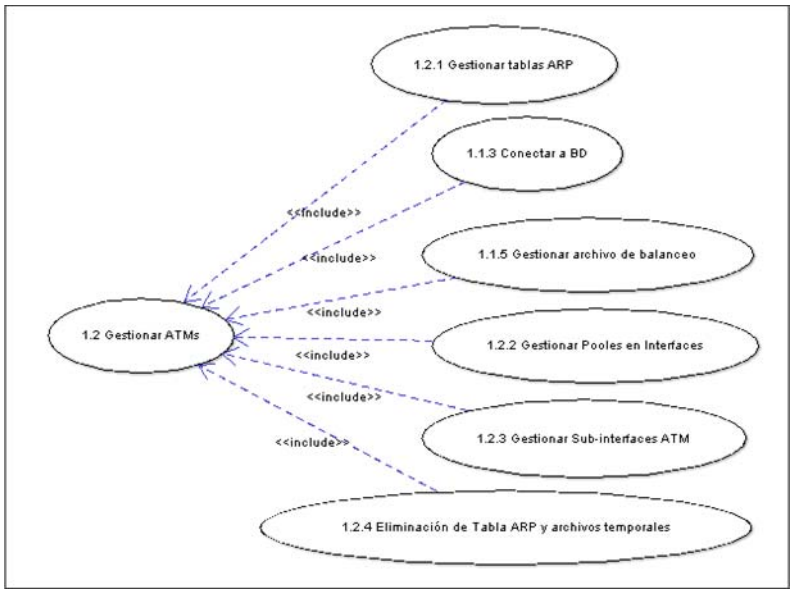

Figura 16. Refinamiento del Caso de Uso "Gestionar ATMs"

Fuente: Elaborado por el autor (2007)

1.2.1 Gestionar Tablas ARP: (Ver Figura 17)

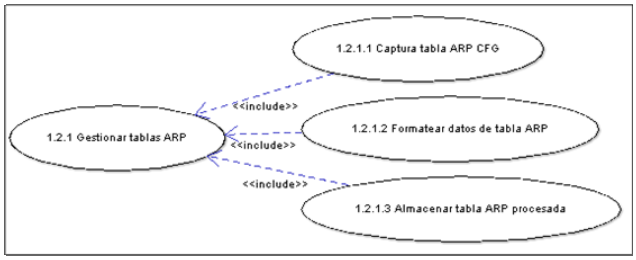

Figura 17. Refinamiento del Caso de Uso "Gestionar Tablas ARP"

Fuente: Elaborado por el autor (2007)

1.2.2 Gestionar Pooles en Interfaces: (Ver Figura 18)

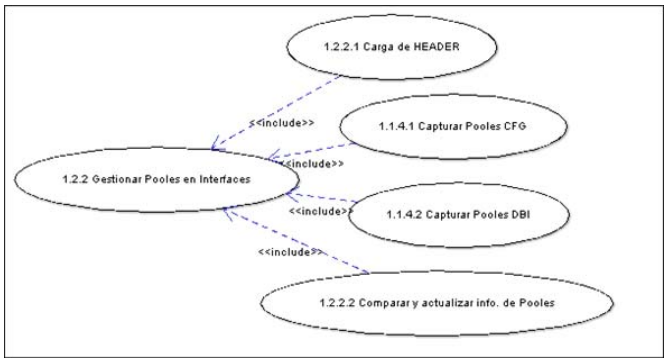

Figura 18. Refinamiento del Caso de Uso "Gestionar Pooles en Interfaces"

Fuente: Elaborado por el autor (2007)

1.2.3 Gestionar Sub-interfaces ATM: (Ver Figura 19)

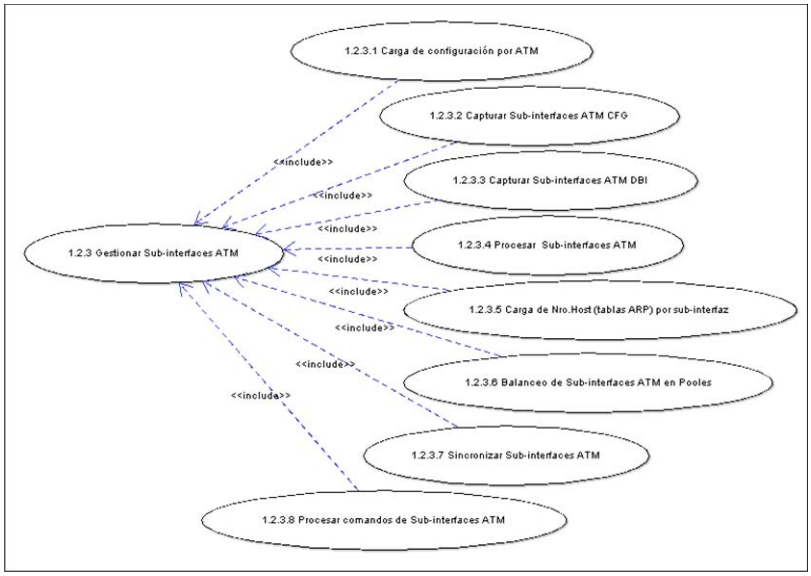

Figura 19. Refinamiento del Caso de Uso "Gestionar Sub-interfaces ATM"

Fuente: Elaborado por el autor (2007)

1.3 Gestionar GETHs: (Ver Figura 20)

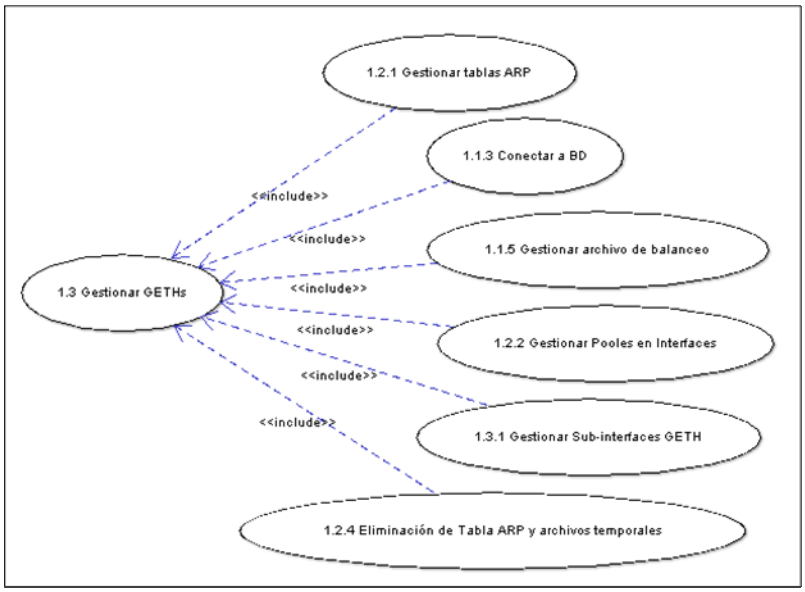

Figura 20. Refinamiento del Caso de Uso "Gestionar GETHs"

Fuente: Elaborado por el autor (2007)

1.3.1 Gestionar Sub-interfaces GETH: (Ver Figura 21)

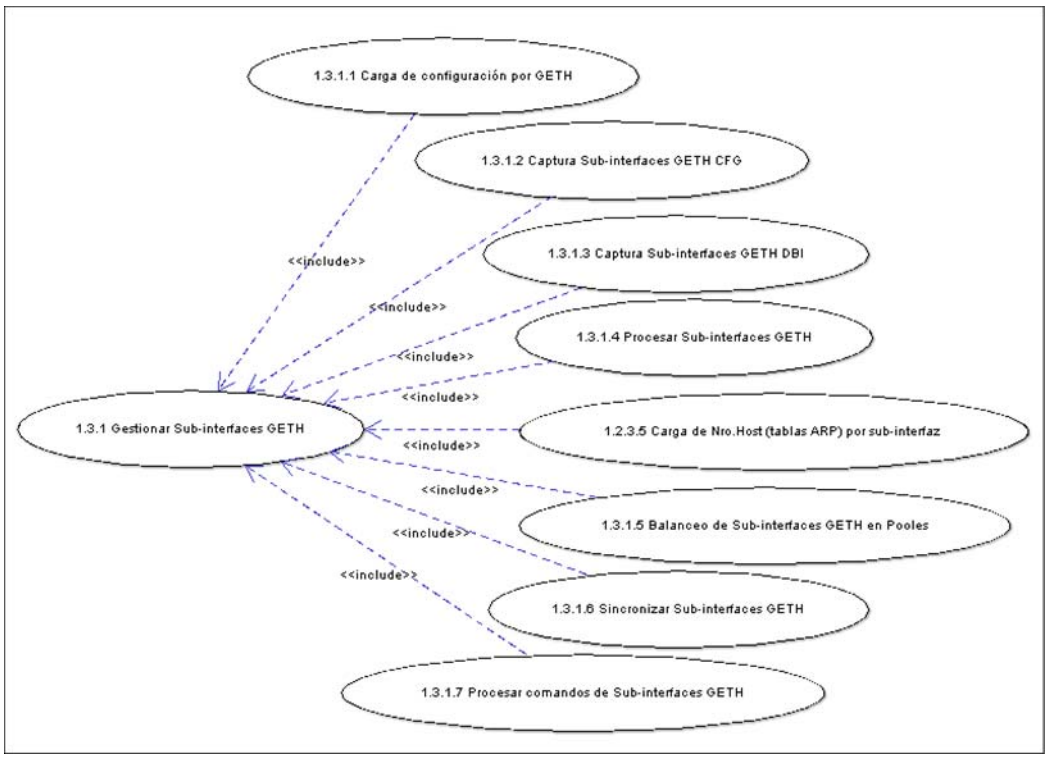

Figura 21. Refinamiento del Caso de Uso "Gestionar Sub-interfaces GETH"

Fuente: Elaborado por el autor (2007)

A continuación se muestra el refinamiento a detalle del resto de Casos de Usos mostrado en la Figura 8:

2.- Ejecutar Scripts Administrativos: (Ver Figura 22)

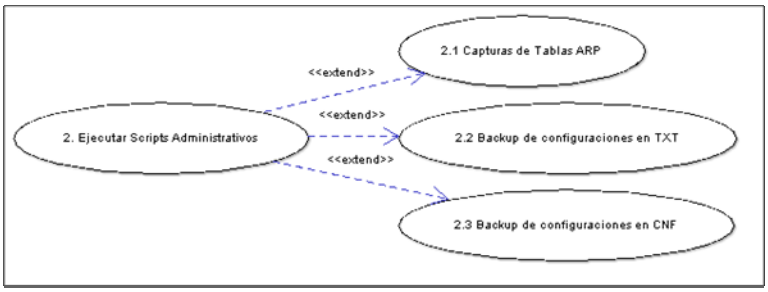

Figura 22. Refinamiento del Caso de Uso "Ejecutar Scripts Administrativos"

Fuente: Elaborado por el autor (2007)

3.- Ejecutar Scripts Miscelaneos: (Ver Figura 23)

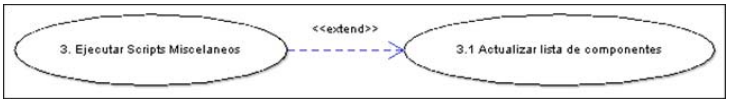

Figura 23. Refinamiento del Caso de Uso "Ejecutar Scripts Miscelaneos"

Fuente: Elaborado por el autor (2007)

3.2.2 DIAGRAMA DE COMPONENTES DE PROVISIONING-COMBO

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Las relaciones de dependencia se utilizan en los diagramas de componentes para indicar que un componente utiliza los servicios ofrecidos por otro componente.

El diagrama de componente hace parte de la vista física de un sistema, la cual modela la estructura de implementación de la aplicación por sí misma, su organización en componentes y su despliegue en nodos de ejecución. Esta vista proporciona la oportunidad de establecer correspondencias entre las clases y los componentes de implementación y nodos. La vista de implementación se representa con los diagramas de componentes. (Rumbaugh, J; Jacobson, I. et al., 2000). A continuación se muestra el Diagrama de Componentes (Ver Figura 24) del sistema analizado:

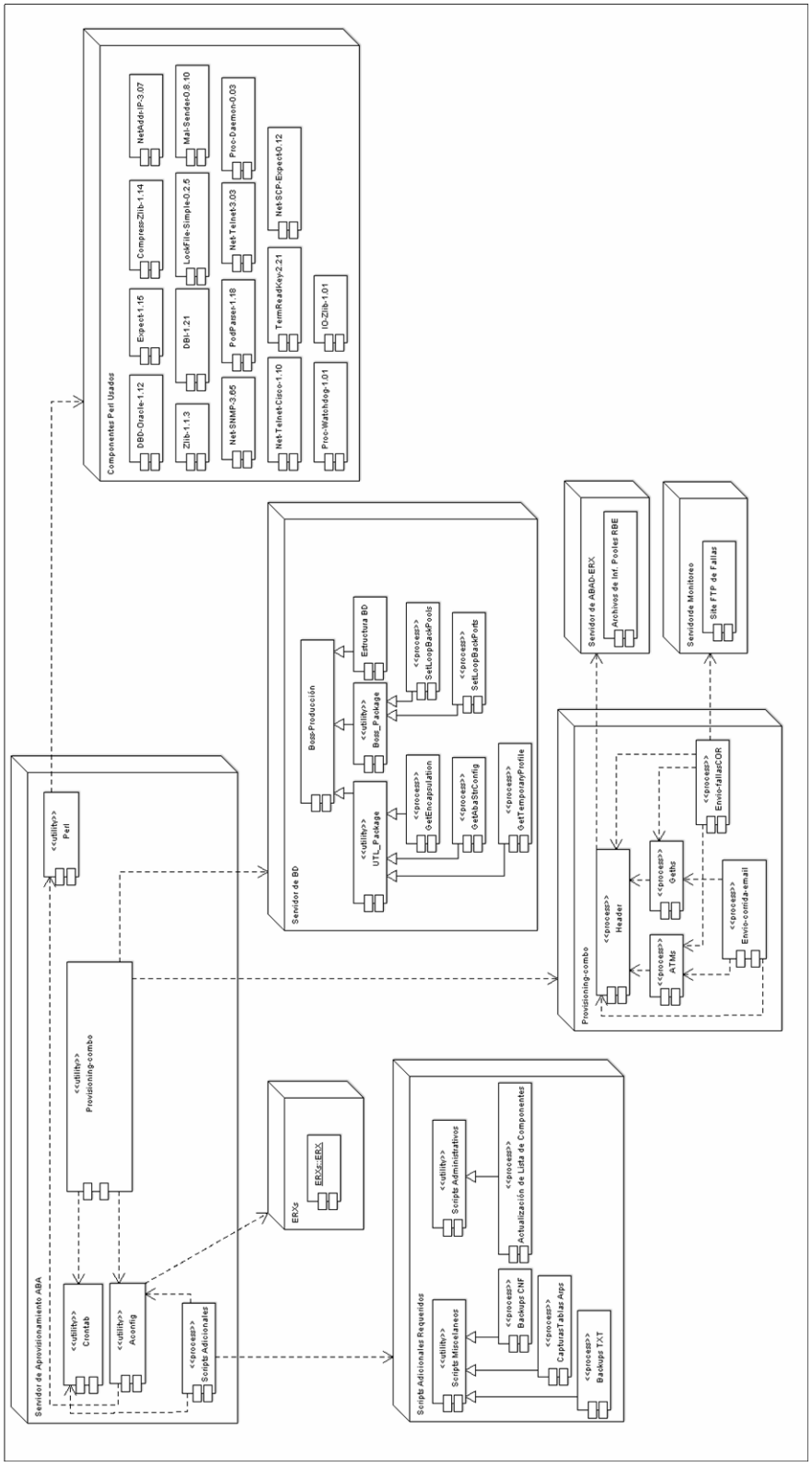

Figura 24. Diagrama de Componentes de Provisioning-combo

Fuente: Elaborado por el autor (2007)

3.3 ESPECIFICACIONES UML PARA LA ABSTRACCIÓN DE LOS DATOS

En base a la metodología de la ingeniería inversa de software para la abstracción de datos se obtiene la información de dos puntos claves:

- Estructuras de datos internas, que permite agrupar variables de programa que estén relacionadas.
- Estructuras de base de datos, permiten definir objetos de datos, y apoyan los métodos de establecer relaciones entre objetos, para definir este modelo de datos.

En esta sección el foco fundamental es determinar todas las estructuras de datos disponibles en el sistema (Internas y de Base de Datos) y sus relaciones. Se realizó el diagrama de UML que permite modelar la vista estática de los sistemas, este es el Diagrama de Clases, con el cual, se generan la mayoría de las declaraciones de estructuras de datos en un programa, el cual, fue llevado a cabo y esta expuesto en la sección 3.3.1 este capitulo.

Según lo que define la metodología ingeniería inversa en este punto, el diseño del diagrama UML mencionado anteriormente, permite obtener la abstracción de datos necesaria para la comprensión y documentación de este aspecto del sistema, por tanto, se muestra a continuación el desarrollo de este diagrama.

3.3.1 DIAGRAMA DE CLASES DE PROVISIONING-COMBO

El Diagrama de Clases es el diagrama principal para el análisis y diseño. Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye definiciones para atributos y operaciones.

El modelo de casos de uso aporta información para establecer las clases, objetos, atributos y operaciones. El mundo real puede ser visto desde abstracciones diferentes (subjetividad).

Relaciones entre clases:

- *Asociación:* Es una relación que describe un conjunto de vínculos entre clases. Pueden ser binarias o n-arias, según se implican a dos clases o más. Las relaciones de asociación vienen identificadas por los roles, que son los nombres que indican el comportamiento que tienen los tipos o las clases, en el caso del rol de asociación (existen otros tipos de roles según la relación a la que identifiquen) indican la información más importante de las asociaciones. Existe una forma especial de asociación, la agregación, que especifica una relación entre las clases donde el llamado "agregado" indica él todo y el "componente" es una parte del mismo.
- *Composición:* Es un tipo de agregación donde la relación de posesión es tan fuerte como para marcar otro tipo de relación. Las clases en UML tienen un tiempo de vida determinado, en las relaciones de composición, el tiempo de vida de la clase que es parte del todo (o agregado) viene determinado por el tiempo de vida de la clase que representa el todo, por tanto es equivalente a un atributo, aunque no lo es porque es una clase y puede funcionar como tal en otros casos.
- *Generalización:* Cuando se establece una relación de este tipo entre dos clases, una es una Superclase y la otra es una Subclase. La subclase comparte la estructura y el comportamiento de la superclase. Puede haber más de una clase que se comporte como subclase.
- *Dependencia:* Una relación de dependencia se establece entre clases (u objetos) cuando un cambio en el elemento independiente del modelo puede requerir un cambio en el elemento dependiente.

En términos generales este diagrama muestra la vista estática de un sistema a través de un conjunto de clases, interfaces y colaboraciones junto con sus relaciones. (Rumbaugh, J; Jacobson, I. et al., 2000).

Para llevar a cabo este modelo se realizaron los siguientes pasos:

- Identificar las clases y sus atributos.

- Identificar las operaciones.
- Identificar las relaciones entre clases.

A continuación se muestra el Diagrama de Clases (Ver Figura 25) diseñado:

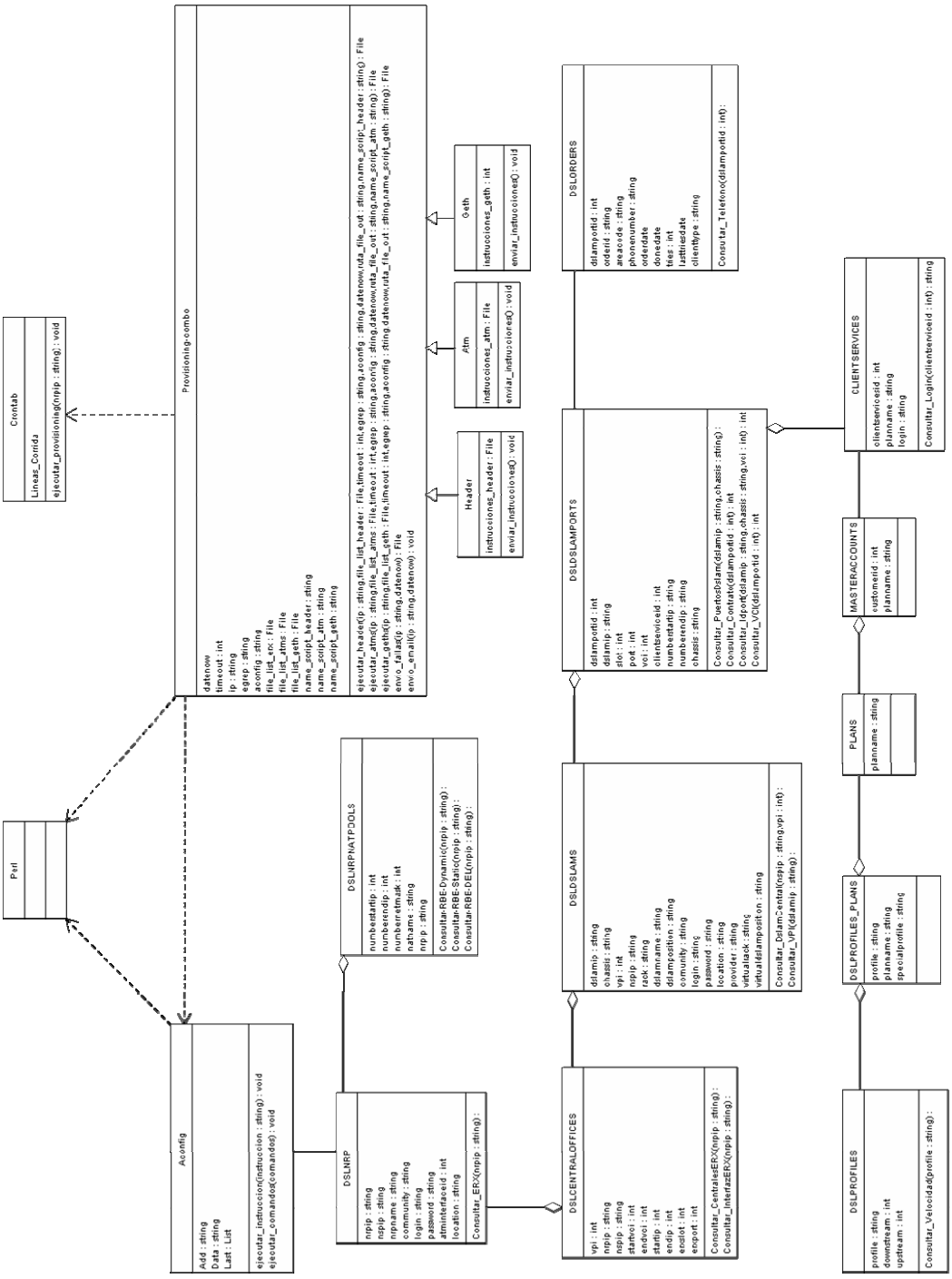

Figura 25. Diagrama de Clases de Provisioning-combo

Fuente: Elaborado por el autor (2007)

3.4 ESPECIFICACIONES UML PARA LA ABSTRACCIÓN DE LA INTERFAZ DE USUARIO

Para esta especificación de la ingeniería inversa de software se debe enfocar la toma de información que permita especificar la estructura y comportamiento de la interfaz usuario ya existente.

La ingeniería inversa tiene como base para la obtención de esta abstracción el modelo de comportamiento, que conlleva a identificar la interfaz externa y la parte interna del comportamiento general del sistema.

Para realizar esta abstracción se debe comenzar por obtener una vista integral del comportamiento del sistema, centrado en la historia de la vida de un objeto, obtenidos de las clases ya identificadas en el punto 3.3 de este capitulo, por ende, se tomo como base los diagramas de UML que permiten modelar vistas de interacción de sistema, existen dos opciones para llevar a cabo estas vistas, las cuales son el Diagrama de Secuencia (centrado en los objetos individuales) y el Diagrama de Colaboración (centrado en los objetos cooperantes), para el caso de este sistema se escogió realizar uno solo de ellos, el cual fue el Diagrama de Secuencia, el mismo fue llevado a cabo y esta expuesto en la sección 3.4.1 de este capitulo.

Posteriormente, se requería complementar esta abstracción con un modelo que representara como es el proceso de negocio del sistema y así ver el comportamiento del sistema a nivel más de flujos de datos que de pases de mensajes entre los objetos del sistema involucrado.

En UML existen dos diagramas adicionales a los ya mencionados que permiten modelar vistas de comportamiento que son: el Diagrama de Actividades (modela el comportamiento mediante flujos de actividades) y el Diagrama de Estado (modela el comportamiento de acuerdo con eventos), para el caso de este sistema se escogió realizar uno solo de ellos, el cual fue el Diagrama de Actividades, el mismo fue llevado a cabo y esta expuesto en la sección 3.4.2 de este capitulo.

Con el uso de los dos diagramas UML mencionados anteriormente, se logra obtener la abstracción de comportamiento necesaria para la comprensión y documentación de este aspecto del sistema, permitiendo con esto cumplir con los lo requerido por la metodología de ingeniería inversa en estas especificaciones, por tanto, se muestran a continuación el desarrollo de ambos diagramas.

3.4.1 DIAGRAMA DE SECUENCIA DE PROVISIONING-COMBO

Este diagrama muestra las interacciones entre los objetos organizadas en una secuencia temporal. En particular muestra los objetos participantes en la interacción y la secuencia de mensajes intercambiados entre ellos.

A diferencia de los diagramas de colaboración, los diagramas de secuencia incluyen secuencias temporales pero no incluyen las relaciones entre objetos. Pueden existir de forma de descriptor (describiendo todos los posibles escenarios) y en forma de instancia (describiendo un escenario real).

Dentro del conjunto de mensajes representados dispuestos en una secuencia temporal, cada rol en la secuencia se muestra como una línea de vida, es decir, una línea vertical que representa el rol durante cierto plazo de tiempo, con la interacción completa. Los mensajes se muestran como flechas entre líneas de vida. Un diagrama de secuencia puede mostrar un escenario, es decir, una historia individual de transacción. Un diálogo de secuencia posee dos dimensiones: la vertical representa el tiempo, la horizontal representa los objetos que participan en la interacción. El uso general de un diagrama de secuencia es mostrar la secuencia del comportamiento de un caso de uso.

En términos generales estos son los componentes claves de este diagrama y con esta lógica permiten lograr su objetivo principal que son mostrar los objetos, el pase de mensajes y los intervalos de tiempo existentes para la ejecución de los estos. (Rumbaugh, J; Jacobson, I. et al., 2000).

A continuación se muestra el Diagrama de Secuencia (Ver Figura 26) del sistema:

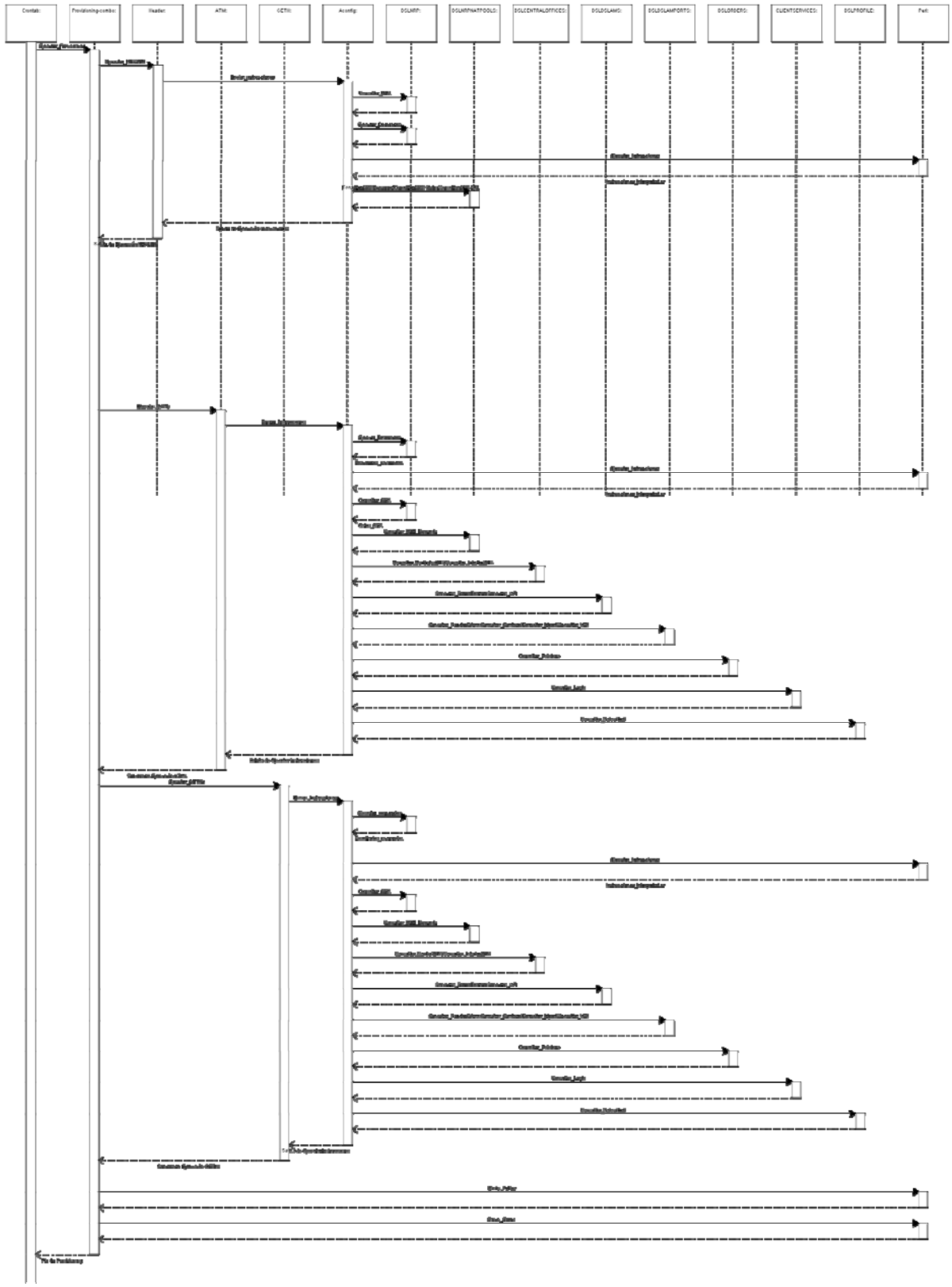

Figura 26. Diagrama de Secuencia de Provisioning-combo

Fuente: Elaborado por el autor (2007)

3.4.2 DIAGRAMA DE ACTIVIDADES DE PROVISIONING-COMBO

El diagrama de actividades es una especialización del diagrama de estado, organizado respecto a las acciones y usado para especificar: un método, un caso de uso o un proceso de negocio (Workflow).

Un estado de actividad representa una actividad: un paso en el flujo de trabajo o la ejecución de una operación. Un grafo de actividades describe grupos secuenciales y concurrentes de actividades. Las actividades se enlazan por transiciones automáticas. Cuando una actividad termina se desencadena el paso a la siguiente actividad.

Un diagrama de actividades es provechoso para entender el comportamiento de alto nivel de la ejecución de un sistema, sin profundizar en los detalles internos de los mensajes. Los parámetros de entrada y salida de una acción se pueden mostrar usando las relaciones de flujo que conectan la acción y un estado de flujo de objeto.

Un grafo de actividades contiene estados de actividad que representa la ejecución de una secuencia en un procedimiento, o el funcionamiento de una actividad en un flujo de trabajo. En vez de esperar un evento, como en un estado de espera normal, un estado de actividad espera la terminación de su cómputo. Cuando la actividad termina, entonces la ejecución procede al siguiente estado de actividad dentro del diagrama. Una transición de terminación es activada en un diagrama de actividades cuando se completa la actividad precedente.

Un diagrama de actividades puede contener bifurcaciones, así como divisiones de control en hilos concurrentes. Los hilos concurrentes representan actividades que se pueden realizar concurrentemente por los diversos objetos o personas. La concurrencia se representa a partir de la agregación, en la cual cada objeto tiene su propio hilo. Las actividades concurrentes se pueden realizar simultáneamente o en cualquier orden.

En resumen, se puede decir que un diagrama de actividades es como un organigrama tradicional, excepto que permite el control de concurrencia además del control secuencial. (Rumbaugh, J; Jacobson, I. et al., 2000). A continuación se muestra el Diagrama de Actividades (Ver Figura 27) del sistema:

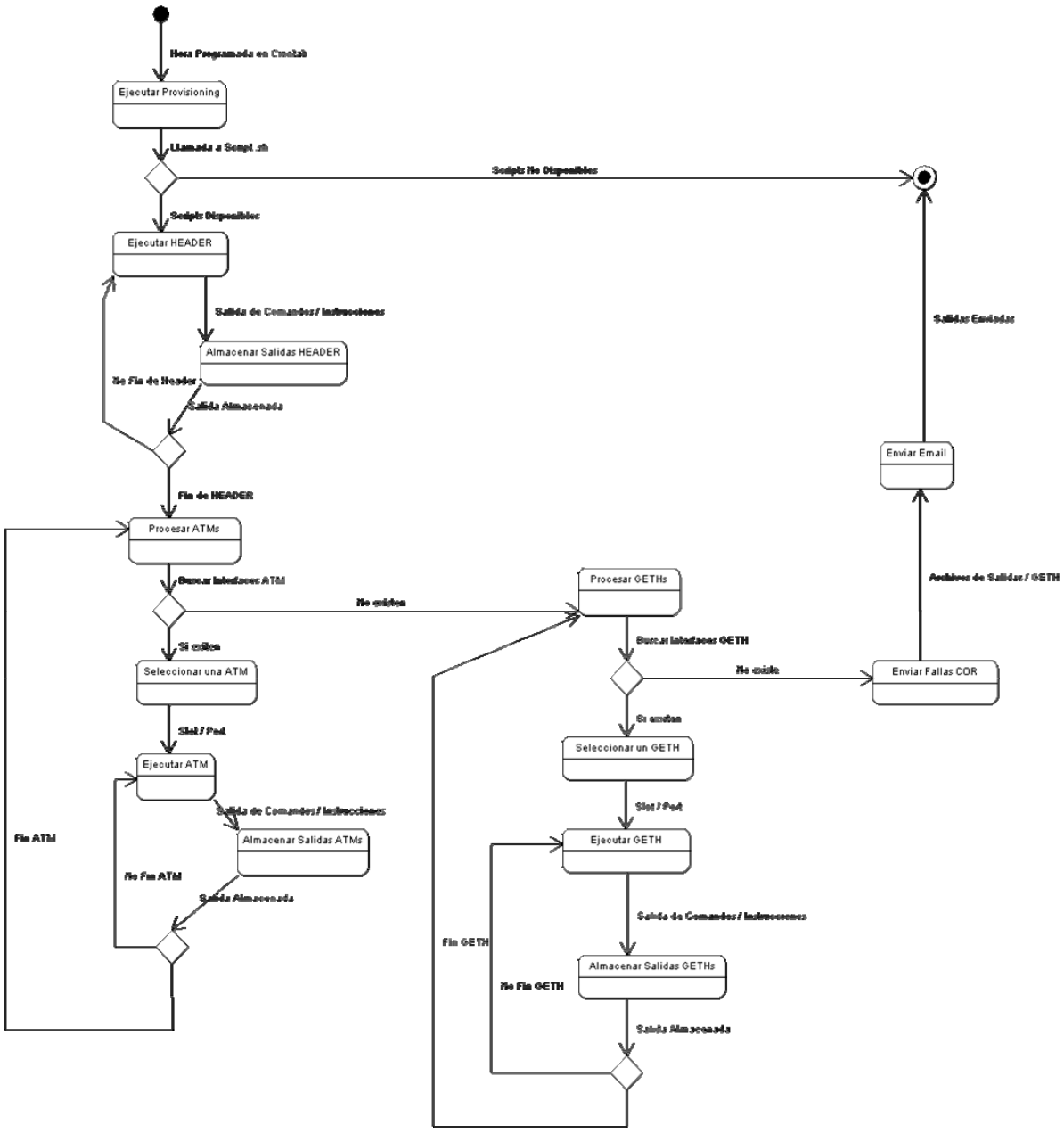

Figura 27. Diagrama de Actividades de Provisioning-combo

Fuente: Elaborado por el autor (2007)

CONCLUSIONES Y RECOMENDACIONES

Como se ha planteado a lo largo de este trabajo presentado, la Ingeniería Inversa, como indica su nombre, hace referencia al proceso contrario al que se realiza habitualmente en la denominada Ingeniería del Software.

En la Ingeniería del Software (ingeniería directa), el objetivo es modelar la realidad en pasos sucesivos, que descienden en el nivel de abstracción, hasta llegar a una representación computacional que resuelva el problema planteado. Se comienza con especificaciones de requisitos en lenguaje natural (o algún lenguaje más formal de muy alto nivel), pasando por modelos conceptuales expresados en lenguajes de modelado (como UML), diseños de arquitectura y modelos de diseño detallado hasta tener finalmente una implementación escrita en un lenguaje de programación y, por lo tanto, ejecutable.

El planteamiento inicial de este trabajo fue tomar un caso real (sistema desarrollado sin ninguna documentación formal ni especificación alguna, solo con el código fuente) y mostrar que aplicando Ingeniería Inversa se obtiene el mismo resultado que aplicando Ingeniería de Software en el desarrollo, un sistema codificado y con el conjunto de especificaciones documentadas de procesamiento, datos e interfaz expresadas en un lenguaje de modelado estándar, en este caso la aplicación de los diversos diagramas UML aplicados a software orientados a objetos.

Las diversas abstracciones aplicadas partiendo del código fuente y lo definido por la Ingeniería Inversa permitieron llegar al objetivo, es decir, obtener el detalle de las especificaciones formales del sistema aplicando una herramienta que permitiera plasmar toda la información de la aplicación original, este resultado final es de vital importancia, ya que, además de cubrir uno de los objetivos propuestos por el Coordinación de Aprovisionamiento de Servicios de la Gerencia de Ingeniería, Desarrollo y Construcción TI/SI, va a ser una base para llevar a cabo

futuras mejoras o cambios en este sistema de manera mas efectiva, uno de los aspectos vitales que obtendrá mejoras significativas son los futuros mantenimientos y la reingeniería de alguno de los aspectos de este sistema (procesos, datos o interfaz), ya que estas especificaciones obtenidas permitirán medir alcances y puntos de acción de una manera rápida y mas real.

Otro de los aspectos alcanzados es la existencia de documentación, ya que con ella se logra que la las aplicaciones no sean dependientes de personas especificas, sino que con el uso de este material puede ser distribuido al personal de la corporación que lo requiera y asegura el conocimiento rápido a los integrantes de la Coordinación, disminuyendo los tiempos de entrenamiento y conocimiento. Además, el conocimiento de la documentación permite incrementar la participación del usuario, en actividades como el modelado formal de interfaces, procesos y modelos de datos facilitando la reingeniería, los mantenimientos, permitiendo la simulación, aumento de calidad, disminución de costos de desarrollo, incremento del cumplimiento de los requisitos de los clientes, las cuales son tareas claves.

En cuanto a las investigaciones realizadas sobre la Ingeniería Inversa se logro observar que la mayoría de referencias, investigaciones o trabajos consultados dan un enfoque a este concepto solo por una vertiente actual, que es el usado por los piratas o hackers de software para lograr obtener especificaciones detalladas de software privados para conocer posibles vulnerabilidades o copiar funcionalidades de los mismos, lo cual, no permite visualizar el buen uso y practicidad de lo que propone la Ingeniería Inversa para el desarrollo de software, según lo presentado en este trabajo se puede observar un caso real que facilito la cobertura de deficiencias de desarrollo existentes y permitió solucionar una necesidad corporativa existente.

Sin embargo, la Ingeniería Inversa no es ignorada del ámbito del desarrollo de software por los especialistas, un ejemplo al respecto, es la existencia en diversas

herramientas CASE de funcionalidades que permiten hacer Ingeniería Inversa de software, con la limitación del lenguaje de programación en que esta escrito el código fuente (No maneja todos los lenguajes Orientados a Objetos sino los mas comerciales), en el caso del lenguaje de programación estudiado que es Perl no era manejado por dichas herramientas. Otro punto es que existen métodos o programas formales definidos por diversos autores para la aplicación de esta Ingeniería de Software, uno de estos es el programa *eLib* (Tonella P. y Potrich A (2005)) que fue consultado como referencia para este desarrollo.

Como recomendaciones encontradas a lo largo del desarrollo de este trabajo, se pueden plantear las siguientes:

- Aplicar la Ingeniería Inversa expuesta en el resto de desarrollos de software existentes en el departamento y considerar las prácticas de documentación y diseño de especificaciones de software como requisito indispensable en los futuros desarrollos planteados.
- Hacer extensible el conocimiento de todo lo desarrollado en la Coordinación a todos los miembros del mismo basándose en los resultados de los documentos resultados de la Ingeniería Inversa.
- Plantear una visión diferente de la Ingeniería Inversa a lo planteado de este concepto en la actualidad haciendo buen uso de la misma.
- Desarrollar más trabajos de investigación en este aspecto para expandir el conocimiento público de esta área de la Ingeniería del Software, la cual, es vital para hacer Reingeniería de Software efectiva.
- Mantener actualizadas las documentaciones de especificaciones de software ante posibles requerimientos nuevos o de continuidad de los diversos software ya documentados, así como la aplicación del manejo de versiones de dichos documentos.

REFERENCIAS

Bauer, F. (1972). Software Engineering, Information Processing, 71, North Holland Publishing Co., Amsterdam.

Bestalanffy, V. (1968). General system theory. New York: Braziller.

Boehm, B.(1988). A Spiral Model of Software Development and Enhancement. *IEEE Computer*. 21(5). Pp 61-72. 1988.

Booch, G. (1994) Object-Oriented Analisys and Desing with Aplications. Benjamin/Cummings company. 1994.

Booch, G; Rumbaugh, J. et al. (1999) The Unified Modeling Language User Guide. Addison-Wesley. Versión castellana “El Lenguaje Unificado de Modelado”. Addison-Wesley.

Breuer, P., y Lano, K.; (1991) Creating Specification From Code: Reverse-Engineering Techniques, Journal of Software Maintenance: Research and Practice, vol 3, Wiley, pp 145-146.

Briand, L. et al. (2003) Towards the Reverse Engineering of UML Sequience Diagrams. Proceedings of the 10th Working Conference in Reverse Engineering, IEEE Computer Society.

Bunge, M. (1979). Treatise on Basic Philosophy: Vol.4, Ontology II: A World of Systems - Reidel, Dordrecht, Holland.

Chikofsky, E. et al. (1990) "Reverse Engineering and Desing Recovery: A Taxonomy." IEEE Software(January): 13-17.

DeMarco, T. (1978). Structured analysis and system specification. Englewood Cliffs, NJ: Prentice Hall. xiv, 352 pages.

Di Lucca, G. et al. (2001) An Approach for Reverse Engineering of Web-Based Applications. Proceedings of the Eighth Working Conference On Reverse Engineering (WCRÉ01), IEEE Computer Society.

Flower M. y Scott, K. (1997). UML Destilled: Applying the Standard Object Modelling Language. Reading, MA: Addison-Wesley. (Caps. 1, 6)

IEEE, (1993). IEEE: Standards Colletion: Software Engineering, IEEE Standard 610.12-1990.

IEEE, (1997). Institute of Electrical & Electronics Enginee; 6th edition; Abril 1, Paperback.

Jackson, M. A. (1983) System Development. Londres: Prentice-Hall.

Jacobson, I. (1992) Object-Oriented Software Engineering. A Use Case Driven Aprroach. Addison Wesley Publishig company.

James Rumbaugh, Ivar Jacobson y Grady Booch (2000) El Lenguaje Unificado de Modelado. Manual de Referencia. Addison Wesley.

Laudon, K. y Laudon, J., (1997) Essentials of Management Information Systems. Organization and Technology. Second Edition. Prentice Hall. New Jersey, USA.

Merlo, E. et al. (1993) Reverse Enginneering of User Interfaces, Proc. Working Conference on Reverse Engineering, IEEE, Baltimore, MD, Mayo.

Novativa (1996) Anónimo, “Si los programadores fueran albañiles…” nº 124, noviembre-diciembre, http://www.124/if124.html

OMG. (2005). Object Management Group “OMG Unified Modeling Language Specification” Version 1.5, http://www.uml.org/, Enero.

Pérez at el. 2002 “Desarrollo de los Módulos Promociones y Administrativo-Ingresos de SIGEPOST: Sistema de Gestión para la Coordinación de Postgrado de la Facultad de Ciencias de la UCV”. Tesis de Grado. Octubre 2002.

Piattini, M. (1996). Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión. Rama. Madrid.

Piattini, M. et al., (2000) Mantenimiento del Software: Modelos, Técnicas y Métodos para la Gestión del Cambio. Spain, RA-MA.

Pigoski, T. M. (1997). Practical Software Maintenance. Best Practices for Managing your Investment. Estados Unidos, John Wiley & Sons.

Premerlani, W.J., M.R Blaha (1994). An Approach for Reverse Engineering of Relational Database, CACM. Vol. 37, nº 5, Mayo, pp. 42-49.

Pressman, R.S. (2002). “Ingeniería del Software. Un enfoque práctico” (Adaptado por Darrel Ince). 5ª ed. Mc Graw Hill, Interamericana de España S.A.U.

Real academia española. (2005) Diccionario de la Real Academia Española. http://www.rae.es/.

Riva, C. et al., (2002) Generation of Architetural Documentation using XML. Proceedings of the Ninth Conference on Reverse Engineering (WCRÉ02), IEEE Computer Society.

Rodríguez, A. et al., (2001) Gestión de la evolución del software. El eterno problema de los legacy systems. JISBD2001. VI Jornadas de Ingeniería del Software. Universidad de Granada. España. 2001.

Ros, Joaquin (2005) Fundamentos de Ingeniería del Software. Facultad de Informática. Universidad de Murcia Campus de Espinardo. 30071 - Espinardo - Murcia (Spain). http://dis.um.es/~jnicolas/

Rumbaugh, J. (1991). Object Oriented Modeling and Design, Prentice-Hall.

Rumbaugh, J. (1995). OMT: The Development Process. Journal of Object-Oriented Programing (JOOP), Mayo, Vol 8, Nro 1.

Rumbaugh, J.; Jacobson I. et al. (1999a). The Unified Modeling Language Reference Manual. Addison-Wesley.

Rumbaugh, J.; Jacobson I. et al. (1999b). The Unified Software Development Process. Addison-Wesley.

Tonella P. y Potrich A. (2005) Reverse Engineering of Object Oriented Code. Springer, 2005, ISBN 0-387-40295-0

Saussure, F. (1931). Grundfragen der sprachwissenschaft. herausgegeben von charles bally und albert sechehaye unter mitwirkung von albert riedlinger. übersetzt von herman lommel -- Reliure inconnue.

Sommerville, I. (2002) “Ingeniería del Software”. 6ta ed. Addison-Wesley, Pearson Educación, México.

Soriano, A. y Zambrano, N. et al. (2000). Curso de Ingeniería de Software UCV, 2000.

Wikipedia. (2005). Wikipedia, La enciclopedia libre, http://www.wikipedia.org/.

ANEXOS

ANEXO A

Producto Acceso Banda Ancha (ABA)

ANEXO A. PRODUCTO DE ACCESO BANDA ANCHA (ABA)

Entre los servicios de valor agregado que ofrece la corporación CANTV a sus clientes es el producto Acceso Banda Ancha (ABA), el cual ofrece a sus clientes un experiencia de navegación, que se traduce en beneficios como mayor velocidad y disponibilidad.

ABA funciona sobre una tecnología llamada Línea de Abonado Digital Asimétrica (ADSL, Assymetric Digital Subscriber Line), que le permite a CANTV aprovechar la infraestructura sobre la que actualmente funciona el servicio telefónico (Pares de cobre), para ofrecer acceso a Internet a velocidades bastante superiores a las de un acceso discado, mejor conocido Dial Up o discado.

La tecnología ADSL pertenece a una familia de tecnologías conocidas como XDSL. El adjetivo de asimétrico indica que las velocidades de envío y recepción de información no son idénticas. ADSL permite compartir una porción del rango de frecuencias que pasa por el par de cobre telefónico sin interferir con el servicio de voz ya que las frecuencias utilizadas son distintas.

Podemos imaginar el par de cobre telefónico como una autopista de varios canales donde uno de ellos es utilizado para el viaje de voz, otro es utilizado para enviar datos y otro para la recepción. En el caso de los canales de datos el de recepción es mucho más grande que el de envío. Esto es así porque la tecnología ADSL fue pensada para el uso de Internet donde la cantidad de información recibida generalmente es mayor que la enviada.

A continuación se muestra gráficamente los elementos que componen una conexión ADSL, para cualquier información adicional consultar la pagina http://www.cantv.net

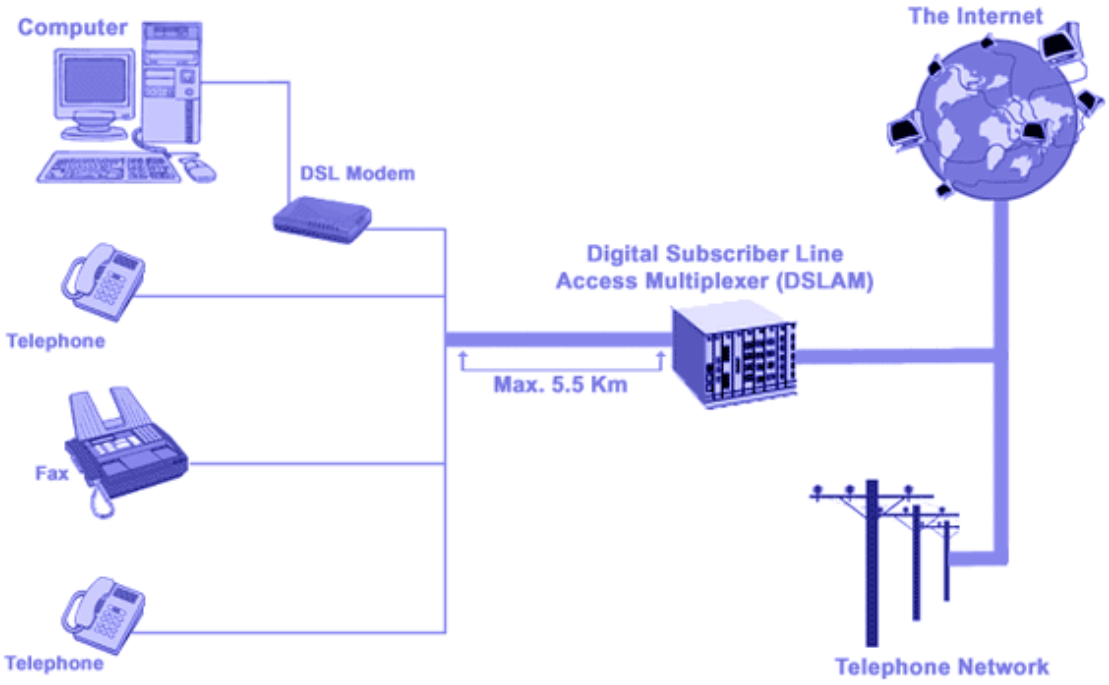

Figura 1. Servicio ADSL

ANEXO B

Administración de Plataforma de Aprovisionamiento ABA (Provisioning-combo)

ANEXO B. ADMINISTRACIÓN DE LA PLATAFORMA DE APROVISIONAMIENTO ABA (PROVISIONING-COMBO)

Gerencia de Ingeniería de Desarrollo y Construcción de Sistemas TI y SI / Gerencia de Ingeniería de Soluciones /Coordinación de Ingeniería de Aprovisionamiento de Servicios

Febrero 2006

Administración de Plataforma de Aprovisionamiento ABA (Provisioning-combo)

Tabla de Contenido

Información General

	Nombre	Unidad	Firma	Fecha
Elaborado por:	Janeth C. Pérez A.	Coordinación de Ingeniería de Aprovisionamiento de Servicios		
Revisado por:				
Aprobado por:				

Código del Documento	**INS-XXXX**
Edición	02
Fecha	XX-XX-XXXX
Tipo de Documento	Instructivo
Nombre del Archivo	INS-XXX-Administracion-Aprovisionamiento-ABA(Provisioning-combo).doc
Área de Gestión	
	Prohibida la reproducción total o parcial de este documento, sin autorización. Propiedad de CANTV ©

Objetivo General

Establecer los lineamientos para la administración, operación y troubleshooting de los sistemas para el aprovisionamiento ABA (Provisioning-combo[1]).

[1] Provisioning-combo: Nombre del script auditor que corre en las madrugadas sobre los agregadores de la plataforma ABA.