UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"HERRAMIENTA PARA LA VINCULACIÓN DEL ANÁLISIS DE LA TAREA DEL USUARIO CON EL PROCESO UNIFICADO"

TESIS

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA (COMPUTACIÓN)

P R E S E N T A:

GUILLERMO DOMÍNGUEZ OSORNO

DIRECTOR DE TESIS:
DR. FERNANDO GAMBOA RODRÍGUEZ

México, D.F. 2003.

Gracias

A mis padres por su apoyo toda mi vida
A Fernando Gamboa por su guia en este proceso
A mis compañeros por su ayuda en la maestría
A mis amigos por su amistad para siempre
A Alma por su Amor y comprension
A mi familia política por el apoyo en todo momento

INDICE

1 Introducción

La comunidad estudiosa de la Ingeniería de Software (IS) ha buscado, desde sus inicios, formas, herramientas, métodos y métricas que le permitan desarrollar software más rápido, más complejo, con mayor calidad y con menos errores. En ese sentido, hemos sido testigos de una evolución en lo que concierne a métodos cada vez más robustos y estables, herramientas de apoyo complejas y versátiles, métricas cada vez más precisas, etcétera.

Así, hacer aplicaciones de calidad que sean eficientes, robustas, apegadas a estándares, portables, y que apoyen a los usuarios de manera satisfactoria en sus tareas, es el deseo de cualquier equipo de desarrollo de software hoy día. Desafortunadamente, el desarrollo de software aún no alcanza, de manera sistemática, dichos objetivos; en particular en lo que se refiere a ayudar a los usuarios en sus tareas de manera efectiva. En efecto, es común encontrar hoy día herramientas sofisticadas, cuyas capacidades, cuidadosamente implementadas, son sub-explotadas por los usuarios finales (en ocasiones francamente rechazadas) dada su organización caprichosa, y su consecuente complejidad de uso.

La literatura coincide en que estos problemas se deben, entre otros factores, a que los criterios que siguen los desarrolladores de software para determinar las características de sus productos, se basan fundamentalmente en agregar a los sistemas tantas funciones como su lógica y sentido común les dicte. Así, el problema que se presenta es una excesiva buena voluntad de parte del equipo de desarrollo, que intenta dar a los usuarios todas las herramientas que imagina les pueden ser útiles; y la poca atención que se pone en averiguar lo que el usuario realmente requiere antes de desarrollar funciones que no serán usadas o, aún peor, funciones que no serán implementadas porque no se creen necesarias o "lógicas".

La comunidad de Ingeniería de Software reconoce que estos son problemas relacionados con la fase conocida como "Análisis y captura de requerimientos", avocada a establecer de manera completa y sin ambigüedades las necesidades del usuario. Sin embargo, puede inferirse de los problemas mencionados que dicho análisis es un proceso complejo y no suficientemente estudiado. Así, a pesar de que dicha fase es considerada de manera explícita en los procesos de desarrollo más maduros y exitosos, su puesta en práctica ha arrojado resultados poco satisfactorios.

El problema de aplicaciones que no responden a las necesidades del usuario también es estudiado por la comunidad dedicada a la Interacción Humano-Computadora (IHC). Dicha comunidad considera esas cuestiones como el resultado de un extremadamente pobre análisis sobre el usuario, su tarea y sus objetivos; y de la casi nula consideración de estos aspectos justamente durante la fase de análisis de requerimientos. Los trabajos de esta comunidad no se detienen ahí, llegando incluso a proponer sus propios procesos de desarrollo, modelos y herramientas que permiten desarrollar software que integra de una mejor manera los objetivos y costumbres del usuario. Desafortunadamente, los trabajos desarrollados en el seno de la IHC también enfrentan serias deficiencias en los productos que obtienen pues, si bien es cierto que interpretan de una manera más fiel los objetivos del

usuario, también es cierto que en términos de su calidad global (apego estricto a estándares, métricas, confiabilidad, robustez, etc.) los productos no tiene el mismo éxito. Así, pareciera que su problema es que carecen de un proceso completo, soportado con herramientas informáticas, y suficientemente probado y afinado, como para atraer a la industria a usarlo. Con esta limitante, los aportes que se han producido en dicha comunidad han tenido como destino común el uso puramente académico y la indiferencia de la industria.

Así, si resumimos los problemas de las comunidades mencionadas, notaremos que hay una coincidencia notable: es posible paliar los problemas de una comunidad mediante los aportes de la otra y viceversa. La pregunta es, evidentemente, porque esto no se ha dado. Precisamente, en ese sentido, uno de los primeros intereses de este trabajo será encontrar la respuesta a esta pregunta y proponer mecanismos que permitan acercar lo realizado por ambas partes, con el objetivo de obtener procesos más completos y de mayor calidad.

En los primeros capítulos del trabajo presentaremos los antecedentes necesarios para poder abortar los temas más específicos del trabajo y, por supuesto, para abordar la propuesta que se presenta.

En el segundo capítulo, hablaremos sobre lo que es la Ingeniería de Software y los procesos de desarrollo más populares. Explicaremos también lo que implica la actividad de captura de requerimientos dentro de los procesos mencionados para después abordar la manera en que se ve el desarrollo de software desde la perspectiva de la comunidad académica de IHC.

Como parte final del segundo capítulo plantearemos la manera en que pretendemos resolver el problema de la captura de requerimientos en el desarrollo de software, mediante la integración de las aportaciones de IS y de IHC.

En el tercer capítulo de este trabajo presentaremos la técnica del análisis de la tarea del usuario, el modelo MAD* mediante el cual modelaremos la tarea del usuario en nuestro trabajo y algunas posibilidades que existen para integrar esta técnica al proceso unificado de desarrollo de software. Este capítulo presenta las técnicas y procesos que se pretende ayudar a integrar.

En un cuarto capítulo presentamos los esfuerzos que se tienen registrados en el campo de las herramientas que asisten a los procesos de desarrollo de software y al modelado de la tarea del usuario.

En el quinto capítulo presentamos la propuesta de herramienta que hacemos en este trabajo describiendo desde la base teórica que motivó su realización hasta la manera en que fue implementada pasando por la forma en que fue diseñada y el porque fue diseñada de esta manera.

Finalmente, en el sexto capítulo mostramos un caso de estudio que permite al lector entender con mayor claridad la utilidad de la herramienta de una manera práctica. En el capítulo séptimo se detallan las conclusiones a que se llegó después de la realización del

trabajo incluyendo los trabajos que se propone se realicen para continuar con el esfuerzo de esta tesis.

2 La Ingeniería de Software y La Interacción Humano Computadora.

En este capítulo presentaremos dos áreas del conocimiento como lo son la ingeniería de software y la interacción humano computadora. Estas áreas de conocimiento, que son estudiadas a su vez por dos comunidades académicas distintas, se han mantenido distantes la una de la otra; en este capítulo mostraremos las debilidades de una y de otra así como sus puntos fuertes. Finalmente propondremos la integración de técnicas de IHC a procesos de IS como medio para mejorar sustancialmente la captura de requerimientos en el desarrollo de sistemas de software.

2.1 La Ingeniería de Software y sus procesos de desarrollo

La Ingeniería de software es un área del conocimiento que ha venido desarrollándose durante más de 30 años y que aún no madura del todo. La IS tiene como objetivo primordial la obtención, mediante el seguimiento de procesos de desarrollo adecuados, de un software de calidad. Este objetivo ha llevado a la comunidad de IS a desarrollar muy diversos modelos para el desarrollo de software desde la década de los 60's hasta nuestros días.

En los años sesentas y setentas algunos de los modelos más aplicados eran los de "En cascada" y "en espiral". En la actualidad, los procesos de desarrollo de software se han desarrollado y algunos de los más usados actualmente son *El Proceso Unificado* (PU) [3], *El Proceso Personal de Desarrollo de Software y El Proceso de Desarrollo de Software En Equipo* (PSP y TSP de sus siglas en ingles: *Personal Software Proces[7]s* y *Team Software Process[8]*) y el *Proceso de Programación Extrema* (XP de sus siglas en ingles: *eXtream Programming [9]*). En esta sección describiremos, a grandes rasgos, en que consisten todos los modelos mencionados.

2.1.1 Modelo En cascada

El modelo "En cascada" presenta una estructura rígida. Este modelo se popularizó en los años setenta y es el más frecuentemente citado en la literatura de la época y de varios años más.

Como lo muestra la Figura 1, este modelo está estructurado como una cascada de fases, donde la salida de una fase constituye la entrada para la siguiente. Cada fase está formada por un conjunto de actividades que pueden ser ejecutadas por diferentes personas al mismo tiempo.

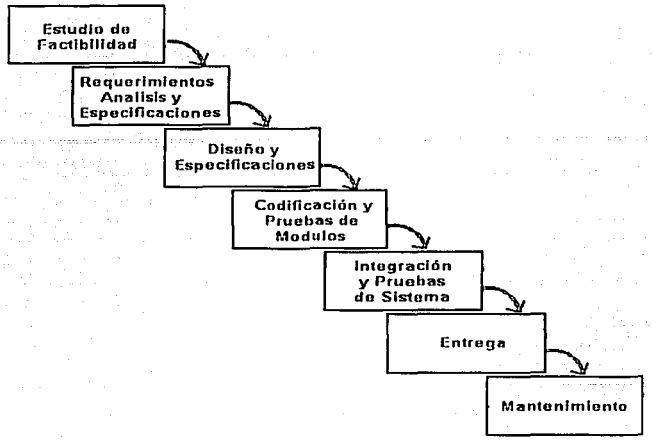


Figura 1 "Modelo en cascada"

El modelo "En cascada" [2, pp. 41-42]es elegido generalmente para aquellos sistemas sencillos y fáciles de entender debido a que los sistemas más complejos pueden requerir de ser divididos en procesos más pequeños con el fin de controlar de mejor manera el proceso y asegurarse más rigurosamente del desarrollo paso a paso. Otro aspecto que favorece la elección de este modelo son los casos en que el usuario final es experto, ya que de no serlo se haría necesario incluir una fase de entrenamiento para el mismo dentro del modelo. Evidentemente, con la creciente complejidad de los sistemas, este modelo es obsoleto para casi cualquier sistema actual.

2.1.2 Modelo En espiral

En segundo lugar se tiene el modelo "En espiral". La meta de este modelo es proveer un marco de trabajo para el proceso de desarrollo guiado por los niveles de riesgo.

La principal característica del modelo En espiral se encuentra en que es cíclico y no lineal, como en el caso del modelo En cascada.

Este modelo es interpretado de distintas formas de acuerdo con la bibliografía que se consulte. Sommerville[1] considera que cada ciclo de la espiral está constituido por cuatro estados y que cada estado está representado por un cuadrante en el diagrama cartesiano. El radio de la espiral representa el costo acumulado hasta el momento en el proceso, el ángulo representa el progreso en el proceso.

En este sentido, el modelo En espiral nos permite reiterar la expansión del crecimiento del sistema contra la corrección del mismo. Después de un ciclo, los requerimientos no

especificados anteriormente se convierten en requerimientos para la siguiente iteración. A consecuencia de esto, el crecimiento del sistema se acerca cada vez más al mejoramiento del mismo. El modelo de espiral, de acuerdo con Sommerville, es ilustrado en la Figura 2.

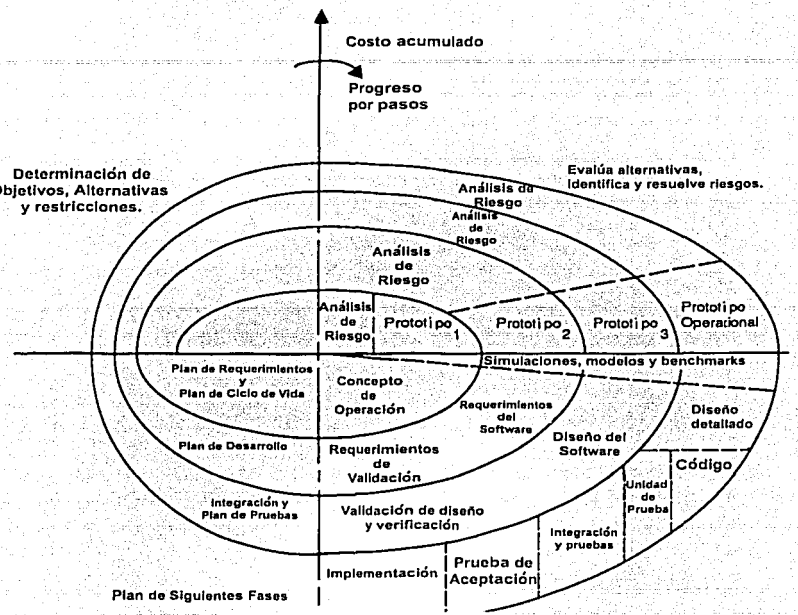


Figura 2 "Modelo de espiral según Sommerville"

Por otro lado, Pressman [2, pp. 28-29], basado en el modelo original de Boehm, considera que el ciclo se divide en seis regiones (Figura 3).

Éstas son:

- ✓ Comunicación con el cliente: las tareas requeridas para establecer comunicación entre el desarrollador y el cliente.
- ✓ Planificación: las tareas requeridas para definir recursos, el tiempo y otras informaciones relacionadas con el proyecto.
- ✓ Análisis de riesgos: las tareas requeridas para evaluar riesgos técnicos y de gestión.
- ✓ Ingeniería: las tareas requeridas para construir una o más representaciones de la aplicación.
- ✓ Construcción y adaptación: las tareas requeridas para construir, probar, instalar y proporcionar soporte al usuario.

- ✓ Evaluación del cliente: las tareas requeridas para obtener la reacción del cliente según la evaluación de las representaciones del software creadas durante la etapa de ingeniería e implementada durante la etapa de instalación.

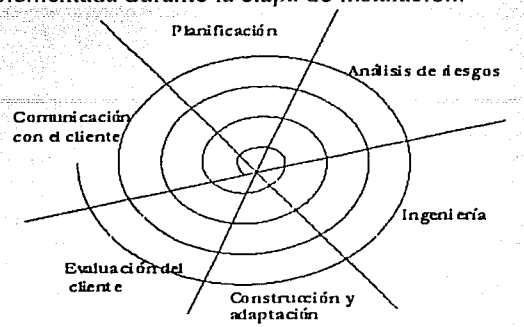


Figura 3 "Modelo en espiral de acuerdo con Pressman"

En la actualidad, el modelo *en espiral* ha dejado de ser utilizado en su forma original debido, al igual que en el caso del *en cascada*, a que la complejidad de los sistemas actuales requiere de procesos que especifiquen de manera mucho más detallada cada una de las fases y actividades del mismo. Aún así, el modelo en espiral fue considerado, desde un principio, como un metamodelo. En este sentido el modelo ha sido muy exitoso, ya que la mayor parte de los procesos de nuestros días utilizan conceptos como análisis de riesgos y, sobretodo, la mayoría de los procesos actuales integran los conceptos de "iterativo" e "incremental", como lo propone el modelo.

2.1.3 El Proceso Unificado de Desarrollo de Software

Durante un tiempo los dos modelos mencionados fueron los principales paradigmas a seguir para el desarrollo de software de calidad; es decir, los modelos más socorridos de la Ingeniería de software. Sin embargo, estos modelos eran solo eso, modelos y no procesos completos que indicaran con mayor detalle qué se tenía que hacer, cuándo se tenía que hacer, y quiénes lo tenían que hacer. La Ingeniería de software se quedó estancada en estos modelos y algunos otros que no lograron establecerse hasta fines de los años noventa, en que tres investigadores diseñaron un paquete completo que logró que la Ingeniería de software tomara un nuevo impulso. Los investigadores Ivar Jacobson, James Rumbaugh y Grady Booch, tres de las personas más reconocidas en el área, desarrollaron un proceso unificado de desarrollo de software (PU) [3], un lenguaje unificado de modelado (UML, de sus siglas en ingles: Unified Modeling Language) [4] y una herramienta que permite modelar, utilizando el mencionado lenguaje, la mayoría de los aspectos de los sistemas.

El PU es un proceso iterativo e incremental, esto significa que se crea un producto que, mediante iteraciones, se va acercando cada vez más al producto final. Estas iteraciones van

desde la captura de requerimientos hasta las pruebas, i.e, se crea el producto final con base en microproyectos.

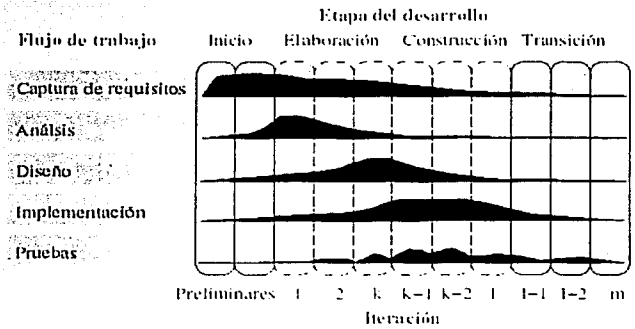


Figura 4 "Diagrama de flujo del Proceso Unificado"

El PU consiste en cinco flujos de trabajo y cuatro fases como lo muestra la Figura 4. Cualquier número de iteraciones pueden ocurrir en cada fase, sin embargo, como lo muestra la misma figura, no todos los flujos de trabajo ocurren con la misma intensidad a lo largo de todo el proceso: mientras que en el inicio la captura de requerimientos es muy intensa, cerca del final del proyecto ya no ocurre.

Este proceso es dirigido por casos de uso, éstos permiten realizar la captura de requerimientos en el lenguaje del cliente y luego traducirlos en una representación formal de ellos utilizando UML. Mediante los casos de uso se pulveriza la funcionalidad del sistema tanto como el desarrollador lo quiera.

Como veremos en el resto del presente trabajo, el Proceso Unificado es el proceso en el que nos enfocaremos para realizar nuestra propuesta. Las razones de esta decisión se verán con claridad cuando expliquemos la manera en que se capturan los requerimientos en cada uno de los procesos (Ver sección 2.2).

2.1.4 Personal y Team Software Process

El PSP [7]fue desarrollado en Carnegie Mellon por el Instituto de Ingeniería de Software (SEI por sus siglas en ingles: Software Engeneering Institute) y define un proceso que pretende ayudar a cumplir con un subconjunto del estándar CMM (Capability Maturity Model)[1].

La premisa de la que se parte al proponer PSP como proceso de desarrollo es que si se logra que el desarrollador de software aumente su efectividad, con base en el establecimiento de

[1] CMM es un estándar de calidad para el desarrollo de software establecido por el SEI.

disciplina mediante procesos de trabajo más estructurados y mejor medidos, la calidad de lo que produzca aumentará también.

Un segundo aspecto importante es la idea de que el ingeniero de software no acostumbra planear, monitorear, y mucho menos medir los resultados de sus actividades. De esta manera, la calidad del software que produce tampoco es medida generalmente.

La aplicación de PSP ha ayudado a la mejora sustancial de los estimados en los proyectos así como en la confiabilidad de las planeaciones. Esto debido a que el proceso está basado en la medición y evaluación de todas y cada una de las actividades del proceso de desarrollo.

El TSP [8], por otro lado, propone una variante de PSP pero enfocada a los equipos de trabajo. Si bien el PSP se enfocaba a mejorar la calidad en los procesos personales de desarrollo, el TSP se propone mejorar el desempeño de los equipos de trabajo. TSP basa la mayor parte de sus métricas en las propuestas en PSP por lo que es importante, para un equipo que pretenda implementar TSP, el conocer y haber puesto en práctica PSP.

En TSP, como en todos los procesos de desarrollo actuales, se tienen iteraciones indefinidas de un ciclo de manera que el software se va incrementando y se deja de iterar en el momento en que el sistema queda concluido.

Un ciclo de desarrollo TSP consta de las siguientes fases:

- Requerimientos
- Diseño de Alto nivel
- Implementación
- Integración y Pruebas

Cada una de estas fases puede ser desglosada en diversas actividades, obtiene datos de diversas fuentes y obtiene diversos resultados. Esto se muestra en la Figura 5.

Los resultados de cada una de estas fases son medidos mediante los mecanismos propuestos desde PSP, y es con base en estas mediciones que se hacen las siguientes iteraciones al ciclo.
Parte esencial de TSP es el hecho de que estas fases no son necesariamente secuenciales como en el caso de un proceso en cascada o incluso un proceso en espiral. En TSP se pretende que los proyectos tengan una o más de estas fases corriendo en cada momento, es decir, mientras una parte del equipo se encuentra realizando el diseño de alguna parte del producto, otra parte puede encontrarse en el desarrollo de otra; de hecho, TSP es un proceso de desarrollo en equipo que ha sido utilizado en equipos multidisciplinarios escapando al ámbito del software.

Si bien podemos acordar que el PSP y el TSP son procesos robustos y que logran el objetivo de aumentar la calidad en los procesos de desarrollo de software, no consideramos

sean el tipo de proceso encaminado a generar un producto de calidad sino, mas bien, están encaminados a generar individuos y equipos con procesos de mayor calidad.

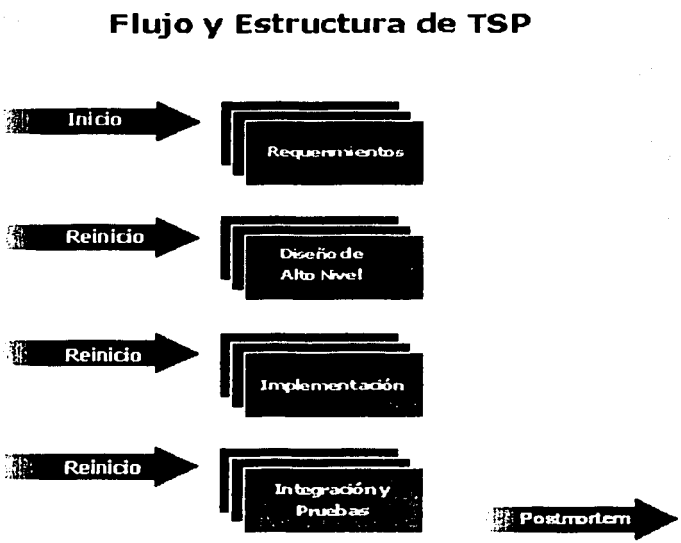


Figura 5 "Flujo y Estructura de TSP"

El SEI dice en su página Web [7]: "El PSP puede ser aplicado a muchas partes del proceso de desarrollo de software, incluyendo el desarrollo de pequeños programas, la captura de requerimientos, la documentación del sistema, las pruebas del sistema y el mantenimiento". Esta oración aclara la sutil diferencia entre un proceso como el PU y otros como PSP o TSP. Esta diferencia se marca en el hecho de que PSP se puede "aplicar a muchas partes del proceso de desarrollo de software", es decir, no es un proceso para mejorar la calidad del software en sí, sino un proceso que deben realizar los individuos, o los grupos de individuos (en el caso de TSP), con el fin de mejorar la calidad de su desempeño particular; y como consecuencia, por supuesto, obtener un mejor producto.

2.1.5 Extreme Programming

XP (Extreme Programming) [9] es un proceso de desarrollo que recientemente cobra gran popularidad, a pesar de tener ya seis años de vida. Este proceso se basa en un proceso de ciclos de desarrollo muy cortos, que permiten el ajuste constante del rumbo del proyecto. XP fue creado en respuesta a dominios del problema cuyos requerimientos cambian. Es probable que se tenga un sistema cuya funcionalidad se espera cambie cada pocos meses. En muchos ambientes de software, el cambio dinámico de los requerimientos es la única

constante. La meta fundamental de este proceso de desarrollo es obtener lo que se necesita cuando se necesita.

XP se basa en tres principios fundamentales, el cambio en los requerimientos, el concepto de equipo y la evaluación de riesgos. Estos son, a su vez, los puntos que hacen diferente al proceso del resto de los procesos de desarrollo de software. En XP, los requerimientos pueden ir cambiando constantemente, aún en etapas avanzadas del proyecto. Esto, por supuesto, resulta muy atractivo para proyectos donde los requerimientos son dinámicos.

La evaluación de riesgos, por otro lado, es también un punto fuerte de XP: el proceso está preparado para atenuar grandes riesgos como un cambio en la tecnología o desarrollos nunca antes realizados.

Finalmente, el factor equipo es fundamental en XP. Para ello, XP requiere que el mismo cliente sea parte del equipo de desarrollo. XP está diseñado para grupos de trabajo entre 2 y 12 personas, a diferencia de otros procesos de desarrollo creados para grandes equipos. Para el correcto funcionamiento de un desarrollo en XP, es necesario mantener un diseño simple siguiendo estándares estrictos de codificación (a fin de que cualquier programador pueda modificar el código de cualquier otro). Se requiere también probar el sistema constantemente, es decir, día a día y comenzando desde el primer día, de tal manera que la retroalimentación sea constante también; para ello, es necesario que el sistema se encuentre funcionando lo más pronto posible aún y si su funcionalidad es mínima.

El proceso de desarrollo XP tiene una cantidad de actividades y flujos entre ellas que no detallaremos en este trabajo, aún así se muestra el flujo básico del proceso. (Ver Figura 6)

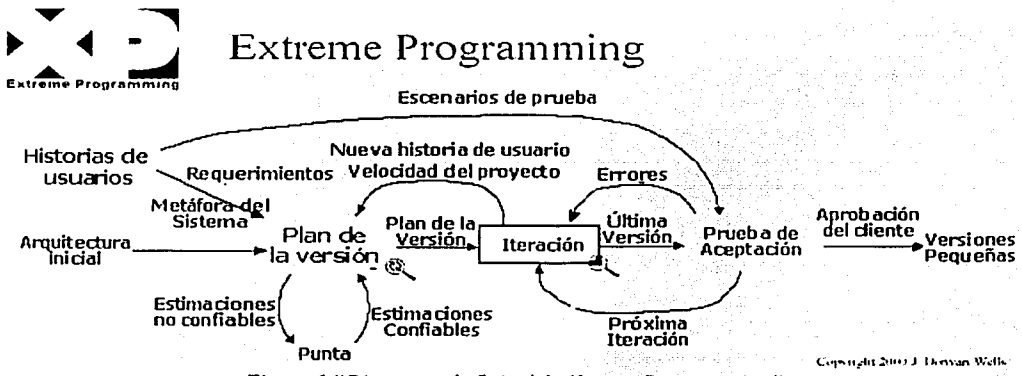


Figura 6 "Diagrama de flujo del eXtream Programming"

2.2 La Captura de Requerimientos

Hemos descrito ya lo referente a la historia de la Ingeniería de Software así como de los procesos de desarrollo que se han generado a través de la misma. A continuación se realiza

un análisis de la situación de la actividad de Captura de Requerimientos dentro de los tres procesos de desarrollo más actuales de los aquí citados.

2.2.1 La captura de requerimientos en TSP

En secciones anteriores (Ver 2.1.4) se examinaron los procesos de PSP y TSP, ambos creados en el SEI. A continuación veremos en que consiste la captura de requerimientos en TSPi (introducción a TSP) [33]y en una versión académica que combina TSPi con el proceso unificado. Veremos estos procesos en particular pues están bien documentados y son suficientemente completos, ya que incluyen el manejo de equipo además de las actividades personales.

En TSPi la captura de requerimientos se basa en el siguiente proceso: el cliente especifica los requerimientos en una serie de enunciados de necesidades (*needs statements)* generando así la primera versión de un documento llamado SRS (Especificación de Requerimientos del Sistema, de las siglas en ingles *System Requirements Specification*). Los enunciados de este documento son, a decir de Humphrey, vagos e imprecisos y por ello deben ser analizados por los desarrolladores mismos que deberán hacer preguntas a los clientes a fin de aclarar todos los aspectos que no estén claros en ellas.

Una vez resueltas sus dudas, los desarrolladores deberán reestructurar los enunciados con sus propias palabras y llevar el nuevo documento al cliente para validarlo. En general, en TSPi el proceso de captura de requerimientos "consiste principalmente en hacer y contestar preguntas".

El proceso de TSPi para capturar los requerimientos tiene varios problemas que coinciden con los que tienen la mayoría de los procesos de desarrollo de software actuales, el principal de ellos es que se cree ciegamente en lo que el cliente o el usuario pide y esto normalmente no es ni preciso ni suficiente ya que generalmente las verdaderas necesidades del cliente no las conoce por completo. Por otro lado, no existe ningún mecanismo que ponga los requerimientos del cliente en un esquema temporal o en algún tipo de simulación que permita encontrar inconsistencias o contradicciones en sus peticiones.

Por otro lado, dentro de la versión académica mencionada al comienzo de la presente sección, la captura de requerimientos comienza aún antes de la fase que lleva ese nombre. Es en una de las primeras fases del desarrollo en que el cliente realiza la llamada "Definición del problema". Este documento, donde el cliente explica el problema en texto simple, es de donde se parte en la "Fase de requerimientos". Esta fase es desglosada en varias actividades como se puede observar en la Figura 7.

La fase consta de muchas actividades, entre ellas se encuentran la actividad "Revisar la Descripción de Necesidades del Sistema" y varias más para la generación de documentos que especifican formalmente los requerimientos del sistema. Algunos de éstos documentos son los diagramas de casos de uso del sistema y el documento SRS (Documento de TSP donde se especifican, en texto simple, los requerimientos del sistema).

Dentro de la actividad de "Revisar la Descripción de Necesidades del Sistema", se incluye la revisión del documento "Definición del problema". En esta actividad se tiene una leve interacción con el cliente pero, en ningún momento, con el usuario. Además de ello, aún si la reunión fuera con el usuario, se parte del principio de acudir a él "para aclarar puntos que podrían ser ambiguos", lo cual implica que todo aquello que el desarrollador considere claro, no será validado con el cliente.

Otro problema con la captura de requerimientos en este proceso es la plena confianza en que el cliente recordará, o sabrá, todos los requerimientos del sistema y los anotará en un documento; es decir, no se tiene ningún mecanismo para averiguar requerimientos que el cliente no exprese de manera explícita.

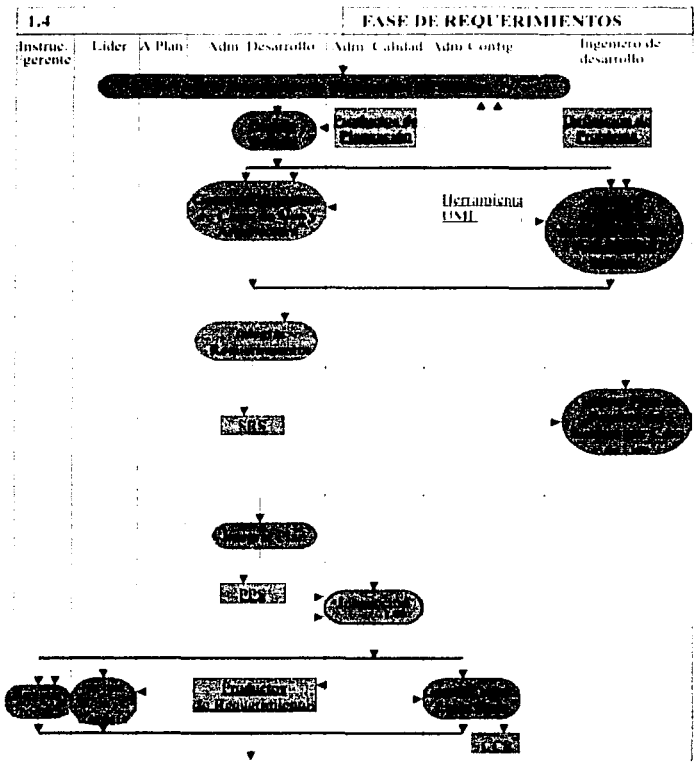


Figura 7 "Fase de Captura de Requerimientos en TSP"

2.2.2 La captura de requerimientos en eXtream Programming

Dentro de XP [9] existen los documentos llamados "Historias de Usuarios" (US del ingles "User Stories"). Estos documentos concentran toda la actividad de captura de requerimientos. Los US tienen un objetivo similar a los Casos de uso mencionados en el Proceso Unificado: tratan de mostrar las funciones que debe cumplir el sistema. Los US son creados por los clientes como un documento donde especifican lo que el sistema debe hacer por ellos, estos documentos son escritos en aproximadamente tres párrafos de texto y están en el lenguaje del cliente.

Los US son usados en varias partes del desarrollo del sistema. Por un lado, es a partir de ellos que se determinan los estimados de tiempo para el plan general del desarrollo. Es también mediante ellos que se crean las pruebas de aceptación; estas pruebas son escenarios diseñados por el cliente que permiten probar la correcta implementación de una US.

Como ya se comentó, XP basa su efectividad en una gran cantidad de ciclos, para cada ciclo, los US deberán tener nuevas pruebas de aceptación que superar, de no ser así, se considera que el US no ha progresado.

La principal diferencia de los US respecto a los documentos de requerimientos a los que nos tienen acostumbrados otros procesos de desarrollo se encuentra en el nivel de detalle. De hecho, los US deben mantenerse a un alto nivel de abstracción que solo sea lo suficientemente específico como para que la dificultad de implementación sea más o menos calculable. Para mayores detalles sobre la funcionalidad requerida, los desarrolladores deberán acudir nuevamente al cliente y recibir una descripción detallada de la función. Una manera de saber si se tiene suficientemente detallado un US es que su implementación no debería ser estimada en más de tres semanas ni en menos de una.

La captura de requerimientos resulta, en XP, una etapa de enorme importancia, aún así, podemos notar que los recursos para obtener los requerimientos del sistema son bastante limitados; de hecho, únicamente se cuenta con los "User Stories" para ello. Mediante los US se tienen severas limitantes en cuanto a la manera de obtener los requerimientos que el cliente no conoce. Es decir, la manera en que XP obtiene los requerimientos del sistema depende, exclusivamente, de lo que el cliente mismo le diga. De esta manera resulta imposible conocer aspectos como posibles incoherencias entre distintos US's o US's que en realidad no le son útiles al usuario. Ésto hasta la implementación de los US's, o peor aún, hasta que el proyecto ha finalizado y el sistema es puesto en operación.

El XP no es considerado como el adecuado para el presente trabajo por dos razones fundamentales: por un lado, el XP se basa en la captura de requerimientos por medio del cliente y su validación por el mismo medio; es decir, si el cliente se equivoca en cuanto a sus requerimientos, el equipo de desarrollo no tendrá forma de contemplar determinada función. De manera un tanto sorprendente, XP no considera esto como una falla.

Otro aspecto a destacar es la diferencia de ámbitos en que XP y nuestra propuesta se pueden aplicar. XP está pensado para sistemas cuyos requerimientos van cambiando

constantemente, mientras que las técnicas que aquí propondremos pretenden, precisamente, evitar que los requerimientos cambien. Es decir, las técnicas que se presentarán intentan apoyar la realización de una mejor captura de requerimientos, de tal manera que en etapas avanzadas del proyecto no se tengan cambios de los mismos. XP, por otra parte, no tiene porque invertir en una mejor captura de requerimientos dado que está preparado para que estos cambien constantemente.

2.2.3 La Captura de Requerimientos en el PU.

Dentro del Proceso Unificado [3], la captura de requerimientos es una actividad de gran importancia. De hecho, es uno de los cinco flujos de trabajo que tiene cada iteración del proceso y que ya se mencionaron anteriormente(Ver 2.1.3).

El flujo de captura de requerimientos comprende cuatro actividades:

- Enumerar requerimientos Candidatos
- Comprender contexto del sistema
- Capturar requerimientos funcionales
- Capturar requerimientos no funcionales

En la primera actividad se listan los requerimientos pedidos por el cliente. Esta primera lista solo es utilizada para medir inicialmente el proyecto y para hacer una primera división en ciclos de desarrollo. La lista, sin embargo, irá evolucionando y para ello a cada requisito propuesto se le guardará junto con información como el estado en que se encuentre (propuesto, aprobado, validado), su costo estimado de implementación (horas hombre/máquina), su prioridad (crítico, importante, secundario) y su nivel de riesgo asociado a la implementación (crítico, significativo, ordinario). A partir de esta información se decide cuales requerimientos serían considerados en la iteración y cuales se dejarán para una posterior y cuales simplemente no serán considerados en el proyecto.

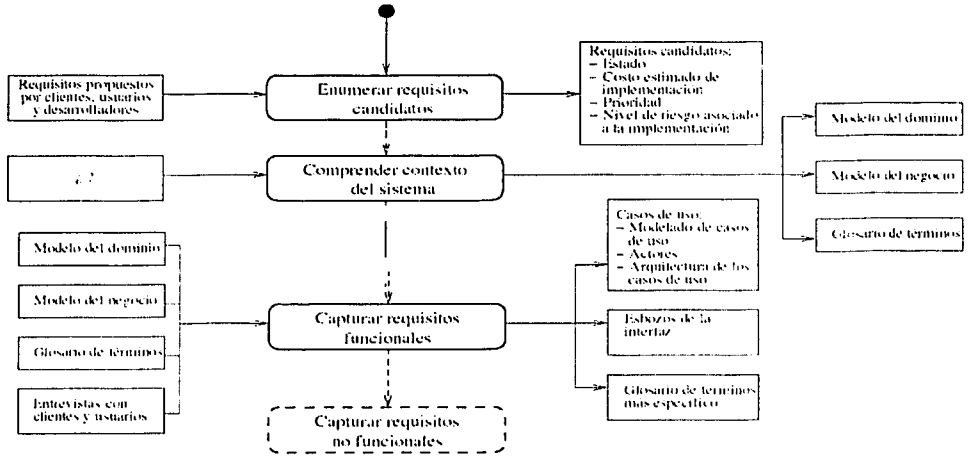


Figura 8 "Flujo de Trabajo Captura de Requerimientos en el PU"

En "Comprender el contexto del sistema" se tiene, obviamente, que comprender el mencionado contexto. Esto implica conocer las reglas del negocio, el ámbito de trabajo, técnicas y procedimientos de los usuarios para realizar su trabajo, su experiencia, etc. De hecho, esta actividad debería permitir la definición de todos aquellos requerimientos que no pidió el cliente pero que son necesarios para el sistema por alguna otra razón.

En la actividad de "Capturar requisitos funcionales", se definen las características de funcionamiento que tendrá el sistema de software, a estas características se les llama requerimientos funcionales. Estas funcionalidades del sistema son representadas mediante los casos de uso de la notación UML. En ellos se especifican los actores que interactuarán con el sistema, las funciones que accesarán, las relaciones entre éstas, etc.

También durante esta actividad se crean los esbozos de la interfaz y un glosario de términos más específico. Con los esbozos de la interfaz se puede discutir la funcionalidad del sistema antes de comenzar su desarrollo, e inclusive antes de firmar un contrato. El propósito del glosario de términos actualizados es asegurarse, lo más posible, de entender lo mismo que el cliente término por término.

Finalmente, en "Capturar requisitos no funcionales", se determinan los requerimientos adicionales del sistema, tales como rendimiento, compatibilidad, etc. Éstos se encuentran también a partir de las entrevistas con clientes y usuarios y de los modelos del contexto del sistema. Ya que los requerimientos no funcionales no son funcionalidades del sistema no son modelados mediante casos de uso, por ello, el proceso unificado solo propone mantener una lista de ellos hasta que ésta se utilice en el flujo de trabajo de diseño.

Como se puede observar en la descripción hecha, la captura de requerimientos es una actividad bien descrita en el PU, sin embargo, también es evidente en la descripción y sobre todo en la Figura 8, que existe una importante ausencia. La actividad de comprender el contexto del sistema no tiene una entrada de datos, es decir, el PU no describe la manera en que esto se puede lograr. Evidentemente se tienen las entrevistas a los usuarios, la lista de requerimientos candidatos, etc. De modo que efectivamente existen elementos para ayudarse a lograr esta comprensión; sin embargo, no existe un método claro para ello. El problema no es pequeño, podemos recordar que es con base en esta comprensión del contexto que debemos generar todos los requerimientos que no menciona el cliente. Si no se comprende bien el contexto del sistema, seguramente se perderán requerimientos que saldrán a la luz en etapas más adelantadas del proyecto y generarán gastos mucho mayores.

2.2.4 Problemas con la captura de requerimientos

Toda vez que hemos examinado la manera en que se realiza la Captura de requerimientos en los procesos de desarrollo de software actuales, veremos cuales son los problemas generales que esta actividad plantea.

Como se ha visto, el desarrollo de software tiene, actualmente, bases muy sólidas para el aseguramiento de su calidad con base en los procesos de desarrollo que se han presentado los últimos años. Aun así, los sistemas que se desarrollan actualmente cuestan mucho dinero en mantenimiento o en cambios que se realizan cuando se está en etapas finales del desarrollo.

Esto se debe a que la captura de requerimientos está resultando fallida. Si bien hemos visto como los procesos de desarrollo le están dando importancia a esta etapa, hemos descrito también las fallas que cada uno presenta. Así, podemos asegurar que las diferentes fallas obedecen a varios aspectos generales.

1. El primero de ellos es que la mayor parte de los procesos de desarrollo de software enfocan al cliente como la entidad de la se deben obtener los requerimientos y esto no debería ser así. En realidad, es el usuario de quien se deben obtener la mayor parte de los requerimientos del sistema. La importancia de analizar al usuario y a la tarea que realiza se explicará con más detalle mas adelante en este trabajo(Ver 2.3).

2. Es también notorio como, aunque los procesos indican que hay que analizar los requerimientos concienzudamente, y de hecho se menciona la necesidad de comprender y modelar el negocio, en ningún proceso se presentan técnicas o metodologías que ayuden para este análisis; de hecho, se deja completamente abierta la manera en que el desarrollador decida hacerlo.

3. Otra deficiencia que presentan los procesos de desarrollo en general es la plena confianza en lo que el cliente, o en el mejor de los casos el usuario, cree que necesita. En ningún momento los procesos de desarrollo contemplan algún tipo de actividad encaminada a averiguar, o validar, que aquello que el usuario dice respecto a sus necesidades sea cierto. Evidentemente esto no significa que los

clientes mientan, lo que se quiere decir es que la mayoría de las veces es necesario averiguar, de alguna manera, necesidades que el cliente no se da cuenta de que tiene y que, por lo tanto, no solicita, pero que hacen que el producto no resulte exitoso al ponerse en funcionamiento.

En términos generales la captura de requerimientos ha sido deficiente por falta de técnicas que investiguen de mejor manera los tres aspectos fundamentales que se requiere conocer que son: ¿Qué necesita el usuario?, es decir, que funcionalidad deberá tener el sistema. ¿Cuándo necesita cada cosa?, es decir, en que momentos requiere ciertas funcionalidades y en que momento requiere otras. Y finalmente, ¿Cómo requiere que se le presenten las mencionadas funcionalidades?, de tal manera que le resulten utilizables, además de útiles.

2.2.5 Buscando Soluciones

Una vez observado el estado actual de la captura de requerimientos en los procesos de desarrollo de software, y habiendo ya identificado los principales problemas que presenta la actividad, el siguiente paso es presentar una propuesta mediante la cual logremos aminorarlos.

La comunidad de Ingeniería de Software no es, de ninguna manera, la única comunidad que estudia el desarrollo de software; de hecho, existe una comunidad que se enfoca al estudio de la Interacción Humano-Computadora (IHC). Esta comunidad se basa en la idea de que para lograr una buena interacción entre el usuario y la computadora, es necesario conocer mejor al futuro usuario de la misma. Por ello, la comunidad tiene avances muy importantes en la comprensión de lo que el usuario hace, es decir, de su tarea. Gracias a esto último, y mediante algunas técnicas de IHC, la comprensión del contexto del sistema puede lograrse de una mejor manera.

Aun con estos avances, esto no ha redundado en mejoras en los sistemas comerciales debido a diversas razones. Por motivos que se analizaran en la siguiente sección, no se ha logrado que estos avances se integren en la industria.

Además de éste análisis, en la siguiente sección se mostrarán los conceptos que dan forma a la IHC explicando también los problemas que presenta esa comunidad académica. Finalmente, en la última sección de este Capitulo mostraremos la forma en que se pretende hacer que los avances de IHC se integren a los de la ingeniería de software y, con ello, a la industria.

2.3 El enfoque de la IHC

En esta sección se mostrarán los conceptos básicos de la IHC. Primeramente se explicará el concepto de "Diseño Centrado en el Usuario"; enseguida, se mostrará la importancia del Análisis del usuario y de su tarea dentro del desarrollo de software y se explicará como, mediante este estudio de la tarea, se logra la captura de requerimientos en IHC. Finalmente,

mostraremos algunos de los problemas que presenta IHC y que evitan la producción de software de calidad.

2.3.1 Diseño centrado en el usuario

Todo sistema, ya sea de software, mecánico, legal o de cualquier otra índole, está hecho para que alguien o algo que lo utilice. Un sistema legal, por ejemplo, no tendría razón de ser si no es creado para que una población lo utilice. No tendría sentido crear una extraordinaria computadora o un aparato mecánico con características de desempeño nunca antes vistas, si éste no será utilizado para ningún fin por nadie o nada.

El usuario, sea quien sea, tendrá que adaptarse al sistema ya que, hasta el momento, no ha sido creado ni se tiene una idea de cómo crear el sistema perfecto o completamente adaptable; aún así, es necesario hacer todos los esfuerzos posibles con el fin de que el sistema se adapte en mayor grado al usuario, de lo que éste se tenga que adaptar al sistema.

Para lograr esto, es necesario conocer las características del usuario. Si se hace esto, las probabilidades de que el sistema sea aceptado por el usuario crecerán enormemente, debido a que tendrá una menor carga de trabajo para adaptarse al sistema. Al diseño de software basado en este aspecto se le denomina como desarrollo ***Centrado en el Usuario***.

A través de los años, la experiencia nos ha mostrado que el desarrollo de sistemas que respondan a las necesidades reales del usuario provoca una enorme inversión de tiempo e implica una gran complejidad, por lo que lograr comprender estas necesidades es de fundamental importancia en el desarrollo de un sistema.

Actualmente, y particularmente en el desarrollo de software, el tiempo de desarrollo del sistema para tener el producto listo para entregarlo al cliente, es solo una parte (40% aproximadamente) del tiempo total de desarrollo, ya que una vez que el producto se entrega, el cliente encuentra cosas que no deseaba, y se da cuenta que el producto carece de otras cosas que sí necesita, por lo que se requiere la realización de cambios que implican una gran inversión de tiempo y dinero.

Es de destacarse que las razones de estos cambios no son "errores" de funcionamiento del producto ya que éstos fueron, o debieron ser, corregidos mediante los múltiples mecanismos que proveen los procesos de desarrollo de software para el aseguramiento de la "calidad" del producto como lo son: las evaluaciones entre pares, las pruebas unitarias, las pruebas de integración y la constante evaluación de riesgos, etc. En realidad, se trata de aspectos que tienen que ver con aquellas funciones que no se sabía que el cliente necesitaba, o con problemas que tampoco se estaba conciente de que el cliente tenía o tendría con el nuevo software. La corrección de todos estos problemas podría evitarse, o por lo menos minimizarse, mediante un mejor estudio de los requerimientos del usuario pero con una visión distinta a la que hasta ahora se utiliza.

Para este estudio es necesario primeramente conocer al usuario. Existen dos vertientes del problema: por un lado, es necesario tener conocimientos de las características de los

usuarios en general. Esto es algo necesario para cualquier diseñador de sistemas, ya que todo lo que sepa en este sentido le será útil en cualquier desarrollo que realice. En este tipo de estudios se pretende conocer las características de la forma en que el usuario utiliza un sistema en general y las cosas que habitualmente requiere de él.

Por otro lado, existe otro tipo de estudio que es necesario hacer. Se requiere conocer las características particulares de los usuarios que tendrá el sistema que se va a realizar, es decir el usuario "tipo". Se conoce como usuario tipo a todo aquel que reúne las características de quien va a utilizar el sistema normalmente. Por ejemplo, el usuario tipo de un software educativo de primaria es todo aquel que cumpla con las características: tener entre 6 y 12 años, tener acceso a computadora en casa, ir a la escuela, etc.

Para conocer a este usuario es necesario hacer un estudio para el sistema que se va a desarrollar ya que se desean las características particulares del usuario de *ese* sistema. Algunas de las características que se deben obtener del usuario tipo son los gustos, los conocimientos, las costumbres y por supuesto, su tarea, aspecto que estudiaremos en la siguiente sección.

Por otro lado, es de considerarse el nivel de conocimientos del usuario. Sabiendo que elementos conoce, éstos pueden ser utilizados en el sistema con la conciencia de que serán comprendidas. Un ejemplo de la importancia de este aspecto es la diferencia que hay entre la interfaz que se realizaría para un sistema que usará un adulto que tendrá su primer contacto con una computadora (ej. un cajero automático) y aquella que se haría para el uso de un programador experto (ej. un ambiente de programación visual).

2.3.2 Captura de requerimientos en el Diseño Centrado en el Usuario: Análisis de la tarea

Si bien el estudio del usuario es muy importante, de igual importancia es el estudio de lo que el usuario hace, de la tarea que realiza, ya que gracias a este estudio se podrá diseñar de mejor manera la herramienta que ayudará a su realización. A este estudio se le llama *Análisis de la Tarea*.

La tarea es la serie de pasos que el usuario realiza para alcanzar un objetivo particular. Nuestro objetivo, como diseñadores de software, es que dichas tareas se reflejen de una manera pertinente en nuestros productos. Cada uno de estos pasos es llamado sub-tarea y tiene un objetivo parcial encaminado siempre al cumplimiento del objetivo de la tarea global; es decir, si el objetivo general es obtener un título de licenciatura podemos considerar que la tarea es cursar la carrera pero esta tarea se compone de sub-tareas como lo son el cursar cada una de las materias, el realizar el servicio social, realizar la tesis y presentar el examen profesional. Cada una de estas sub-tareas tiene como objetivo parcial el lograr una calificación aprobatoria pero ésta va encaminada a lograr el objetivo global de obtener el título.

Uno de los aspectos en los que se hace evidente la importancia de este análisis es en la congruencia que debe existir entre la tarea que hace actualmente el usuario, y la tarea que

realizará con la ayuda del sistema: Es decir, dado que el usuario realiza su tarea de determinada forma, es importante minimizar el cambio al que tenga que adaptarse, esto con el fin de evitar que decida continuar haciendo la tarea de la manera original debido a que no comprende la forma de realizarla con el nuevo sistema.

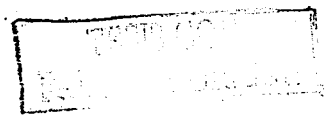
Otro aspecto de vital importancia, y que se deduce a partir del análisis de la tarea, son los objetivos que el usuario tiene en mente cuando realiza su tarea. La importancia de estos objetivos radica en que el nuevo sistema debe permitir alcanzar los objetivos que tenía la original. Además, el análisis de estos objetivos nos permitirá diferenciar entre las diferentes sub-tareas que se realizaban; por ejemplo, si la sub-tarea tiene como objetivo preparar la papelería que será necesaria para enviar una serie de correos, el objetivo de la tarea que la precede seguramente es algo como "enviar información a cuenta habientes". Mediante el sistema que se desarrolle, no se utilizará papelería, se enviarán correos electrónicos mediante formatos que el sistema ya tiene; por ello, la tarea cuyo objetivo era preparar papelería pierde sentido y deja de existir pues ya no es necesaria en vías de cumplir con "enviar información a cuenta habientes".

En general podemos decir que si la no ejecución de una sub-tarea no afecta el cumplimiento del objetivo de la predecesora, ésta deja de ser necesaria y por ello, ya no es necesario continuar considerándola en la nueva tarea. En caso de que una sub-tarea tenga un papel importante en el cumplimento de un objetivo y por algún motivo no pueda seguir siendo considerada en la tarea, es necesario encontrar una vía para compensar esta falta ya que los objetivos deben necesariamente cumplirse.

Cuando se diseña un sistema mediante el modelo llamado "orientado a objetos" es de fundamental importancia la identificación de los objetos del sistema. El paradigma orientado a objetos, de ingreso relativamente reciente al mundo del desarrollo de software, basa toda su organización en la suposición de que el ser humano concibe todo lo que le rodea en relación con objetos, tipos de objetos(clases), atributos y acciones(métodos) de los mismos, etc. Es precisamente basándose en esta idea que se establece que la tarea del usuario puede ser modelada en términos de los objetos que se encuentran involucrados, sus características conforme transcurre el proceso y las acciones que realizan. Si se logra modelar la tarea del usuario en estos términos, será de gran utilidad para el modelado del sistema si se utiliza el modelo orientado a objetos ya que la traducción se hará de forma mucho más directa si se habla de las mismas cosas en ambos modelos.

2.3.3 Problemas con el diseño centrado en el usuario

Actualmente, el análisis de la tarea es muy poco considerado en el desarrollo de software. Esto significa que los desarrollos que se hacen actualmente carecen de la información que este análisis provee lo que genera sistemas que no consideren de suficientemente las necesidades del usuario.



Lo poco que se considera actualmente es un análisis del usuario promedio; es decir, no se estudia al usuario particular que utilizará al sistema, sino que se toman en cuenta reglas establecidas de antemano que son hechas con base a las características del usuario en general (ejemplo de ello son las guías de estilo que se tienen para interfaces de usuario Windows®).

El área de la publicidad, por ejemplo, toma con mayor seriedad el estudio del público particular que utilizará el producto, verá o escuchará el promocional. Esto se realiza sobretodo en los grandes desarrollos o campañas ya que se sabe del gran impacto que éstos estudios pueden tener en el éxito o fracaso del producto. En el campo del desarrollo de sistemas de cómputo de escritorio la situación es aún más dramática: en esta área no se realiza un estudio de las características del usuario objetivo, mucho menos de su tarea, aún y cuando se trate de proyectos de gran envergadura como el desarrollo de un sistema operativo.

Esto último podría ser considerado una exageración o incluso una mentira si se considera el gran nivel alcanzado en las interfaces de usuario de los sistemas operativos Windows® o los paquetes de oficina como Office®, sin embargo no es así. La estrategia que se utiliza actualmente en las grandes empresas para lograr interfaces de usuario de calidad en sus productos no es otra que la antiquísima técnica de prueba y error. El proceso de producción de casi cualquier producto de éstos es una realización del sistema, un lanzamiento al mundo para ser probado, una corrección, un lanzamiento a la venta y finalmente una etapa de un año o dos de actualizaciones o parches al producto con base en las quejas de los usuarios.

La anterior forma de lograr un producto que responda a las necesidades del usuario tiene dos desventajas fundamentales que se eliminan o por lo menos se atenúan mediante los desarrollos que realizan un estudio del usuario como parte de la captura de requerimientos. La primera diferencia que se plantea es evidente, en caso de preguntarle las necesidades al usuario antes de desarrollar el software, éste saldrá al mercado con una mayor aproximación al producto deseado y se requerirán menos versiones del mismo. La otra ventaja de los desarrollos que llamaremos "centrados en el usuario" es que si se realiza un correcto análisis de la tarea se logrará obtener información sobre necesidades que el usuario tiene y en las cuales no ha reparado. Si se espera la retroalimentación del usuario de la manera tradicional se corre el riesgo de que éste jamás se de cuenta de estas necesidades.

Esta estrategia indudablemente se aleja de lo que se propone mediante el análisis de la tarea. De hecho, lo que aprovecha la estrategia de prueba y error es la característica de que los actuales desarrollos deben ser iterativos e incrementales, es decir, la gran mayoría de los procesos de desarrollo de software que se utilizan actualmente se basan en la producción de sistemas pequeños, y la corrección y ampliación de sus capacidades en ciclos sucesivos de un procedimiento preestablecido que generalmente implica el análisis, diseño e implementación de las nuevas características. Éste proceso continúa hasta que el sistema es considerado como completo.

La pregunta que surge inmediatamente es ¿porqué se tienen que realizar tantas correcciones, porqué no se hace bien desde un principio? La respuesta a esto estriba en que desde un inicio no se logró comprender perfectamente lo que el usuario deseaba y poco a

poco se va descubriendo que es lo que realmente necesita. Las razones de que este mal entendimiento se presente son muy diversas, y van desde la posibilidad de que el desarrollador haya entendido mal, hasta que el usuario haya explicado mal; pasando por la más común de ellas: el usuario estaba equivocado respecto a lo que quería. Lo anterior podría sonar ilógico, sin embargo, es muy común. Existen una gran cantidad de cosas que el usuario no se da cuenta que necesita, o incluso que cree que necesita de determinada forma, y sólo al poner el sistema en práctica se percata de que lo requería de otra manera. Este tipo de situaciones son las que generan una gran cantidad de errores que corregir, mismos que en ocasiones provocan cambios estructurales al sistema y por consiguiente, una gran cantidad de tiempo, dinero y esfuerzo perdidos.

Lo que propone el uso del análisis de la tarea dista mucho de querer cambiar el modelo de desarrollo iterativo e incremental; más bien, lo que se intenta con este estudio previo es el ahorro de una gran cantidad de ciclos, ya que se parte de un entendimiento mucho más avanzado de lo que realmente desea el usuario.

Si bien el análisis de la tarea no es un concepto nuevo, hasta el momento las grandes empresas han optado por no utilizarlo debido a varios factores. Algunos de ellos son los siguientes:

- Falta de un proceso completo que lo contemple
- Tiempo
- Falta de pruebas y estadísticas que prueben sus beneficios
- Falta de herramientas que lo hagan viable

El primero de estos factores es de fundamental importancia: los desarrolladores requieren de procesos de desarrollo con el fin de lograr controlar el proceso y sobretodo la calidad de los productos del mismo. Es difícil y poco conveniente que un desarrollador decida realizar sus sistemas sin un proceso definido con el cual guiarse. En la actualidad existen una gran cantidad de procesos de desarrollo de software que se utilizan en la industria, sin embargo, ninguno de ellos contempla al análisis de la tarea como una de las actividades para lograr la captura de los requerimientos del usuario. Por otro lado, existen también una buena cantidad de procesos que si consideran al análisis de la tarea, sin embargo, estos procesos han sido desarrollados por los estudiosos del mencionado análisis y no consideran algunos factores esenciales en los procesos más utilizados. Además, por su carácter fundamentalmente académico, han sido rechazados por la industria.

Otro factor de gran peso es el tiempo que el realizar este análisis implica para el proceso de desarrollo. El análisis de la tarea de un sistema puede llegar a duplicar el tiempo de desarrollo, lo cual podría hacer imposible lograr contratos con los clientes, ya que a éstos siempre les resulta demasiado el tiempo y demasiado el costo.

Finalmente, y probablemente el factor decisivo para el abandono del análisis de la tarea por parte de las empresas, es el innegable hecho de que no se tienen pruebas contundentes y fuera de ambientes académicos de la eficacia de realizar este análisis.

Existen varias pruebas que se requeriría tener para convencer a los desarrolladores y sobretodo a los clientes de la utilidad del análisis de la tarea: por un lado, es necesario que se utilice el análisis de la tarea en una gran cantidad de proyectos con el fin de poder comparar el éxito de los unos con el de los otros.

Por otro lado se requiere también cuantificar de alguna forma la cantidad de dinero y de tiempo que se ahorraría una empresa en los ciclos de correcciones que ya no se realizarían al realizar el análisis de la tarea.

El cuarto punto se refiere a la falta de herramientas que hagan viable el empleo del análisis de la tarea. Sobre esto profundizaremos en la siguiente sección, ya que es a lo que se abocará la presente tesis.

2.4 *Planteamiento de la Tesis: La integración como posible solución*

Como se ha observado en las primeras dos secciones de este Capítulo, las comunidades de IS y de IHC tienen enormes virtudes, pero también carecen de elementos para erguirse por si solas con la solución para el desarrollo de un software de calidad y usable.

En este trabajo partimos de la hipótesis de que los trabajos de ambas comunidades han permanecido refractarios a los de la otra por varias razones (algunas de ellas fueron explicadas en la sección anterior); sin embargo, en este momento nos enfocaremos en que existe una falta de herramientas especializadas: La industria del software ha dado una gran importancia a la creación de herramientas que apoyen la producción de software complejo, en cada vez menor tiempo y con menos errores. Desafortunadamente, el vasto abanico de herramientas desarrolladas no contempla (salvo muy raras y notables excepciones) los aspectos del usuario, la tarea y el contexto que aquí se mencionaron (esto, además, es perfectamente congruente con el divorcio entre ambas comunidades que aquí se expone). Esto ha obstaculizado que dichas informaciones sean consideradas en los procesos de desarrollo, pues se argumenta que se trata de nueva información que debe ser gestionada manualmente además de toda la información que ya se genera, con el consiguiente incremente en el tiempo y esfuerzo necesarios. Se esgrime entonces que los proyectos se vuelven poco viables, por el tiempo y dinero que requieren.

En esta Tesis nos avocamos fundamentalmente a atacar el este problema; es decir, diseñar e implementar una herramienta que permita a un equipo de desarrollo, capturar y manipular las informaciones relacionadas con el usuario y su tarea, y que permita integrar dicha información, de manera clara y eficiente, a un proceso de desarrollo ampliamente aceptado por la comunidad de la ingeniería de software, de modo que la propuesta tenga oportunidades reales de impactar al medio.

Para estos efectos se parte del trabajo teórico que se ha desarrollado en el Laboratorio de Interacción Humano-Máquina del CCADET, en el que se han definido modelos para el análisis de la tarea del usuario, y se han establecido las equivalencias que permiten, a partir

de dicha información, establecer los requerimientos de la futura aplicación. Con el objetivo de que este proceso logre tener un impacto real en la Ingeniería de Software, dichos trabajos fueron inscritos en el marco de un Proceso de Desarrollo aceptado en la Ingeniería de Software, como lo es el Proceso Unificado. En particular, estos trabajos permiten establecer casos de uso a partir de la tarea del usuario. Desafortunadamente, como es de esperarse, el tener que realizar este proceso de manera manual lo vuelve inviable, pues los tiempos del proyecto crecen de manera demasiado importante.

Así, el objetivo es implementar los modelos y los métodos ahí desarrollados en una herramienta informática que permita a un grupo de desarrollo obtener los casos de uso en los que basará el desarrollo de la futura aplicación, a partir del análisis de la tarea del usuario. Es importante remarcar que esta implementación de modelos no es directa y que no carece de retos importantes, al punto que varios aspectos de los modelos propuestos deben ser revisados y adaptados para poder llevar a buen término dicho trabajo.

3 Análisis de la Tarea del Usuario

Si bien ya se explicó en que consiste el análisis de la tarea en términos generales, en este Capítulo mostraremos algunos detalles de importancia respecto a ésta actividad incluyendo la descripción de algunos de los modelos de la tarea más utilizados.

Finalmente presentaremos una descripción de los posibles usos que pueden dársele a un modelo de la tarea del usuario dentro del Proceso Unificado de desarrollo de software.

3.1 Qué tipos de modelos existen

Dependiendo de la notación, un Análisis de Tarea captura distintos tipos de información sobre el usuario y su tarea, por lo que éste se puede clasificar en:

Análisis Jerárquico de la Tarea: Toma como base una representación gráfica de la descomposición de una tarea de alto nivel en tareas más simples. El proceso consiste en identificar las tareas y sus objetivos, categorizarlas, descomponerlas y luego revisar la exactitud de la descomposición y del esquema.

Análisis Cognitivo de la Tarea: Se basa en técnicas que capturan con alguna forma de representación el conocimiento que la gente tiene, o que requiere, para completar una tarea. Gran parte de estas técnicas se basan en guías generales que se han desarrollado y que establecen los métodos apropiados para que los humanos interactúen con las computadoras.

Análisis de "Cómo hacer" de la Tarea: El objetivo principal es estudiar la efectividad del mapeo tarea - acción. Dado que el usuario conoce lo que se requiere hacer para obtener un objetivo, este análisis estudia las acciones que se requieren realizar. Esto se debe a que aunque los usuarios saben lo que deben de hacer para efectuar una tarea, al final pueden tomar otras acciones.

En particular, para este trabajo, utilizaremos un modelo del primer tipo. Si bien los tres tipos de modelo de la tarea aportan aspectos muy interesantes de la misma, el análisis jerárquico nos aporta aspectos de secuencia de las tareas así como de estructura y objetivos de la misma que nos resultan de mayor utilidad para lograr un sistema que responda a los mencionados objetivos. Por ejemplo, si nosotros requiriéramos modelar el conocimiento del usuario, como podría ser para un sistema experto, un modelo cognoscitivo de la tarea nos podría resultar muy útil, no así para. Por otro lado, el estudio de "Como hacer" la tarea, nos daría información sobre lo que el usuario "debería hacer" para lograr sus objetivos. Para la realización de un sistema, nuestro interés se centra en evitar que el usuario tenga que cambiar sustancialmente la manera en que "hace" las cosas, por ello, deberá estudiarse a fondo el orden en que realiza sus actividades y eso es bien reflejado en un modelo jerárquico.

Dentro de los modelos jerárquicos utilizaremos uno llamado MAD* que nos permite también guardar mucha más información de gran utilidad en la realización del sistema. A continuación se explicará en que consiste este modelo.

3.2 MAD*[2] (del frances : Méthode Analytique de Description)

MAD*[10] es un modelo jerárquico que permite modelar la tarea del usuario con gran detalle. Este modelo esta diseñado para que sea comprensible tanto por los ergónomos, como por los desarrolladores, de esta manera se establece una vía de comunicación entre estos.

3.2.1 Características principales

El modelo se basa en un análisis jerárquico que describe las tareas en términos de objetos, sus atributos, estados, objetivos, acciones y pre- y post-condiciones. Mediante un tipo especial de atributo llamado constructor se especifica la secuencia y flujo entre las tareas. Los constructores pueden operar de distintas formas dependiendo del estado de los atributos restantes de la tarea en la que se encuentra.

El análisis de tarea con MAD* comienza con entrevistas al usuario "no dirigidas", en las cuales el plantea lo que cree que son los requerimientos del sistema. De estas reuniones se obtiene, de forma iterativa e incremental, un árbol jerárquico de la tarea. Este árbol contiene al objetivo, las tareas y las acciones de la tarea global. Además de ello tiene una numeración de tareas y los constructores de cada una de ellas. La enumeración sirve para asignar una ficha de tarea a cada nodo del árbol MAD*.

3.2.2 Atributos

Las fichas que describen a cada uno de los nodos o tareas del árbol MAD* se componen de varios atributos que tienen. Estos son descritos a continuación:

- ***Número***: Numérico, representando la posición de la tarea en el árbol.
- ***Nombre***: Alfanumérico, nombre de la tarea.
- ***Objetivo***: Alfanumérico, descripción del objetivo de la tarea.
- ***Prioridad***: Entero, permite resolver conflictos entre tareas que quieren ejecutarse simultáneamente.
- ***Facultativa***: Booleano, indicando si la tarea es obligatoria o se puede omitir su ejecución.
- ***Interruptible***: Entero. 0 indica no interruptible, 1 interruptible con posibilidad de continuar la tarea, 2 indica interruptible y al regresar se reinicia la tarea, y 3 indica interruptible y no se regresa a la tarea.

[2] STAR (del Frances tâchéS orienTé spécificAtion d'inteRfaces)

- *Tipo*: Alfanumérico, indicando si es sensorimotriz o cognitiva.
- *Constructor*:
  - **Secuencial**: Las subtareas se ejecutan en el orden especificado. La ejecución termina al haber ejecutado todas las subtareas obligatorias, y las optativas han sido ejecutadas o han sido omitidas.
  - **Paralelo**: Las subtareas se ejecutan en cualquier orden y sin compartir datos. La ejecución termina al haber ejecutado todas las subtareas obligatorias, y las optativas han sido ejecutadas o han sido omitidas.
  - **Simultaneo**: Las subtareas se ejecutan de forma simultanea y sin compartir datos, lo que implica que son realizadas por distintos usuarios de forma simultanea. La ejecución termina al haber ejecutado todas las subtareas obligatorias, y las optativas han sido ejecutadas o han sido omitidas.
  - **Y**: Las subtareas se ejecutan de forma simultanea y compartiendo datos. La ejecución termina al haber ejecutado todas las subtareas obligatorias, y las optativas han sido ejecutadas o han sido omitidas.
  - **O**: Las subtareas se ejecutan en cualquier orden y comparten datos. La ejecución termina cuando al menos una de las subtareas obligatorias se ha
  - completado.
  - **Alternativa**: Solo una de las subtareas es ejecutada.
  - **Elemental**: La tarea no tiene subtareas. Este constructor indica que el nodo es una tarea simple o una acción que se ejecuta directamente.
- *Pre-condiciones*:
  - **Condición de inicio**: Estado del mundo antes de que se ejecute la tarea.
  - **Condición de ejecución**: Estado del mundo que dispara la ejecución de la tarea. El estado inicial debe de cumplirse para que la tarea arranque.
  - **Condición de paro**: Estado del mundo que indica cuando detener la ejecución de la tarea (y de sus subtareas). Este atributo es útil si se quiere que la ejecución de la tarea se repita hasta lograr una condición.
- *Post-condición*: Estado del mundo después de que la tarea termino su ejecución.

3.2.3 Ejemplo(s)

A continuación mostraremos algunos ejemplos de tareas modeladas mediante MAD*.

El primer ejemplo es el árbol de la tarea de enviar un e-mail (ver Figura 9). Se seleccionó esta tarea por la simplicidad de la misma y ya que permite mostrar los conceptos básicos de los árboles MAD*.

La tarea modelada es "mandar correo", esta tarea es comúnmente parte de una tarea más general llamada "administrar correos"; sin embargo, para efectos de la simplicidad del ejemplo tomaremos a la tarea como si la única labor a realizar mediante el nuevo sistema fuese mandar correos. Otro aspecto importante a destacar es el carácter facultativo (no indispensable) de las tareas de entrar y salir de Internet así como de la sesión de la cuenta de correo ya que dependiendo del ambiente en el que se esté y del usuario del sistema, las tareas podrían o no tener que realizarse.

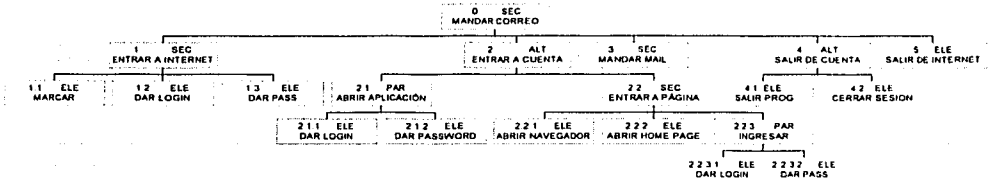


Figura 9 "Árbol de la tarea Mandar un e-mail"

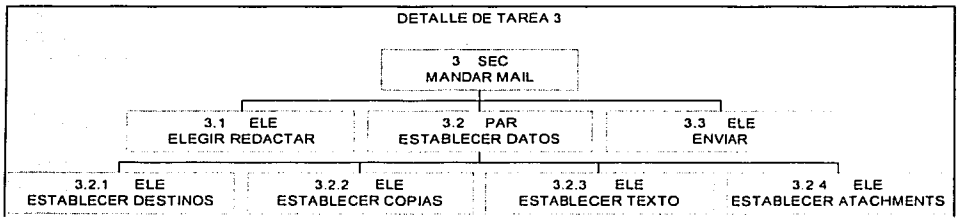
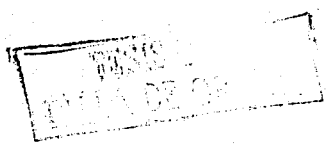


Figura 10 "Detalle de la tarea mandar mail, dentro de la tarea mandar correo"

La tarea "mandar mail" es secuencial y por lo tanto las tareas de la uno a la cinco deberán realizarse en ese orden. La tarea entrar a cuenta requiere que la conexión se encuentre habilitada, este tipo de condiciones se establece en las fichas mostradas a continuación.

Número: 0	Nombre: Mandar correo	Constructor: SEC
Objetivo: Enviar un mensaje y/o archivos a una o más personas		
Precondición de inicio:		
Precondición de arranque: Conexiónf.habilitada=T		
Precondición de paro:		
Post Condición: Mensaje.enviado=T		
Prioridad: 10	Obligatoria: True	Int: Reinicia

Ficha de tarea principal



Número: 1	Nombre: Entrar a internet	Constructor: SEC
Objetivo: Lograr una conexión a internet		
Precondición de inicio:		
Precondición de arranque: Conexiónf.habilitada=T		
Precondición de paro:		
Post Condición: ConexiónL.habilitada=T		
Prioridad: 10	Obligatoria: F	Int: Reinicia

Ficha de tarea uno

En estas fichas podemos observar que el objetivo de la tarea más general es el de enviar un mensaje y o archivos a una cuenta. En este caso se trata de un objetivo fundamental de la tarea (no siempre es el caso en que se encuentre en la tarea más global). Otro punto a destacar es el como se establecen las pre y postcondiciones mediante valores de las propiedades de los objetos, por ejemplo, se destaca que para que la tarea uno se pueda realizar la propiedad habilitada del objeto "conexión física" deberá ser verdadera. Se observa también que al finalizar la tarea el estado de la conexión lógica es de habilitada también.

A continuación mostramos la ficha de una tareas más para ejemplificar algunas otros aspectos:

3.2	Establecer datos	PAR
Establecer la información necesaria para enviar el correo		
Anychosen.redactar=T		
Datos.completos=T		
Datos.completos=T		
10	T	Cont

Ficha de la tarea 3.2

Como puede observarse la tarea es paralela, esto significa que las subtareas (establecer destinos, establecer copias, establecer texto y establecer atachments) se realizan sin una secuencia definida, es decir, se comienza a realizar cualquiera de ellas y en cualquier momento es posible suspender para continuar con otra, interrumpirla y seguir con otra o con la primera y así sucesivamente hasta terminar con todas las subtareas no facultativas.

También es de destacarse el valor Cont (Continúa) en la propiedad de interruptible. Esto implica que cuando se están estableciendo los datos del mensaje, el usuario puede interrumpir la labor, atender cualquier asunto que surgiese y regresar a "continuar" con su labor.

3.3 Posibilidades de uso de MAD* en el PU

El tener un modelo de la tarea puede resultar de gran ayuda para un proceso de desarrollo de software como el PU. Existen actividades del diseño del sistema que podrían ser realizadas con mayor facilidad y con una base teórica más sólida mediante el análisis de la tarea, algunas de ellas son:

- Definición de clases
- Generación de interfaces de usuario
- Comprensión del contexto del sistema

Respecto al primer punto debemos recordar que el modelo de la tarea MAD* expresa, a través de las pre y post condiciones del sistema, los objetos implicados en cada una de las subtareas así como los estados en que estos objetos se pueden encontrar, es decir, los atributos que poseen y sus posibles valores antes y después de cada tarea. Si tomamos en cuenta esta información es posible utilizarla como base para definir las clases (y sus atributos) que tendrá el sistema que se va a desarrollar.

Otro punto de gran utilidad que se puede obtener a partir del modelo MAD* de la tarea son las interfaces de usuario. Si se parte de lo que el usuario presenta como tareas y subtareas se puede obtener una relación con lo que serían las interfaces de usuario. Para esta relación no se tienen mapeos directos que se puedan aplicar automáticamente ya que definitivamente la labor de traducción será siempre creativa y por lo tanto humana; aún así, la información obtenida a partir del árbol MAD*, puede resultar una buena guía. La mayor parte de ésta información se obtiene mediante los constructores del árbol; por ejemplo, si se tiene un constructor secuencial, las tareas deberán realizarse en el orden especificado; ésto implica que deberá ser presentado en una interfaz tipo *wizard*. Por otro lado, si se tiene una tarea que se realiza de manera paralela, es posible que se requiera un *panel de tabuladores*; si se tiene una tarea opcional, deberá presentarse algún componente de selección como un *combo box* o un *radial button* para seleccionar la tarea.

Para trabajos de futuras tesis, se podría realizar una herramienta que asista en la obtención de prototipos rápidos a partir de un árbol de la tarea. Esto, sin duda, ayudaría a la agilización de la presentación de prototipos y, en consecuencia, a la Usabilidad de los productos obtenidos. Otra posibilidad sería una herramienta que cree las clases mediante la información que provee el árbol y que permita, de alguna manera, ligarlo a alguna herramienta ya usada en el medio.

El tercer punto, la "Comprensión del contexto del sistema", es el aspecto en el que nos concentraremos en este trabajo. Como ya observamos en secciones anteriores, el Proceso unificado no define la manera en que se puede lograr la mencionada comprensión. Para ello se propone la realización de un análisis de la tarea.

El análisis de la tarea se ha mostrado como una técnica de gran eficiencia para comprender el ámbito en el que trabajará el sistema. A través de técnicas como las entrevistas "dirigidas" y "no dirigidas", es posible lograr que el usuario exprese más sobre su tarea y el

contexto en que trabaja que mediante algunos otros artefactos como "cuestionarios escritos" o "textos explicativos sobre su función" que el cliente entrega.

Por otro lado, la realización de un árbol de la tarea MAD exige, del analista, tanta información de la tarea del usuario que resulta un excelente medio para ayudarle a saber cuando, efectivamente, tiene toda la información que requiere. Sin el árbol, es muy fácil que el analista considere tener la información completa cuando en realidad todavía hay aspectos que no conoce.

4 Herramientas para generar software

La ayuda que una buena herramienta puede proporcionar en el desarrollo de software ya ha sido motivo de estudios, encontrándose reducciones de hasta cinco veces en el tiempo de desarrollo, como fue el caso con el sistema MacApp de Apple.

Existen otros datos impresionantes que muestran el gran aumento que el mercado de las herramientas está teniendo, alcanzando cifras de hasta 133 millones de dólares en herramientas para UNÍX únicamente en 1993 [21]. Evidentemente esta cifra debe ser ridícula comparada con mercados como Windows en fechas más recientes.

Por ejemplo, de acuerdo con Brad A. Myers en 1996[21], el desarrollo de interfaces de usuario involucraba cuatro tipos de persona: el usuario, el diseñador y el programador, como es aceptado generalmente, y adicionalmente, contempla al programador de herramientas. Esto significa que en un equipo de desarrollo se contemplaba una parte importante dedicada únicamente a desarrollar herramientas que utilizará el diseñador, de manera que realice mejor y más fácilmente su trabajo. Si bien esto ha cambiado desde 1996, lo cierto es que da una idea de la importancia que la creación de más y mejores herramientas estaba cobrando y sigue teniendo dentro del desarrollo de software.

Sin embargo, desarrollar herramientas tiene un costo que puede llegar a ser alto. En ese contexto, antes de desarrollar una nueva herramienta cabe preguntarse ¿Qué se espera lograr con ella? Este es de hecho, uno de los puntos centrales de este trabajo, y se discutirá en detalle más adelante en este Capítulo. Antes de ello se mostrará brevemente la forma en que las herramientas de software han evolucionado con el tiempo. Enseguida daremos una descripción de las herramientas existentes para la captura de requerimientos. Para finalizar se discuten las virtudes y desventajas de las herramientas actuales, y con ello, se verán las necesidades que nuestro trabajo pretende cubrir.

4.1 Evolución de las herramientas

El desarrollo de software ha evolucionado desde el uso del código convencional en lenguajes como el ensamblador, a las herramientas de prototipaje rápido de nuestros días, y a nuevos paradigmas que permiten diseñar los sistemas basándose en modelos con lo que se reduce cada vez más la labor de programación.

4.1.1 Los primeros desarrollos

En un principio se producía software mediante el uso de lenguajes de muy bajo nivel de abstracción; es decir, lenguajes que utilizaban comandos que la computadora entendía prácticamente sin traducción alguna. Ejemplos de este tipo de lenguajes son el lenguaje ensamblador. Con un mayor nivel de abstracción pero aún con un lenguaje más cercano al de la computadora que al del usuario se tenía el lenguaje C.

4.1.2 Las bibliotecas

Al pasar el tiempo, se generó una gran cantidad de fragmentos optimizados de código que se utilizaban con frecuencia, a lo que se les llamó bibliotecas. Éstas se pusieron a disposición de la comunidad de programadores y se comenzaron a reutilizar, lo que facilitó la labor del programador al librarlo de muchas tareas rutinarias, al tiempo que se logró estandarizar un poco la estructura de los programas.

El uso de las bibliotecas se llevó más allá al desarrollar "plantillas" de programas completos para sistemas de producción frecuente. De esta manera se reducía buena parte de la labor del programador, ya que únicamente tenía que codificar aquellas partes del programa que lo diferenciaban del resto, independientemente de aquellas partes que quisiera cambiar de esta "plantilla".

Tanto las plantillas, como las librerías de código, lograron facilitar la labor de los programadores de lenguajes como C. Sin embargo, el uso de las bibliotecas no fue suficiente para manejar la creciente complejidad de los sistemas por lo que fue necesaria la creación de lenguajes con mayores niveles de abstracción, llamados lenguajes de alto nivel como el SQL o aún Lingo® de Macromedia™.

4.1.3 Herramientas

Estos avances en la forma de producir código lograron facilitar la labor de los programadores. Sin embargo, el apoyo obtenido sólo es cierto en términos de desarrollar sistemas similares; no obstante, es de mencionarse que las tareas que un programador debe cumplir resultan cada vez de un mayor grado de complejidad. Por ello, los avances que se tienen en estas y otras ayudas más recientes apenas logran amortiguar la creciente dificultad de los problemas a resolver. Esto provoca, además, que cada vez resulte más impensable que un solo programador se encargue de un sistema completo por lo que las herramientas empiezan a considerar el trabajo distribuido dentro de sus arquitecturas.

Es por esta creciente complejidad que en los últimos años se han desarrollado gran cantidad de herramientas que permiten realizar labores de enorme complejidad de manera relativamente sencilla. Ejemplo de esto son los "builders" como Visual Basic® de Microsoft™ o JBuilder@ de Borland™, que se han desarrollado para facilitar el uso de los lenguajes de alto nivel, o incluso las herramientas que tienen su propio lenguaje de muy alto nivel como Director® y su "lingo"®.

Por otra parte, es necesario considerar que la historia del desarrollo de software ha incluido, dentro de su evolución, la creación y uso de herramientas para las distintas etapas del proceso, así como para los distintos tipos de desarrollo. Se han citado ya ejemplos de herramientas disponibles actualmente para el desarrollo del software de diversos tipos (ej.

Director®, Visual Basic®, etc); falta agregar, sin embargo, la gran cantidad de herramientas que se han desarrollado para otras etapas de la creación de un sistema.

4.1.4 Herramientas en la ingeniería de software y en IHC

En particular, la comunidad de ingeniería de software ha logrado, en últimas fechas, conformar procesos de desarrollo de software bastante completos como el Proceso Unificado, el TSP o el XP. Las distintas etapas de estos procesos han sido soportadas por diversas herramientas desde las de programación, las de diseño de bases de datos, diseños de interfaz e incluso herramientas para la administración del proyecto o para su documentación. Aun así, todavía quedan áreas de gran importancia que no han sido soportadas por herramienta alguna. Es el caso de la llamada "Captura de Requerimientos", parte esencial en cualquier proceso de desarrollo.

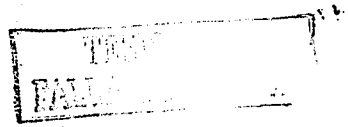
En particular, en el mercado existen dos herramientas que pretenden asistir a todo el proceso de desarrollo. Un pequeño análisis de las mismas nos muestra sus carencias en la etapa particular de captura de requerimientos.

La primera de estas herramientas es el *Rational® Requisite Pro®* [14]. Esta herramienta permite llevar un registro detallado de todos los requerimientos del sistema, una bitácora del estado en que se encuentran, un registro de las características de cada requerimiento, etc. En términos generales se observa que permite un correcto registro y seguimiento de los requerimientos, aún así, no contempla ningún módulo que asista en la captura de los mismos. .

Por otro lado, el *Visual Studio .Net* de *Microsoft* [15]ni siquiera tiene una herramienta para el registro y seguimiento de los requerimientos; si bien es posible usar algunas herramientas de planeación como el Project® para registrar los requerimientos, también es cierto que no están hechas para ello y que por supuesto, no asisten de ninguna manera a la captura de los requerimientos.

La captura de requerimientos es una de las actividades más importantes en términos de los problemas y costos que causa a un proyecto. A pesar de ello, es también una de las actividades menos dominadas, y es por fallas en esta captura que se tienen graves problemas en las etapas posteriores de los proyectos.

El problema de la captura de requisitos, por otro lado, es atacado de mejor manera por una comunidad académica dedicada al estudio de la interacción humano-computadora. Esta comunidad ha estudiado por varios años los conceptos de análisis de la tarea y del usuario (explicados en el primer capítulo del presente trabajo) en base a lo cual, ha desarrollado herramientas basadas en modelos de la tarea del usuario, que permiten la creación de interfaces de usuario e incluso el diseño de la futura aplicación. Aún así, estas herramientas no son utilizadas en el medio del desarrollo de software por varias razones que a continuación se analizan:



Esta comunidad, al igual que la comunidad de ingeniería de software y que la mayor parte de las comunidades académicas, ha querido independizarse del resto. Esto ha provocado que pretendan que el desarrollador decida, de *motu propio*, adoptar su perspectiva sobre como desarrollar software. Sin embargo, la comunidad de IHC no ha desarrollado, hasta la fecha, un proceso de desarrollo de software sólido y lo suficientemente completo para basar en él el desarrollo de un software de gran tamaño. Si así hubiese sido, aún tendrían que enfrentar el problema de que su proceso fuese bien difundido de manera que la industria lo tomara en cuenta. Sin embargo, uno puede observar que esto no es así, y no hay ningún sentido en pensar que sucederá. La verdad es que los procesos de desarrollo de software utilizados son otros y las herramientas que se producen en IHC tienen demasiado poco en común con ellas para poderse comunicar, por lo que la mayor parte de la industria decide simplemente ignorar las propuestas de IHC.

Así, aunque parezca que las herramientas han evolucionado suficiente, esto nunca es cierto. El reto actualmente, es la producción de herramientas que soporten las partes de los procesos de desarrollo de software que aún deben realizarse manualmente, lo que incluye, por supuesto, la captura de requerimientos. Otro reto mayor es lograr que los resultados de esta etapa mejoren mediante el uso del análisis de la tarea del usuario final. Esta actividad permitirá relacionar la información de la tarea del usuario con los datos, artefactos y procesos utilizados en la ingeniería de software. En relación con esto se han desarrollado varios proyectos que aportan valiosos elementos y que examinaremos a continuación. Aún así, falta mucho por desarrollar y es en este sentido que el presente trabajo presenta una propuesta.

4.2 Herramientas en la captura de requerimientos

En la presente sección introduciremos las herramientas "basadas en modelos", en general, y en particular las "basadas en modelos de la tarea". Una vez que tengamos definida esta diferencia explicaremos cuales son algunos de los desarrollos actuales en el área de las herramientas basadas en modelos de la tarea. Finalmente se dará una breve discusión sobre las herramientas descritas incluyendo sus ventajas y sus carencias.

4.2.1 Herramientas basadas en modelos

En la sección "Evolución de las herramientas" se mostró el proceso que se ha tenido en el desarrollo de software para pasar, de la codificación en lenguajes de bajo nivel, a la programación mediante lenguajes de muy alto nivel de abstracción. El siguiente paso en esta evolución aún no ha podido ser concretado en la industria y se refiere a las llamadas "herramientas basadas en modelos"; el propósito de este tipo de herramientas es el de lograr desarrollos de software a partir de "modelos" o especificaciones. De acuerdo con esta óptica, el desarrollador podría especificar *qué* debe hacer la herramienta y *cuándo* lo debe hacer; la herramienta debería, a partir de estas especificaciones, generar la aplicación deseada, lo que generalmente implicaría generar el código de la aplicación. A estas herramientas se les conoce como "Basadas en Modelos".

Las herramientas basadas en modelos aún no son una realidad en la industria; sin embargo, sí se tiene un gran avance en este sentido ya que proyectos como MDA (del ingles: Model Driven Architecture) del Grupo de Administración de Objetos (OMG del ingles: Object Managment Group) [13] tienen como objetivo la creación de herramientas basadas en modelos del sistema hechos en UML. A partir de estos modelos se propone la creación del código para cualquier lenguaje, es decir, se tendrían modelos del sistema independientes del lenguaje de programación, de la plataforma de desarrollo, etc. Aun así, el proyecto todavía tiene mucho por desarrollar y las herramientas que lo soporten al cien por ciento aún no están disponibles.

Para la especificación del sistema en UML se presenta el problema de que UML es un lenguaje gráfico, por lo que había que codificarlo de alguna manera en texto. El Formato de Modelado de Objetos (MOF del ingles: Model Object Format), definido por la OMG permite especificar los mismos elementos que UML, a partir de este estándar se creo el lenguaje de Intercambio de Metadatos XML (XMI, del ingles **X**ML **M**etadata **I**nterchange). Mediante este subconjunto de XML es posible representar los elementos que se representan en UML en formato XML. Gracias a éste lenguaje es posible tener una especificación completa de un sistema en un archivo de texto codificado en un lenguaje absolutamente independiente a la plataforma y lenguaje de programación.

En la industria se tienen ya herramientas que buscan ese objetivo y cada vez se acercan más a lograrlo, aún así, un breve análisis de las mismas todavía encuentra grandes deficiencias. Un ejemplo de esto es el Rational Rose®. Esta herramienta intenta, a partir de especificaciones en UML, generar código que sirva como plantilla para el desarrollo de la aplicación, es decir, Rational intenta generar las especificaciones de clases, atributos y métodos en el lenguaje que el analista desee partiendo de una especificación en UML. Evidentemente, la ayuda que provee Rational es importante; sin embargo, dista mucho de generar el código de la aplicación completa.

Otro proyecto que se acerca a la meta de producir software basándose en modelos es el Visual Studio® de Microsoft™. En esta herramienta es posible dibujar las clases del sistema en UML y obtener, con base en este modelo, el código en alguno de varios lenguajes. Evidentemente, está lejos del objetivo de lograr el código de toda la aplicación ya que únicamente permite definir clases, atributos de las mismas y métodos en ellas.

Hasta aquí hemos analizado el alcance de las herramientas actuales, en términos de lograr código basándose en un modelo. Aún con el gran avance que se tiene en el desarrollo de herramientas es de destacar que la captura de requerimientos no es soportada por ninguno de estos proyectos; y como se enfocan fundamentalmente al diseño de la aplicación ni siquiera contemplan el asistir de alguna manera en la captura de los requerimientos; es decir, una vez que se hayan definido los requerimientos las herramientas existentes ayudarán mucho para diseñar e implementar un sistema. Como vemos, las herramientas actuales ayudan a *diseñar*, *modelar* e *implementar* el sistema; sin embargo, no asisten para encontrar el *qué* debe hacer el sistema.

Si bien acabamos de comentar que existen o se pretende que existan herramientas basadas en modelos es bueno puntualizar que cuando se habla de modelos se habla de algo muy general, de hecho, se pueden modelar muchas cosas dentro del proceso de desarrollo de un software. Por ejemplo, se puede modelar la interfaz que se presenta al usuario, se puede modelar el diálogo que ésta tendrá con el mismo, etc. Como podremos observar en la siguiente sección existen muchos proyectos que basan el desarrollo de aplicaciones en el modelo de la tarea del usuario. Existen, por otro lado, proyectos que basan el desarrollo de aplicaciones en otros modelos.

Un ejemplo de proyecto cuya herramienta basa su desarrollo en modelos es el proyecto ITC de IBM. En este proyecto, la herramienta tiene una arquitectura de 4 capas relacionadas con modelos del diálogo, de la interfaz y de la aplicación, además de una capa de reglas que los relacionan. Este tipo de proyectos tienen gran utilidad, al grado de que ITC fue utilizado para el sistema de la EXPO SEVILLA 92 [21]. Aún así, es evidente la ausencia del modelo de la tarea.

Hasta el momento hemos comentado el estado de las herramientas basadas en modelos. A pesar de ellos, existe un modelo en particular cuya importancia fue comentada ya en el Capitulo 2. Debido a la importancia particular de este modelo también es importante no confundir a las herramientas basadas en modelos con aquellas basadas, específicamente, en modelos de la tarea, que son las que conciernen al presente trabajo. A continuación comentaremos los estudios existentes en la actualidad con este motivo.

4.2.2 Herramientas basadas en el modelo de la tarea

El área del desarrollo de software no ha permitido el ingreso del análisis de la tarea en sus procesos más populares. En el mundo existe, sin embargo, un trabajo considerable alrededor de la creación de herramientas basadas en modelos de la tarea del usuario. De hecho, en el aspecto particular del desarrollo de prototipos existen algunos trabajos que han integrado, con éxito, el modelo de la tarea al modelo del prototipo. Entre ellos se destacan ADEPT[16], MASTERMIND [25], DIANE [20; 19] y TRIDENT [29]. Cada uno de estos proyectos tienen aspectos muy positivos que se retomarán en el presente trabajo, por lo que vale la pena dar una pequeña descripción de cómo trabaja cada uno de ellos, y sobre qué conceptos descansa su desarrollo teórico.

En el caso de la propuesta de este trabajo, presentamos una diferencia fundamental con todos estos trabajos y que dará a la herramienta mayores posibilidades de éxito que en los trabajos que se mostrarán. Esta diferencia consiste en aquello que se obtiene a partir del análisis realizado. En efecto, en los trabajos que se han realizado hasta el momento se tiene como objetivo una interfaz de usuario "centrada en el usuario". A pesar de ello, estos trabajos no han logrado incorporarse a ningún proceso de desarrollo de software utilizado en la industria. La propuesta que aquí se presenta traduce el resultado del análisis de la tarea a lenguaje XMI, de modo que se representan los elementos UML; es decir, el lenguaje estándar actual dentro de la mayor parte de los procesos de desarrollo de software. Además de esto, el resultado se importa desde el Rational Rose®, programa líder de la industria

para la documentación de procesos de desarrollo de software, lo que le añade posibilidades de ser aceptado en la industria. De esta manera será posible que más desarrollos incorporen al análisis de la tarea y, como consecuencia, mejore el nivel de usabilidad de los productos que se obtengan.

4.2.2.1 DIANE

DIANE [20]es un proyecto que presenta modelos para varios aspectos del sistema(la tarea, la interacción, la interactividad o diálogo y la implementación). Uno de ellos es el modelo de la tarea, con información que permite desprender de él una interfaz de usuario. En este modelo se expresa la secuencia en que se realizan las tareas, la forma en que éstas se descomponen, quién o quienes debe realizar cada una de ellas y algunas otras características como la interactividad que tiene la tarea, misma que permitiría deducir otros aspectos como: el tipo de widget que se deberá utilizar en la interfaz final.

Existen, sin embargo, algunas carencias en la expresividad de su modelo de la tarea. Una de ellas es que no contempla la posibilidad de que una tarea sea interrumpida, ya que no establece si esto puede suceder, y mucho menos define que se hace al terminar la interrupción.

Otra carencia del proyecto se encuentra en la falta de un modelo diferente para la tarea antes y después de utilizar el nuevo sistema; es decir, un modelo que muestre la tarea del usuario actual y otro que la muestre como se realizará mediante el nuevo sistema.

El proyecto DIANE, por otro lado, tiene una gran virtud que se encuentra en el hecho de que no sólo utiliza el modelo de la tarea; de hecho, cuenta con una etapa de gran importancia: la obtención de los objetos de la tarea; estos objetos son utilizados en otros modelos como el modelo de interacción, el de implementación y el de interactividad o diálogo del sistema. El modelo de interacción es similar al de la tarea, sólo que presenta modificaciones al mismo con el fin de presentar características adicionales como la interactividad u obligatoriedad de la tarea.

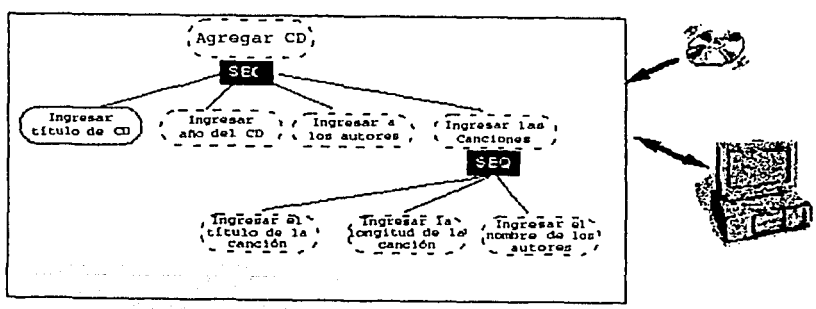


Figura 11 "Ejemplo de modelo de la tarea -Agregar CD - en DIANE"

En la Figura 11 se muestra la tarea de agregar un CD a la base de datos mediante DIANE. Como se puede apreciar, las tareas se simbolizan en rectángulos redondeados con líneas punteadas en el caso de tareas opcionales, o con líneas continuas, en el caso de tareas obligatorias. El rectángulo negro bajo la tarea principal y bajo "establecer las canciones" muestra la forma en que las tareas derivadas se deberán ejecutar, en este caso, secuencialmente.

Finalmente, las líneas que salen de la primera tarea indican que establecer el título, el año, los autores y las canciones del CD son parte de la tarea "Agregar CD a la BD" y deberán realizarse en ese orden. Así mismo, "establecer canciones" se compone de establecer título, longitud y nombre del autor de cada canción.

Una novedad en DIANE es el modelo de implementación, ya que da un paso adelante en el posible uso o traducción del modelo de la tarea para la realización del sistema, esto, ya que en el se definen los objetos del sistema; al tener definidos los objetos, éstos pueden ser programados directamente como clases según el paradigma orientado a objetos.

Si bien es cierto que la traducción del modelo de la tarea al de implementación la realiza el diseñador del sistema con base en su criterio debemos conceder también que el modelo de implementación obtenido puede ser plasmado, mediante lenguajes comúnmente usados como UML, en lo que se conoce como diagrama de clases. Por ello, mediante el modelo de implementación se tiene una buena idea de cómo se podría organizar el sistema en un lenguaje orientado a objetos.

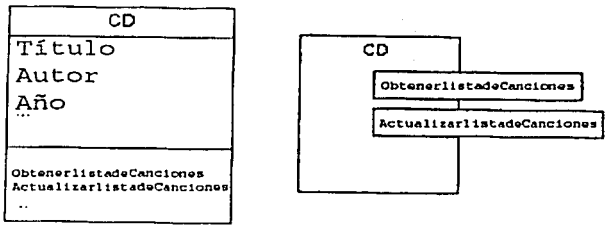


Figura 12 "Clases, atributos y métodos obtenidos de la tarea –Agregar CD- mediante DIANE

En la Figura 12 se observa como, de la tarea "Agregar CD a BD" se obtiene la clase CD con atributos como el título, los autores y el año de producción. Así mismo, se muestran los métodos "obtener lista de canciones" y "actualizar lista de canciones".

Para concluir, debemos decir que si bien DIANE no presenta avances significativos en términos de una herramienta, si lo hace en términos de la traducción a objetos. Esto cobra gran importancia si se considera que la orientación a objetos comienza a dominar el ambiente del desarrollo de software.

4.2.2.2 TRIDENT

El objetivo fundamental de TRIDENT (del ingles: Tools foR an Interactive Development EnvironmeNT) [29] es lograr una herramienta que permita la generación de prototipos rápidamente y que estén basados en un modelo de la tarea del usuario.

La definición del proyecto propone una estructura con un núcleo semántico y un modelo del diálogo que permita separar los procesos propios de la aplicación de los procesos de comunicación con el usuario.

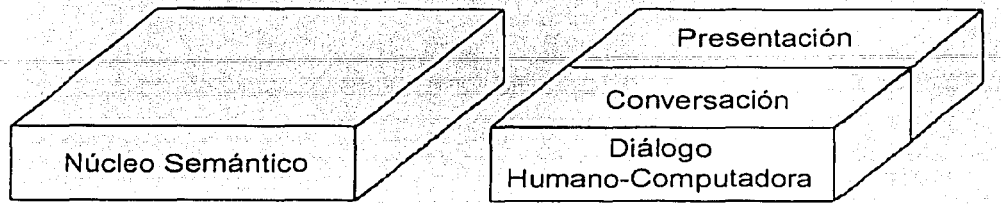


Figura 13 "Arquitectura de TRIDENT"

El diálogo a su vez tiene una parte dinámica llamada "Conversación" y una parte estática, llamada "Presentación" (Ver Figura 13). La primera de ellas contempla las características típicas de una interfaz de usuario moderna, como lo es el multiproceso y la asincronía con la que el usuario interactúa. La segunda parte, la presentación; considera los criterios ergonómicos existentes, así como las guías de estilo a fin de lograr una mejor interfaz.

En términos generales, TRIDENT establece que al usuario se le debe proveer de las herramientas necesarias para lograr prototipos de la manera más automática posible y que el diálogo con el usuario debe estar basado en el modelo de la tarea.

Como ya se mencionó, la arquitectura de TRIDENT maneja un núcleo semántico, un diálogo (la conversación) y una presentación. El núcleo semántico permite mantener la consistencia entre lo que se presenta al usuario y lo que se tiene en el modelo de la interfaz. Además de ello, el núcleo semántico se encarga de la comunicación con el manejador de la base de datos.

El núcleo semántico y la presentación, a su vez, interactúan con otros componentes externos y se logra así un sistema completo. Como se puede observar en la Figura 14 , el núcleo semántico interactúa con elementos como el manejador de la base de datos a través de un administrador semántico. Éste tomará los comandos del núcleo semántico y los traducirá en los comandos necesarios para el manejador de la base de datos, finalmente, el manejador se comunicará con la base de datos.

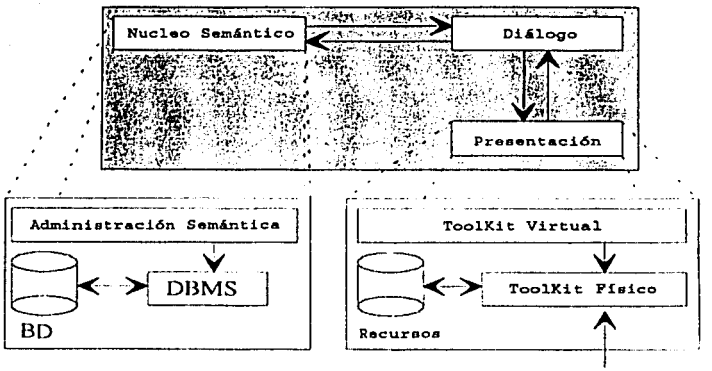


Figura 14 "Arquitectura Externa de TRIDENT"

La presentación, por otro lado, interactúa con otros elementos como lo son los Toolkits virtuales y físicos. Los *toolkits*, o conjuntos de herramientas virtuales se refieren a los elementos de diálogo abstractos como lo es un campo de texto, una etiqueta o una lista desplegable. Los *toolkits* físicos son implementaciones particulares de los *toolkits* virtuales como lo podría ser el *jTextField* de Swing en Java, el *textField* en AWT del mismo Java o un campo de texto *ActiveX* en Visual Basic.

La forma en que esta arquitectura se implementa en el proyecto TRIDENT es mediante una serie de objetos en cada uno de los componentes. La Figura 15 muestra como los objetos se comunicarán tanto con los demás objetos del componente como con los que se encuentran fuera de él.

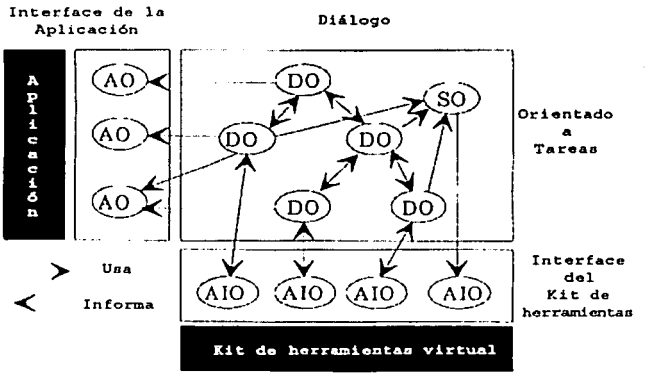


Figura 15 "Detalle de la arquitectura de TRIDENT"

El componente del núcleo semántico se forma de objetos de la aplicación (AO del ingles: Aplication Objects) como lo podrían ser clases en un lenguaje orientado a objetos, objetos

COM en una aplicación de Visual Basic, etc. En el diálogo se tienen objetos de diálogo (DO del ingles: Dialog Objects) y objetos supervisores (SO del ingles Supervisor Objects) que se encargarán de mantener el orden de ejecución de los DO. Finalmente la presentación consiste también en dos tipos de objetos, los objetos de interacción(IO del ingles: Interaction Objects) y los objetos abstractos de interacción(AIO del ingles: Abstact Interaction Objects). Las diferencias entre ellos son las ya mencionadas entre toolkits reales y virtuales respectivamente. Cada objeto de cada uno de los módulos se compone de eventos generados y recibidos, de memoria y de primitivas o métodos que puede ejecutar (Ver Figura 16).

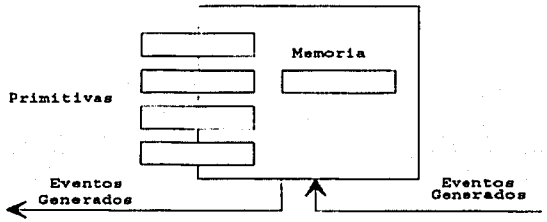


Figura 16 "Objeto genérico en TRIDENT"

El proyecto TRIDENT modela el diálogo con el usuario mediante dos métodos principalmente: por un lenguaje de reglas y mediante diagramas de estados. El lenguaje de reglas se refiere a sentencias de implicación donde si se tienen ciertas condiciones se disparan ciertas acciones. Los diagramas de estados, por otro lado, permiten expresar la forma en que se comporta un objeto mediante nodos de un diagrama que simbolizan las situaciones más comunes en que un objeto se puede encontrar y arcos entre estos nodos que indican mediante que acciones se pasa de una situación a otra.

A continuación se presenta una ventana modelada por diagrama de estado y por reglas:

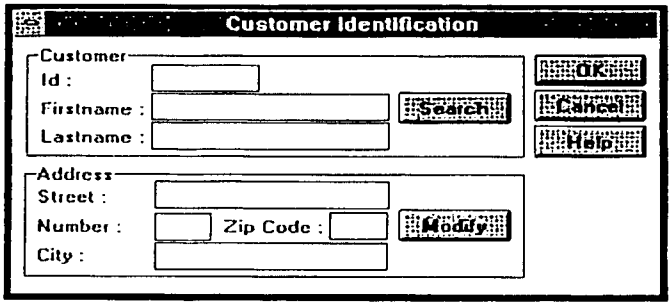


Figura 17 "Ventana de identificación de cliente"

En la Figura 17 se muestra la ventana de identificación de cliente con campos como el identificador, el nombre y los apellidos del mismo así como su dirección. La ventana permite buscar y modificar los datos antes de aceptar o cancelar la operación. También permite solicitar ayuda.

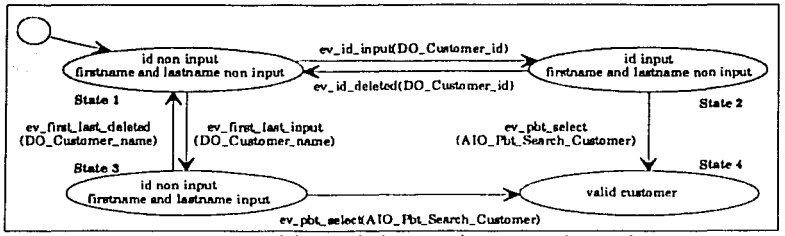


Figura 18 "Modelo del diálogo en diagramas de estado"

En la Figura 18 , por otro lado, se muestra un diagrama de 4 estados que corresponden a las posibles combinaciones entre establecer o no el identificador y el nombre completo del cliente. Cada uno de los primeros tres estados (correspondientes a los casos en que se tiene establecido uno o ninguno de los datos mencionados), recibe eventos referentes al establecimiento de alguno de los datos faltantes, o al borrado de alguno de los datos que se encuentran establecidos. En caso de encontrarse en un estado en que un dato está establecido y recibirse un evento que establezca el otro dato, se pasa al cuarto estado ("cliente válido") en el cual no se reciben más eventos.

```
State 1 : IF ev_id_input (DO_Customer_id)
          THEN pr_activate (AIO_Pbt_Search_Customer)
               State = 2
          IF ev_first_last_input (DO_Customer_name)
          THEN pr_activate (AIO_Pbt_Search_Customer)
               State = 3
State 2 : IF ev_id_deleted (DO_Customer-id)
          THEN pr_activate (AIO_Pbt_Search_Customer)
               State = 1
          IF ev_pbt_select (AIO_Pbt_Search_Customer)
          THEN current_id = pr_get_id (DO_Customer_id)
               IF Validate_Cust_id (current_id) = TRUE
               THEN State = 4
                    Generate (ev_Valid_Customer)
               ELSE Display_message ("Invalid Customer Identification")
State 3:  IF ev_first_last_deleted (DO_Customer_name)
          THEN pr_desactivate (AIO_Pbt_Search_Customer)
               State = 1
          IF ev_pbt_select (AIO_Pbt_Search_Customer)
          THEN current_firstname = pr_get_firstname (DO_Customer_name)
               current_lastname = pr_get_lastname (DO_Customer_name)
               IF Validate_Cust_fl(current_firstname,current_lastname)= TRUE
               THEN State = 4
                    Generate (ev_Valid_Customer)
               ELSE Display_message
                    ("Invalid Customer firstname and lastname")
```

Figura 19 "Modelo del diálogo mediante reglas"

En la Figura 19 se muestra la misma información que en el diagrama de la Figura 18 , con la salvedad de que en esta ocasión se expresa a través de sentencias "if-then" que determinan en que estado se encuentra y dentro de las sentencias a ejecutar se cambia al estado siguiente.

Tanto en el modelo por diagramas de estados como en el de reglas, se realizan ante determinado evento acciones que habilitan o deshabilitan los botones dependiendo del estado al que se accede.

Finalmente TRIDENT utiliza una serie de reglas tomadas de muy diversas referencias para poder presentar, mediante una interfaz gráfica (Ver Figura 20) a manera de árbol de decisiones, los objetos de interacción posibles para cada elemento que haya que presentar y permitir al diseñador elegir de entre las alternativas en cada caso como se muestra en el ejemplo.

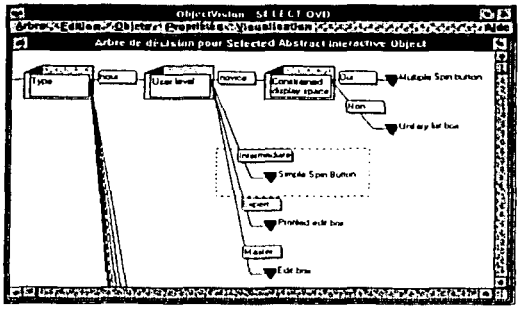


Figura 20 "Interfaz de TRIDENT"

En conclusión, TRIDENT aporta importantes avances en la traducción del modelo de la tarea en un modelo de la interfaz. Aún así, TRIDENT tiene tareas pendientes como la producción de prototipos automáticamente con base en el modelo de la interfaz que se tiene. Además de ello es importante la omisión que se hace de la tarea "antes" de tenerse el sistema, tarea de fundamental importancia para el diseño de la tarea que se hará mediante el sistema.

4.2.2.3 MASTERMIND

Otro proyecto digno de mencionar es MASTERMIND[25]. MASTERMIND trabaja con base en una arquitectura en la que se tiene un modelo servidor que, a través de una capa de comunicaciones CORBA mantiene a los distintos modelos actualizados y evita la inconsistencia de los datos.

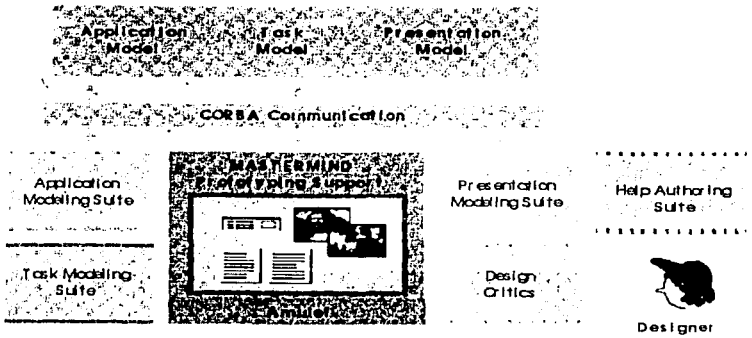


Figura 21 "Arquitectura externa de MASTERMIND"

Este proyecto basa su trabajo en tres modelos, el modelo de la aplicación, el de la tarea y el de presentación. (Figura 9).

El modelo de la aplicación es la forma en que se especifica el funcionamiento de la aplicación en un lenguaje orientado a objetos y basado en CORBA IDL. Este lenguaje fue ampliado únicamente con 2 extensiones muy útiles para el caso particular de este modelo. La primera extensión permite tener precondiciones para la realización de un método, es evidente la necesidad de esta característica si consideramos la naturaleza de los tareas condicionadas. Por otro lado, se tiene la extensión que permite el aviso y consecuente actualización de los datos que sean cambiados en el modelo servidor a fin de que los modelos cliente se percaten del movimiento.

El modelo de la tarea, por otro lado, se basa en una estructura, igualmente jerárquica, pero que describe la secuencia entre las subtareas como una propiedad de la misma tarea. El atributo Task_Connection es el que se encarga de definir esto y, a su vez, tiene algunos atributos para ello. El primer atributo de Task_Connection es la serie de subtareas de la tarea correspondiente. El segundo atributo define el orden mediante el cual éstas serán ejecutadas; ejemplos de valores posibles son secuencia, paralelo, irrestricto, one_of, etc. Estos valores permiten definir, casi por completo el orden en que las tareas deberán ejecutarse en el tiempo de corrida; sin embargo, existen además banderas en el modelo de la tarea que indican propiedades como si la tarea puede ser interrumpida o no, si se puede regresar a ella o no después de la interrupción, si es obligatoria o no, etc.

Finalmente, el modelo de presentación permite modelar los elementos de la interfaz de usuario. Para ello se tiene una descripción muy similar al caso del modelo de la tarea; al igual que ahí, se tiene una serie de datos del elemento dentro de los cuales algunos son más complejos y son, en si, otro elemento con una serie de características. Tal es el caso del atributo "parts", que tiene en el una serie de elementos que también tendrán su propia descripción, es decir, el atributo "parts", del objeto presentación es un arreglo de objetos presentación.

En MASTERMIND se hace evidente la gran mejoría en cuanto a la expresividad de los modelos propuestos, sin embargo, aún hay aspectos por modelar como podría ser las distintas formas en que una tarea puede reiniciarse una vez que se regresa de una interrupción o el manejo de distintos usuarios para distintas tareas. En general, MASTERMIND maneja un concepto mucho más acercado a lo que se pretende en el presente trabajo.

4.2.2.4 ADEPT

ADEPT (Del ingles: Advanced Design Environment for Prototyping with Task Models) [16] es un trabajo desarrollado a principios de los 90's que propone un marco de trabajo con base en el modelado del usuario, de la tarea y de la interfaz.

El proyecto ADEPT se basa en el modelado jerárquico de la tarea, la traducción de éste modelado a un lenguaje formal, un modelo de la interfaz de usuario y las reglas para que estos modelos se mantengan consistentes entre ellos.

El modelado de la tarea se realiza mediante una propuesta en la que cada tarea se modela con un nodo de un árbol jerárquico. El modelado de la secuencia que las tareas deben seguir se realiza mediante distintas notaciones en las líneas que unen a las tareas entre sí. Por ejemplo, si de una serie de tareas, existe una que debe hacerse antes que el resto, ésta se marca mediante una flecha que entra a la tarea proveniente de la tarea padre. Por otro lado si una tarea debe ser antecesora de otra, esto se denota mediante una flecha de la tarea antecesora a la sucesora. (Figura 22)

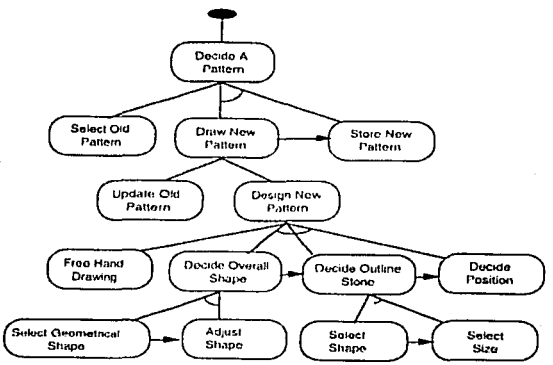


Figura 22 "Modelo de la tarea en ADEPT"

El modelo obtenido en el árbol jerárquico puede ser formalizado mediante un Proceso de Comunicación Secuencial (CSP del ingles; Communicating Secuential Process) donde cada una de las notaciones anteriores tiene un equivalente directo.

Una vez que se tiene este modelo, es traducido a un modelo de interfaz que se le presenta al diseñador. Éste podrá hacer cambios sobre la interfaz que deberán ser agregados al modelo de la interfaz, pero que no serán agregados al modelo de la tarea. Ejemplos de esto son aquellos elementos del diálogo de la interfaz que no pertenecen a la tarea en sí como podrían ser los botones de aceptar y cancelar. Aún así, el modelo formal si reflejará los cambios.

El problema con ADEPT es la poca información que se tiene sobre la tarea, es decir, si bien se expresa con claridad la división de una tarea en subtareas y se muestra la secuencia básica entre las mismas, no se tiene información que nos anuncie comportamientos de la misma en situaciones como las interrupciones, no nos otorga criterios (como prioridades) para decidir que tarea iniciará en caso de no especificarlo el orden normal. Tampoco nos permite tener elementos para asignar tareas a los diferentes miembros de un equipo de trabajo a fin de poder paralelizar realmente las labores, aunque sí prevé este posible comportamiento.

4.2.2.5 ALACIE

ALACIE (del frances Atelier Logiciel d'Aide á la Concepción d'Interfaces Ergonomiques) es un proyecto desarrollado en Francia [10]y que está basado en un modelo de la tarea llamado MAD* y un modelo que permite la traducción del modelo anterior a una interfaz de usuario, este segundo modelo es llamado SSI (del frances Spécification Sémantique de l'Interface). ALACIE es el conjunto de dos herramientas que implementan los modelos mencionados, IMAD* (Implementación de MAD*) e ISSI (Implementación de SSI).

MAD* es un modelo jerárquico para representar la tarea del usuario explicado en la sección 3.2 del presente trabajo.

SSI, por otro lado, permite modelar prototipos de interfaz a un alto nivel de abstracción (en términos de secuencia, estados y sincronización de tareas) basados en un modelo MAD*. SSI tiene cuatro características principales:

Establece una arquitectura con dos niveles de abstracción: el nivel tarea, que especifica a la misma así como a sus objetivos, y el nivel semántico, que especifica la secuencia de sub-tareas necesarias para cumplir con la tarea especificada.

La definición de equivalencias entre los elementos del modelo SSI y aquellos del modelo MAD*.

La incorporación de los términos, principios y sintaxis de MAD* con el fin de tener un marco de desarrollo homogéneo.

La definición de correspondencias entre los objetos abstractos y los objetos concretos de las interfaces de usuario. Esto permitirá que, a partir de una misma definición abstracta de la interfaz, el analista pueda generar varios prototipos distintos.

Para realizar el modelo de la interfaz, SSI hace uso de tres tipos de objetos: los EP (Esquemas de procedimientos), los P (Procedimientos), los OF (Objetos Funcionales).

Los primeros dos elementos (EP y P) son dos distintas particularizaciones de un tipo de objeto llamado "SSI abstracto" y permiten implementar los niveles de abstracción del modelo; los Procedimientos permiten especificar el nivel tarea y los esquemas de procedimiento implementan el nivel semántica. La información para estos elementos provienen directamente de las tareas del árbol MAD*.

Los objetos funcionales permiten modelar los objetos encontrados a partir de las pre y post condiciones de las tareas del árbol MAD*, aspecto similar al aportado en el proyecto DIANE.

La implementación de IMAD* junto con ISSI permiten la generación de diversos prototipos de interfaces de usuario de manera rápida y basadas en la descripción de la tarea del usuario.

Esta implementación, sin embargo, no resuelve el problema de la captura de requerimientos en los procesos de desarrollo de software. Esto es debido a que las interfaces generadas no se encuentran en un lenguaje formal y por lo mismo no pueden ser tomadas por los procesos de desarrollo actuales como especificación de requerimientos.

4.2.2.6 ConcurTaskTrees Environment (CTTE)

CTEE [28] es una herramienta gráfica para el análisis de modelos de la tarea.

Esta herramienta también cuenta con un modelo de la tarea llamado *ConcurTaskTrees* con bastante expresividad ya que permite especificar aspectos como el tiempo aproximado de duración de una tarea o la frecuencia con que se realiza o el usuario que la realiza aunque carece de otros aspectos como la información sobre la interruptibilidad. El modelo, basado en la notación LOTOS [30], tiene además objetos de interfaz y objetos del dominio del problema que se asocian a las tareas. Finalmente, el modelo también cuenta con una notación gráfica basada en iconos que permiten especificar aspectos como la interactividad de la tarea.

La herramienta, por otro lado, tiene funciones que permiten editar los árboles de la tarea creando o borrando tareas fácilmente. Otro aspecto importante de la misma es que guarda su salida en formato XML con una DTD desarrollada para los modelos especificados en ConcurTaskTrees. Esta salida es de fundamental importancia ya que al apegarse a estándares permitirá un fácil acoplamiento con otras herramientas. Finalmente, el aspecto más sobresaliente de la herramienta es la posibilidad que ofrece de analizar a fondo un

modelo de la tarea mediante el simulador que provee. Esto, sin duda, es una enorme ayuda para los analistas.

Esta herramienta, sin embargo, solo permite un análisis exhaustivo del modelo de la tarea ya que su salida no genera elementos UML que pudiesen ser tomados como base en un proceso de desarrollo. Este aspecto está considerado dentro de los aspectos a trabajar en un futuro por los autores de CTTE.

4.3 Discusión acerca de las herramientas

Para poder determinar hacia que rumbo se necesita trabajar en el desarrollo de herramientas como las tratadas en este capítulo, es necesario hacer una recapitulación de la estructura general que tienen estas herramientas, así como las características que tienen en común, o aquellas que las distinguen. De esta manera será posible identificar que características faltan por desarrollar y se podrá encontrar el cúmulo de condiciones que deberá cumplir una nueva herramienta encaminada a mejorar la situación actual.

Para comenzar este análisis consideraremos que los proyectos mencionados tienen, en general, dos aspectos a ser analizados: Por un lado examinaremos que aspectos de la tarea se modelan en cada uno de los proyectos así como la forma en que traduce entre ellos. Por otro lado, se considerará que aspectos de la tarea son representados en el modelo la tarea utilizado, de modo que podamos medir la expresividad del mismo.

Los modelos que se consideran en los proyectos presentados se mencionan en la primera columna de la Tabla 1; en el renglón superior se muestran los proyectos considerados y las casillas de intersección tendrán un asterisco en los casos en los que el modelo sea considerado en el proyecto.

Modelos\Proyectos	Adept	Mastermind	Diane	Trident	Alacie	CTTE
Modelo de la tarea del usuario					*	
Modelo de la tarea modificado	*	*	*	*	*	*
Modelo formal de la tarea del usuario	*					*
Modelo de la interfaz de la aplicación	*	*		*	*	
Modelo de la aplicación			*	*	*	
Modelo del diálogo de la aplicación			*	*		

Tabla 1 "Comparativa de modelos contemplados en distintos proyectos"

Como se puede observar, hay un aspecto notable: de todas las herramientas, sólo Alacie toma en cuenta el modelo de la tarea original; es decir, todos estos proyectos comienzan a modelar la tarea como se desea que se realice con el sistema. Ya hemos descrito la importancia que tiene el estudio y modelado de la tarea tal como se realiza antes de que el sistema aparezca, por ello este punto es de gran valor.

Por otro lado, dado que son herramientas basadas en modelos de la tarea, es también necesario conocer las características de éstos modelos. A continuación se muestra en la Tabla 2, en ella se muestra la expresividad de los modelos de la tarea empleados; en la columna izquierda se presentan las características de una tarea y en el renglón superior los distintos proyectos analizados.

Característica Proyectos	Adept	Mastermind	Diane	Trident	Alacie	CTEE
División de tareas	*	*	*	*	*	*
Interruptibilidad		*			*	
Prioridades		*			*	*
Roles		*	*		*	*
Pre y post Condiciones		*			*	Solo Pre

Tabla 2 "Comparativa de expresividad del modelo de la tarea en distintos proyecto2"

En esta tabla también tenemos un aspecto de importancia, solo Mastermind y Alacie logran expresar de manera completa la tarea.

Para el caso de esta tesis, tomaremos el modelo MAD* debido a los dos aspectos ya destacados: por un lado la expresividad de éste modelo junto con el que utiliza Mastermind es mayor a la del resto de los modelos en aspectos como la interruptibilidad así como en las pre y post condiciones. Por otro lado, Alacie, tiene una ventaja sobre el resto de las herramientas y es el hecho de que se parte del árbol de la tarea del usuario tal como la realizaba antes de pensarse siquiera en una herramienta que lo asistiera, el resto de las herramientas parten de un árbol de la tarea donde ya se considera la tarea utilizando el sistema que se va a construir. Finalmente, modelos como el de la interfaz y el del diálogo, que son ventajas de herramientas como TRIDENT sobre el resto, no serán útiles para el trabajo de esta tesis ya que no buscaremos la producción de interfaces de usuario.

Esto último es de fundamental importancia en la presente tesis: como ya observamos, existen gran variedad de herramientas que permiten generar prototipos. Mas ejemplos de ellas son esfuerzos como TADEUS [32], HUMANOID [27]; sin embargo, todas estas herramientas permiten la generación de interfaces de usuario o bien únicamente el análisis exhaustivo del modelo, como en el caso de CTTE. Ni las interfaces de usuario ni el modelo de la tarea son un documento suficientemente formal para ser aceptados en los procesos de desarrollo como especificaciones de requerimientos. Como ya se mencionó, lo que se requiere es obtener, a partir del análisis de la tarea, un elemento que ya sea considerado en los procesos de desarrollo ya que solo así se tiene la oportunidad de ser aceptado por los desarrolladores de software actual. En este trabajo se intentará obtener casos de uso, y con ellos integrar los resultados al proceso unificado.

4.4 Conclusión

En este capítulo se han descrito las ventajas que presentan las herramientas actuales en la ingeniería de software. Se ha analizado la manera en que el desarrollo de software ha evolucionado desde el código en lenguaje ensamblador hasta los primeros intentos de herramientas "basadas en modelos". Evidentemente, la calidad de los productos de software se ha venido incrementando enormemente a través de los años y, sin duda, los progresos han incluido áreas como la usabilidad del software.

Por otro lado, en el presente capítulo hemos descrito también las deficiencias que persisten en el área de la captura de requisitos como la falta de consideración al usuario final y la falta de modelos que permitan analizar la tarea que realiza este usuario y deducir sus necesidades a partir de ella. Basándonos en las deficiencias en el campo, se examinaron los esfuerzos realizados en este sentido por la comunidad académica de IHC y se establecieron las ventajas que los proyectos de esta comunidad han aportado para la actividad de la captura de requerimientos.

Siguiendo con el análisis crítico de las herramientas existentes mostramos las desventajas que presentan las herramientas que incorporan modelos de la tarea.

Finalmente, tomando como punto de partida las desventajas que presentan las herramientas actuales, como la limitada expresividad de la tarea y el no partir de la tarea del usuario, se pretende el desarrollo de una nueva herramienta que logre subsanar la mayor parte de las deficiencias por las que aún no se ha logrado tener una captura de requerimientos de calidad en el desarrollo de software actual. En el próximo capítulo describiremos la propuesta de este trabajo que consiste en la realización de una herramienta que permita al analista de la tarea tomar un árbol de la tarea realizado en IMAD*, analizarlo, construir un nuevo árbol de la tarea y obtener los casos de uso del sistema. La herramienta podrá leer el archivo generado por IMAD*, asistir en el análisis del mismo, en la creación de un nuevo árbol de la tarea y generar los casos de uso del futuro sistema a partir del árbol creado por el analista. La herramienta será descrita a detalle desde la manera en que se determinaron los requerimientos que se tienen para la misma, hasta la descripción de la forma en que fue diseñada e implementada.

5 MAD2UML: una propuesta

En el presente Capítulo presentaremos la herramienta que asista al analista de la tarea en su actividad y en la obtención de casos de uso a partir se su análisis. La presentación se arma de la siguiente manera: Inicialmente presentaremos algunas de las posibilidades de traducción que MAD* presenta hacia los casos de uso UML, esto debido a que nuestra propuesta se basa en ayudar al análisis durante la obtención de casos de uso a partir del análisis de la tarea. Enseguida se presenta el análisis que se realizó para definir los requerimientos funcionales y no funcionales que tendría la herramienta.

Como parte de este análisis se estudió el proceso que se pretende asistir y que permitirá el paso desde el análisis de la tarea hasta los casos de uso UML. Una vez habiendo terminado el análisis, se procede a describir el diseño de la herramienta, es decir, cual es la arquitectura de la misma.

Finalmente, se muestran algunas tecnologías que fueron empleadas para la implementación de la herramienta.

*5.1 Posibilidades de traducción MAD**

Como se ha visto en los capítulos anteriores, la intención del presente trabajo es ayudar, mediante una herramienta, a la integración del análisis de la tarea para mejorar la captura de requerimientos dentro del Proceso Unificado.

Como se ha visto también, el lenguaje utilizado por el proceso unificado es UML y por ello es fácil deducir que la herramienta que logre asistir a esta integración deberá entregar como salida elementos expresados en este lenguaje.

Finalmente, requerimos definir que tipo de elementos debemos obtener a fin de que sirvan como punto de partida para la captura de requerimientos dentro del proceso unificado. UML tiene, dentro de sus múltiples elementos, uno en especial llamado Caso de uso. Este elemento expresa las funciones que tiene el sistema y puede ser organizado de manera jerárquica, ya que un caso de uso puede contener subcasos de uso.

Después de analizar las características de un diagrama de casos de uso podemos encontrar una relación estrecha con lo que expresa un árbol MAD*, aunque debemos reconocer que mucha de la información que contiene un árbol MAD* se pierde en el diagrama de casos de uso. Aún así, la relación entre el árbol MAD* y los casos de uso en UML es la más cercana dentro de lo que ofrecen UML y MAD*.

Para analizar esta relación podemos referirnos a la tesis de. En este trabajo, se analizan cuatro distintas formas para obtener los casos de uso a partir del árbol de la tarea modificado. Estas formas para realizar el mapeo se basan en la identificación de cuatro

distintos criterios: Los objetivos del usuario, los objetos involucrados, las acciones realizadas y las tareas que conforman el árbol.

1. Método por objetivos: El primero de estos métodos de mapeo consiste en la identificación de los objetivos del usuario cuando realiza su tarea. Este método busca que el analista de la tarea determine, con base en su propio criterio, cuales son los objetivos fundamentales de la tarea del usuario.

   Una vez determinados los objetivos del usuario, el siguiente paso del proceso es asignar alguno de estos objetivos a cada una de las tareas del árbol MAD*, de esta manera se obtiene una gráfica donde los objetivos son nodos padres, y las tareas que a él se relacionan son los nodos hijos. Existen casos en que una tarea no es ligada al objetivo que le corresponde sino a la su tarea padre en el árbol MAD*. Esta variante al método evita perder la estructura jerárquica, aún así, decidir si aplicar el método original o la variante queda completamente al criterio del analista.

2. Método por objetos: El método de los objetos involucrados consiste en determinar, para cada tarea, cuál es el objeto con el que se tiene más relación. Una vez habiendo analizado todas las tareas del árbol y habiendo decidido cuál es el objeto ligado con cada tarea, se grafica con los objetos como nodos padres y las tareas como nodos hijos. Al igual que en el método anterior, se deja al criterio del analista el ligar tareas a su tarea ascendente en vez de a un objeto.

   Este método tiene como principal ventaja el hecho de que, si se pretende utilizar el paradigma de programación orientada a objetos, los casos de uso estarán agrupados por objetos. Además de ello, se tiene la experiencia de que agrupaciones de este tipo son muy usadas en los programas actuales; es común encontrar menús como Archivo que tiene como submenús todas las acciones que sobre él se realizan como abrir, cerrar, imprimir, renombrar, etc.

3. Método de las acciones: En este método, al igual que en los métodos anteriores, el objetivo es determinar cuales son las principales acciones de la tarea para, a partir de ahí, se relacione a todas las tareas con alguna acción o con su tarea predecesora como en los métodos antes explicados. El método de las acciones es el que menos éxito mostró en el caso del trabajo citado, aún así no se descarta que sea el adecuado en otros casos si consideramos menús tan comunes como Insertar que se encuentran en los principales procesadores de texto.

4. Método de las tareas: El cuarto método es un poco distinto a los tres primeros: en los primeros tres casos se identificaban determinados elementos y se ligaban a ellos cada una de las tareas. En el caso del método de las tareas, el mapeo es uno a uno; es decir, por cada tarea que se tiene en el árbol se crea un caso de uso. Por cada subtarea de la tarea se crea un caso de uso hijo o subcaso de uso.

Analizando la manera de traducir un árbol de la tarea en casos de uso, podemos observar que en todos los casos se traslada la estructura jerárquica del árbol de la tarea, a un grupo

de casos de uso, los cuales están relacionados entre sí por alguna característica que es de interés al analista.

5.2 El proceso de traducción

Para la realización de la presente herramienta se realizó el análisis de la tarea de un analista de tareas que pretende obtener casos de uso. Es decir, se analizó la manera en que un analista de tareas realiza este análisis para averiguar cuales son los requerimientos de la herramienta que lo asista.

Durante este análisis se observó que el analista de tareas tiene como primera actividad la traducción de la tarea del usuario a una tarea modificada; es decir, la tarea que modela la nueva actividad del usuario con el futuro sistema.

En este punto es importante destacar que existen marcadas diferencias entre las dos tareas mencionadas hace un momento: La primera de éstas diferencias se finca en la razón de ser del sistema; es decir, existen tareas que se realizaban en la tarea original y que ya no se realizarán cuando se use el nuevo sistema, ya sea por que se automatizarán o por que dejarán de ser necesarias. Evidentemente, esto implica tareas que aparecerán y desaparecerán del árbol de la tarea final.

Otro aspecto que marca diferencias es el hecho de que los sistemas obligan a la realización de nuevas tareas: ejemplo de esto son las tareas de aceptar, cancelar o validar las operaciones.

Existen tareas, por otro lado, que se siguen realizando pero que no forman parte del sistema ya que son realizadas manualmente, estas tareas también desaparecen del árbol de la tarea final, puesto que sólo interesan las tareas con el sistema.

Lo fundamental durante estas modificaciones es reconocer que la parte medular del árbol (los objetivos del usuario) debe mantenerse, y esta afirmación se basa en el hecho de que el nuevo sistema buscará obtener lo mismo que se lograba antes, pero de manera más eficiente.

Cuando hablamos de los objetivos del usuario, nos referimos a aquellos objetivos que motivan la ejecución de toda la tarea, es decir, si bien cada una de las subtareas tiene un objetivo particular, la gran mayoría de las subtareas buscan reunir los elementos necesarios para lograr uno o varios objetivos generales de toda la tarea. Estos objetivos, difíciles de reconocer en ocasiones, son fundamentales debido a que, sin importar que suceda en la transformación del árbol, la tarea final deberá, necesariamente, cumplirlos cabalmente.

Para formar el árbol de la tarea modificada, el analista requiere realizar varias actividades:

- El analista necesita poder utilizar las tareas o grupos de tareas del árbol de la tarea original en el nuevo árbol.

- El analista requiere crear o eliminar tareas en el nuevo árbol.
- El analista requiere ligar o desligar tareas en el nuevo árbol para reorganizar las tareas en el mismo.

Sin embargo, para poder tomar las decisiones de que tareas crear, eliminar, copiar o ligar en el nuevo árbol, el analista requiere de mucha información. La información que requiere se encuentra en los atributos del árbol MAD* de la tarea original por lo que el analista necesita un medio para revisar estos atributos eficientemente.

Partiendo del análisis anterior definiremos, en la siguiente sección, los requerimientos para la herramienta que se propone en este trabajo.

5.3 Análisis de requerimientos para la herramienta

Una vez analizado el proceso que se desea apoyar con la herramienta, y sabiendo que lo que se quiere tomar como punto de partida es un árbol de la tarea MAD*, es posible continuar con el desarrollo de la herramienta definiendo los requerimientos funcionales y no funcionales de la misma, para después pasar al diseño y la implementación de la misma.

5.3.1 Requerimientos Funcionales

Para el modelado de la tarea modificada con base en la tarea del usuario, el analista necesita conocer varios tipos de información y poder realizar ciertas acciones. Éstas se describen a continuación:

1. Información sobre la tarea

Se requiere saber los siguientes aspectos de las tareas:

- Pre y Post condiciones (expresadas en términos de objetos)
- Objetivos
- Trabajador que la realiza
- Prioridad
- Interruptibilidad
- Modo
- Constructor (Secuencia de realización)

Es, por lo tanto, necesario que la herramienta provea al analista de esta información sobre las tareas.

2. Información sobre características comunes entre tareas:

El analista requerirá de asistencia para poder distinguir, en todo momento, grupos de tareas que comparten alguna característica en el árbol original. La herramienta deberá permitir

agrupar las tareas con base en la información antes mencionada y tendrá que facilitar también la manipulación de estos grupos.

3. Funciones de edición de tareas:

Si bien es necesario que el analista investigue el árbol de la tarea del que parte, también es necesario que pueda ir conformando el nuevo árbol con base en la información que la herramienta le provee. De esta forma, la herramienta tendrá como requerimiento la posibilidad de tomar grupos definidos en el primer árbol y llevarlos al nuevo árbol. De hecho, se necesitará también que la herramienta permita crear y borrar tareas en el nuevo árbol en el momento que lo desee.

4. Exportación de casos de uso

Mediante los requerimientos ya expresados, se permitiría al analista realizar el diseño de un nuevo árbol de la tarea modificada. Aún así, la funcionalidad de la herramienta no estará completa si no se logra que el modelo obtenido sea expresado en un diagrama de casos de uso que pueda ser empleado como punto de partida en la captura de requerimientos del Proceso Unificado. Por ello, esto último se convierte en el último requerimiento funcional.

5.3.2 Requerimientos No funcionales

Si bien el requerimiento funcional sólo indica que, mediante la herramienta, se deberán obtener los posibles casos de uso del sistema. Es también de destacarse que el objetivo de esta herramienta es permitir la integración del análisis de la tarea al proceso de desarrollo de software. Se ha comentado también en esta Tesis que la herramienta líder en la documentación de procesos de desarrollo de software es Rational Rose®. Esto último lleva a plantear como un requerimiento No funcional la obtención de una salida que pueda ser leída en Rational Rose® y que responda a los estándares internacionales actuales. De esta manera, la industria podrá acoger de mejor manera a la herramienta propuesta.

Finalmente, y también como requerimiento no funcional, podemos establecer el hecho de que existe una herramienta que permite la generación de árboles MAD*. Esta herramienta, llamada IMAD* existe actualmente y la herramienta propuesta deberá poder leer los archivos generados por ella. De esta manera, el archivo de origen para la herramienta queda definido también como un requerimiento no funcional.

Con los requerimientos ya mencionados, el siguiente paso es el diseño de la herramienta y a la descripción de este diseño se avocarán los siguientes subtemas.

5.4 Diseño de la herramienta

Enseguida se describirán los bloques de los que consta el diseño de la herramienta, así como la manera en que se estructuran los datos que maneja. Se mostrará también a partir

de que información funciona la herramienta y que información es posible generar con su ayuda.

5.4.1 Bloques

La herramienta está definida por 4 bloques principales:

- Entrada y salida de datos
- Manejo de la información
- Manipulación de árboles
- Visualización de datos

Como se puede observar en la Figura 23, todos los bloques de la herramienta se relacionan entre sí, ya que acceden al modelo de datos en donde se guarda toda la información.

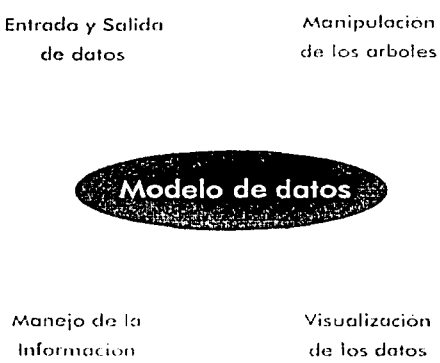


Figura 23 "Diagrama de bloques de la herramienta"

A continuación se describen las funciones que cada uno de ellos deberá proveer a la herramienta. El modelo de datos es explicado posteriormente.

5.4.1.1 Entrada y salida de datos

Este bloque permite satisfacer el requerimiento funcional de exportación a casos de uso así como los requerimientos no funcionales que pedían a la herramienta leer los archivos .MAD y exportar a un formato legible en Rational Rose (Ver 5.3.1 y 5.3.2).

Este bloque tiene varias funciones:

1. Por un lado, obtiene los datos del árbol de la tarea del usuario. Estos datos los toma mediante un parseo del archivo .MAD generado por la herramienta IMAD* (ver sección 4.2.2.5 pag. 49). Una vez que el programa lee los datos del archivo, los escribe en un modelo de datos jerárquico que se manejará en el resto del programa y que se detalla más adelante (ver 5.4.2 en pag. 62).

2. Por otro lado, la herramienta lee un segundo archivo generado por IMAD* para obtener los objetos manejados en las pre y post condiciones de las tareas. Gracias a la información obtenida de este archivo, será posible corroborar que las condiciones expresadas en el árbol MAD* estén correctamente expresadas.

3. Este bloque también se ocupa de la generación del archivo del proyecto; es decir, de generar un archivo que plasme el estado actual del proyecto. Esto incluye una referencia al árbol de la tarea original, así como una al "árbol modificado". Este archivo se genera con el fin de que el proyecto pueda ser guardado y abierto más tarde para continuar con él. El *árbol modificado* es escrito en un archivo XML a partir del modelo de datos interno. Como complemento a esta función, el módulo lee el archivo anteriormente mencionado.

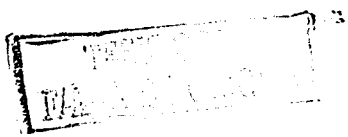
4. Finalmente, el bloque también permite exportar el árbol modificado obtenido en un archivo en el lenguaje XMI, estandar de codificación de UML en XML. Este archivo, además de estar en un lenguaje estandar como XMI, puede ser leído directamente por Rational Rose®. Para esta lectura, rational emplea un plug-in de la compañía Unysis® que le permite leer XMI.

5.4.1.2 Manejo de la información

El bloque de "manejo de la información" permite realizar "búsquedas" sobre el árbol de la tarea; esto es, examinar el conjunto de todas las tareas que se tienen en el árbol y, a partir de ciertos criterios, obtener subconjuntos de tareas de interés para el usuario. Lo anterior satisface el requerimiento de poder tener información sobre las características comunes de las tareas (Ver 5.3.1) y se logra mediante tres funciones particulares.

1. Como ya se mencionó, la herramienta permite obtener un conjunto de tareas con base en una serie de criterios. Estos criterios, a los que en adelante llamaremos "criterios de selección", consisten en restricciones que se establecen sobre los valores que pueden tomar determinadas propiedades de la tarea. Estas propiedades son:

    a. Los objetos en las condiciones
    b. La prioridad
    c. La interruptibilidad
    d. El modo en que se realiza la tarea
    e. El trabajador que la ejecuta.



Los objetos a los que se refiere el punto a) son aquellos mediante los cuales el analista estableció, en el árbol de la tarea del usuario, las condiciones de inicio, de arranque, de paro y las postcondiciones de cada tarea. En la herramienta es posible establecer los objetos que interesan al analista así como en que tipo de condiciones se desea que se busque a los objetos determinados.

El segundo criterio permite al analista determinar si únicamente desea considerar aquellas tareas con prioridades mayores a cierto valor.

El tercer criterio, por otro lado, permite al analista despreciar a aquellas tareas que tengan ciertos tipos de interruptibilidad, es decir, el analista podría desear examinar únicamente aquellas tareas que sea posible interrumpir o viceversa.

El cuarto criterio hace posible decidir si desea considerar a las tareas que tengan un modo de realización en particular (ej. solo las tareas automáticas).

Finalmente en lo referente al quinto criterio, éste permite al analista obtener como resultado sólo aquellas tareas que sean realizadas por determinado trabajador.

Los *criterios de selección* permiten al analista determinar qué características desea que tenga el grupo de tareas obtenido como resultado de la búsqueda. Los valores a los que el analista podrá restringir cada uno de estos criterios son definidos mediante la interfaz de usuario, misma que se presentará más adelante.

2. La segunda función es la de, con base en los criterios descritos, examinar el árbol de la tarea del usuario y obtener el conjunto de tareas que satisfacen los parámetros fijados en la búsqueda. A éste conjunto de tareas lo llamaremos "*tareas resultantes*".

3. Finalmente, el módulo coloca las "tareas resultantes" dentro de una tabla de resultados que el analista puede ver, e interactuar con ella. Así, además de presentar las "tareas resultantes" al analista, el módulo presenta estas tareas de tal manera que puedan ser manipuladas mediante el modulo de "manipulación de árboles".

5.4.1.3 Manipulación de árboles

Los árboles de la tarea están formados por tareas, y por grupos de tareas que forman ramas del árbol. El bloque de "manipulación de árboles" permite, precisamente, manipular tareas o grupos de tareas y crear nuevas tareas en el árbol modificado; todo esto era requerido de acuerdo a lo establecido en el segundo y tercer requerimiento funcional (Ver 5.3.1).

Las funciones del módulo se pueden agrupar tomando en cuenta los objetos que maneja.

1. Se encarga de brindar todas las herramientas para transferir tareas aisladas del árbol de la tarea del usuario al árbol modificado. Permite también crear nuevas tareas en este último y eliminarlas en caso de así desearlo.

2. Por otro lado, éste bloque permite transferir segmentos completos (ramas) del árbol de la tarea de un árbol al otro. Además de ello, permite ligar o desligar tareas dentro del árbol modificado lo que da lugar a la creación y modificación de la estructura jerárquica del nuevo árbol.

5.4.1.4 Visualización de datos

Por último, el bloque de visualización. Este bloque se encarga de mostrar al usuario toda la información del árbol en diversos formatos; esto se necesitaba de acuerdo al primer requerimiento funcional establecido (Ver 5.3.1):

1. Mediante una tabla, se muestran los atributos MAD* de la tarea que se tenga seleccionada.

2. En campos independientes se muestran los atributos más importantes de la tarea; los valores de algunos de ellos pueden ser reestablecidos para el árbol modificado (es el caso del modo en que se realiza la tarea y del constructor de la misma).

3. El bloque provee también de una vista miniatura del árbol que permite manejar el nivel de acercamiento que se desee. De esta manera, el analista tendrá una vista general de la totalidad del árbol, o una vista más detallada de secciones particulares de acuerdo con sus necesidades.

4. Finalmente, se tiene también una función de marcado que permite que una tarea, o un grupo de tareas, sea marcada con un color que el analista decida, a fin de que pueda tener grupos de tareas visualmente bien definidos.

5.4.2 Modelo de datos

En lo referente al modelo que nos permitirá guardar y manipular los datos, es importante mencionar que se parte de un modelo de datos que provee la herramienta jViews (esta herramienta es explicada a detalle en la sección de tecnologías utilizadas (ver seccio 5.6.1 pag 79). Mediante el modelo de datos que provee jViews se pueden guardar elementos, agregar elementos, quitar elementos o modificar la información los elementos que se tengan guardados (Los elementos en el modelo, en nuestro caso particular, son las tareas, las ligas, etc). También se pueden copiar elementos o grupos de elementos al portapapeles, así como pegar los elementos del portapapeles al modelo. Mediante estas últimas dos operaciones, se hace posible la transferencia de elementos entre distintas instancias del modelo gracias a lo cual se logró la implementación del módulo de "manipulación de árboles".

El modelo de datos consta de elementos que tienen propiedades y estas propiedades, tienen valores. También permite borrar o agregar elementos, y a éstos, agregarles o quitarles propiedades. Por supuesto, también es posible modificar los valores de sus propiedades.

Para la herramienta propuesta en esta Tesis, se utilizaron dos instancias del modelo; es decir, se utilizó el modelo provisto por jViews para crear dos casos particulares del mismo (instancias), que se ajustaran a las necesidades de nuestra herramienta. En la primera, se crearon dos tipos de elementos: **las tareas** y **las ligas** entre tareas. Las primeras tienen como propiedades todos los atributos de una tarea MAD*. Las segundas tienen como propiedades la tarea origen y la tarea destino. Es utilizando las funciones que provee esta instancia del modelo que se implementa todo lo referente al bloque de "manipulación de los árboles".

Por otro lado, se tiene una instancia del modelo que guarda, únicamente, todos los objetos utilizados en las pre-post condiciones de las tareas. Como ya se comentó, la herramienta lee la información de los objetos manejados en las condiciones de las tareas del árbol así como de sus propiedades. La herramienta utiliza la información de este segundo archivo para constatar que todas las pre y post condiciones del árbol MAD* estén expresadas en términos de estos objetos. Es utilizando las funciones que provee esta segunda instancia del modelo, que es implementada la funcionalidad del bloque de "manejo de la información".

Cabe recordar que tanto el árbol de la tarea como los objetos que utiliza son definidos en la herramienta IMAD*, por lo que para mayor información recomendamos al lector recurrir a [10]

5.5 Implementación de la herramienta

Al contar con las funciones que proveen los bloques definidos en este capítulo ("Entrada y salida de datos", "Manipulación de árboles", "Manejo de la información" y "Visualización de datos"), se cumple con los requerimientos funcionales de la herramienta; es decir, con el "¿qué?" debe hacer la herramienta. A continuación veremos los aspectos que se refieren al "¿cómo?" y "¿cuándo?" lo hará, es decir a la implementación de la misma. Primero, se muestra la interfaz de usuario sección por sección. Al tiempo que se hace esto, se describe la manera en que se implementó la funcionalidad correspondiente a la sección. Enseguida, se describen los archivos de donde la herramienta toma la información y finalmente los archivos en donde escribe la información que genera.

5.5.1 Interfaz de usuario y su implementación

La interfaz de usuario utiliza las funciones de los bloques antes mencionados y las hace accesibles para el usuario. Por lo anterior, durante la explicación de las distintas secciones de la interfaz, se hará referencia a los bloques con los que cada una de estas secciones se relaciona.

La Figura 24 muestra la primera pantalla de la herramienta. En ella podemos apreciar varias *secciones* encerradas en rectángulos:

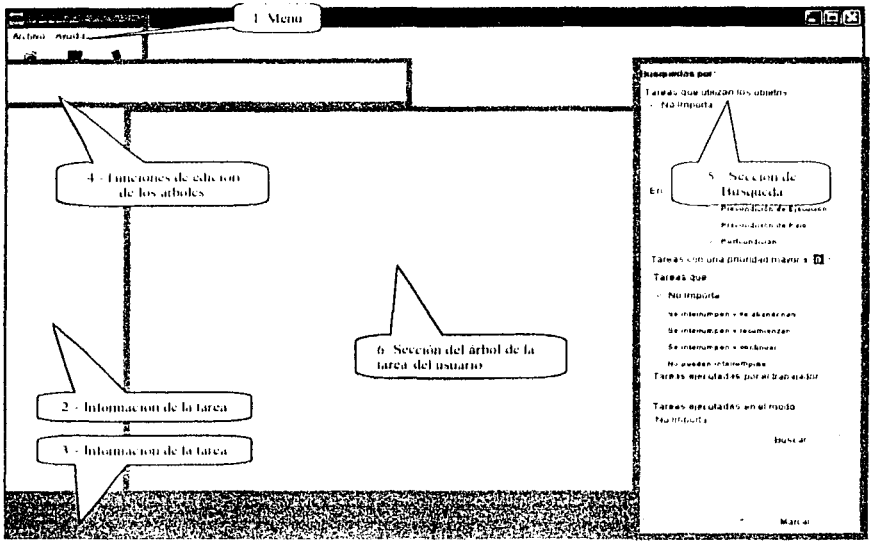


Figura 24 "Pantalla de inicio"

5.5.1.1 Menú

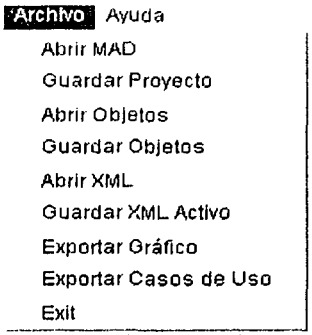


Figura 25 "Menú 'Archivo'"

El menú 'Archivo' (Figura 25) da acceso a tres funciones principales: Primero, "Abrir MAD" permite abrir un archivo generado por la herramienta IMAD* para dar inicio a un

nuevo proyecto. Segundo, "Guardar Proyecto" permite guardar el trabajo realizado para continuar editándolo en otro momento, esta función permite al usuario grabar los arboles de tareas original y modificado así como el arbol de objetos de tal manera que al querer abrir el proyecto nuevamente se tengan todos los elementos para continuar. Es importante destacar que los arboles mencionados se guardarán en ese orden y para cada uno se pide un nombre por lo que se recomienda darles nombres descriptivos. Por otro lado, para leer el proyecto guardado es necesario leer cada uno de los arboles guardados previamente: Mediante el menú "Abrir Objetos" se leen los objetos para las búsquedas, mediante "Abrir XML se lee un arbol y se coloca en el arbol activo por lo que es necesario llamar esta función en el árbol original y en el árbol modificado leyendo los respectivos archivos.

Por otro lado, todos los arboles pueden ser guardados individualmente mediante la s funciones "Guardar objetos" y "Guardar Arbol Activo". Finalmente, "Exportar Casos de Uso" permite la generación del archivo XML de casos de uso que podrá ser abierto desde Rational Rose® y "Exportar Gráfico" permite exportar el gráfico activo en formato SVG[3]. Como se puede observar, las funciones de esta sección de la interfaz de usuario corresponden a las que provee el bloque de "entrada y salida de datos".

La implementación de esta sección se basa en el siguiente proceso:

- Lectura del archivo .MAD
- Parseo del mismo para obtener la información en variables
- Creación de archivo XML con la información obtenida
- Lectura y grabación del archivo XML o SVG mediante el modelo de datos

El proceso anterior podría parecer sencillo pero no es así, la sencillez de la implementación se debe a la modularización del código que se tiene en la aplicación misma que se explica a continuación. En términos generales, la herramienta basa su funcionalidad en tres niveles de código: el código en la interfaz, el código en clases independientes y el código del modelo de datos. En el caso de la funcionalidad aquí explicada, los primeros tres puntos son codificados directamente en la interfaz, el cuarto punto, por otro lado, es parte de la funcionalidad que provee el modelo de datos de jViews.

5.5.1.2 Sección de información sobre la tarea

Para utilizar esta sección es necesario, tener cargado en la herramienta un árbol MAD*, esto puede lograrse con la opción "Abrir MAD" del menú Archivo, o abriendo un proyecto ya existente.

Como se puede apreciar en la Figura 26, al seleccionar una tarea del árbol toda la información de la misma es desplegada en la tabla de la parte izquierda. Por otro lado, sus características más importantes (nombre, objetivo, constructor y el modo en que se ejecuta),

[3] SVG es un lenguaje para el guardado de gráficos vectoriales en XML por lo que es un estandar que la gran mayoría de las herramientas gráficas soporta e incluso puede ser visualizado en un explorador de Internet.

se muestran en la parte inferior para ser vistas con mayor facilidad. Dos de estas características de la tarea (el constructor y el modo), son además editables ya que es posible que estas características de la tarea cambien al realizarse en el nuevo sistema. Las funciones que utiliza esta sección de la interfaz corresponden al bloque de visualización de los datos.

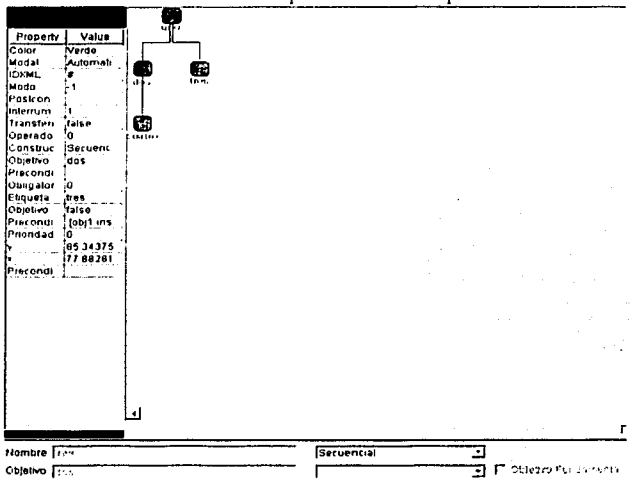


Figura 26 "Seccion de información de la tarea"

La implementación de estas funciones se divide en:

- Inclusión del "inspector" en la pantalla.
- Edición de propiedades de las tareas.

El primero de estos puntos se detalla a continuación: El modelo de datos de jViews, provee de una tabla de visualización de las propiedades del modelo; para desplegar la tabla como lo requería nuestra herramienta se creó una versión modificada de la clase que genera la tabla.

El segundo de estos puntos fue implementado mediante la lectura y escritura de las propiedades de la tarea que se tuviese seleccionada. Un ejemplo de esto se muestra en el Fragmento de Código 1.

```
IlvSDMEngine engine = TheViewer.getEngine();
      Enumeration enum =engine.getSelectedObjects();
      while (enum.hasMoreElements()){
            IlvDefaultSDMNode nodo = (IlvDefaultSDMNode)enum.nextElement();
            nodo.setProperty("Color",jComboBoxColores.getSelectedItem());
      }
```

Fragmento de Código 1

Como se puede observar en el fragmento de ejemplo, en la primera línea se cargan los objetos seleccionados, en la cuarta línea se toma uno de ellos y en la quinta se asigna la propiedad "color" con el valor seleccionado en la lista desplegable o *combo box* respectivo.

5.5.1.3 Sección de funciones de edición

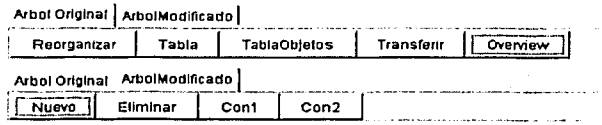


Figura 27 "Barras de herramientas de árboles 'de la tarea del usuario' y 'modificado'"

Las funciones mostradas en la barra de herramientas de la Figura 27, corresponden a las funciones que provee el bloque de "manipulación de árboles" y permite al usuario transferir tareas del "árbol de la tarea del usuario" al "árbol modificado", crear y eliminar tareas de este último, así como ligar y desligar tareas dentro del mismo. Con estas herramientas el usuario puede diseñar su árbol de la tarea modificado.

La implementación de algunas de estas funciones se logró valiéndose de las capacidades del modelo de datos para copiar, pegar, borrar o modificar los elementos que guarda así como para saber por si solo, cual tarea se encuentra seleccionada. En el Fragmento de Código 2 se muestra un caso en el que se toma el elemento seleccionado, se le cambia la propiedad "transferida", se le copia del árbol de la tarea del usuario y se le coloca en el árbol modificado.

```
while (selected.hasMoreElements()){
          IlvDefaultSDMNode nodo=(IlvDefaultSDMNode)selected.nextElement();
          nodo.setProperty("Transferida","true");
        }

        engine.copy();
        engine = viewermod.getEngine();
        engine.paste();
```

Fragmento de Código 2

La función de crear tareas (llamada con el botón "nuevo" del menú superior del árbol modificado), por otro lado, fue implementada creando elementos nuevos para el modelo de datos que contiene al árbol modificado, basados en los datos que el usuario coloca en la pantalla respectiva (ver Figura 28). El código que realiza esto se muestra en el Fragmento de Código 3.

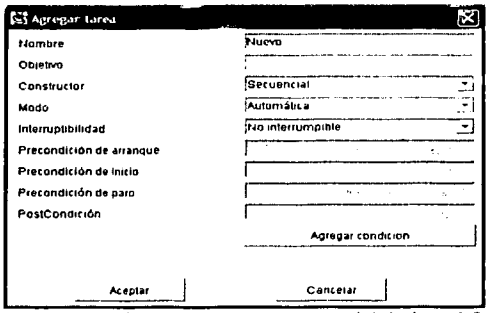


Figura 28 "Pantalla para Agregar tareas al árbol modificado"

```
IlvDefaultSDMNode nodo= (IlvDefaultSDMNode)modelo.createNode("tarea");
modelo.setObjectProperty(nodo,"Etiqueta",jTextFieldNombre.getText());
modelo.setObjectProperty(nodo,"Objetivo",jTextFieldObjetivo.getText());
modelo.setObjectProperty(nodo,"PrecondicionInicio",jTextFieldPrecondicionInicio.getText());
modelo.setObjectProperty(nodo,"PrecondicionArranque",jTextFieldPrecondicionArranque.getText());
modelo.setObjectProperty(nodo,"PrecondicionParo",jTextFieldPrecondicionparo.getText());
modelo.setObjectProperty(nodo,"Postcondicion",jTextFieldPostcondicion.getText());
modelo.setObjectProperty(nodo,"Constructor",jComboBoxConstructor.getSelectedItem());
modelo.setObjectProperty(nodo,"Modo",String.valueOf(jComboBoxModo.getSelectedIndex()));
modelo.setObjectProperty(nodo,"Interrumpibilidad",jComboBoxInterrumpible.getSelectedItem());
modelo.addObject(nodo,null,null);
```

Fragmento de Código 3

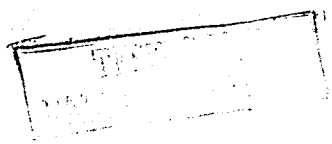
Finalmente, para ligar tareas dentro del árbol modificado se creo una liga cuyas propiedades son la tarea origen y la tarea destino, previamente seleccionadas por el usuario. El código para implementar esto último se muestra a continuación.

```
IlvDefaultSDMLink nodol= (IlvDefaultSDMLink)modelo.createLink("transition");
nodol.setFrom(nodoorigen);
nodol.setID("233");
nodol.setTo(nododestino);
modelo.addObject(nodol,null,null);
```

Fragmento de Código 4

5.5.1.4 Sección de búsqueda

La *sección* de búsqueda (5) contiene las funciones proporcionadas por el bloque de *manejo de la información*. Los métodos que este bloque provee permiten realizar búsquedas en el árbol de tareas en base a los *criterios de selección* (Ver 5.4.1.2) como se observa en la Figura 29.



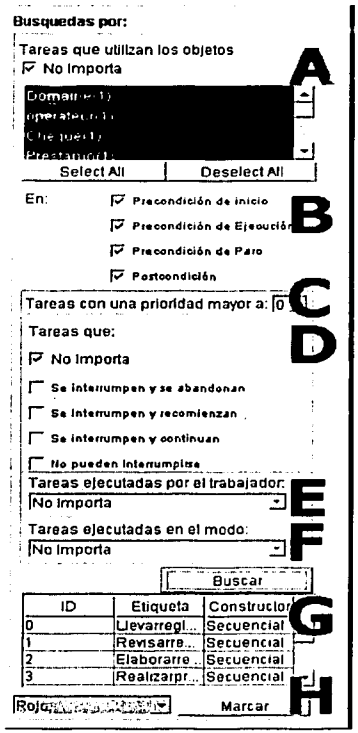


Figura 29 "Sección de búsquedas"

La búsqueda puede ser realizada de acuerdo con los siguientes "criterios de selección":

1. Es posible restringir los objetos involucrados en las condiciones de la tarea. En la lista (a) se muestran todos los objetos involucrados en el árbol de la tarea. La herramienta permite al usuario seleccionar aquellos que le interesen. Por otro lado, en las casillas de verificación (b), se pueden seleccionar el tipo de condición(es) dentro de la(s) cual(es) se desea buscar a los objetos señalados en la lista (a).

2. Otro "criterio de selección" es la prioridad de la tarea. En caso de seleccionar en la lista desplegable (c) una prioridad mayor a cero (cero está establecida por defecto), sólo las tareas con una prioridad mayor a la seleccionada serán incluidas en el conjunto de "tareas resultantes" generado.

3. La manera en que una tarea se puede o no interrumpir es también un "criterio de selección". Para usarlo, simplemente se selecciona la o las casillas (d) de aquellos tipos de interruptibilidad que se desee considerar en las "tareas resultantes". Se

presentan para ello las casillas correspondientes a las cuatro posibilidades que ofrece MAD* en este campo.

4. Quién ejecuta la tarea (es decir, qué trabajador la ejecuta), y el modo en que se realiza, son aspectos que pueden ser limitados también durante la búsqueda. Para ello, sólo se requiere seleccionar el valor deseado para éstas propiedades (e) y (f) dentro de la lista de selección respectiva.

Si bien es posible filtrar los resultados de la búsqueda mediante todos estos criterios, y encontrar con ello tareas de un perfil muy especifico; también es posible realizar búsquedas con pocos o ninguna restricción que filtre sus resultados. Para ello, cada criterio tiene una opción para no considerarlo en la búsqueda. Esta opción está seleccionada por defecto al ingresar a la herramienta por lo que, para el caso en que no se considere ningún criterio explícitamente, las "tareas resultantes" son el total de las tareas del árbol.

Las "tareas resultantes" son desplegadas en la tabla (g) y son seleccionadas en el árbol de la tarea del usuario automáticamente. Esto con el objetivo de que el usuario las identifique y pueda manipularlas según sus necesidades. Finalmente, en esta sección se cuenta con el botón "marcar" (h), mediante el cual, es posible marcar las tareas seleccionadas (ya sea por el sistema o por el usuario) en determinado color. Gracias a esto el usuario puede destacar sus propios grupos de tareas.

Para la implementación de la sección de búsqueda se requirió un algoritmo que revisara todo el árbol de la tarea y validara cada tarea contra los criterios de selección elegidos en la interfaz de usuario de la herramienta. El código de éste algoritmo se muestra en el Fragmento de Código 5.

```
while (tareas.hasMoreElements()){
        tareatmp= (IlvDefaultSDMNode)tareas.nextElement();
        System.out.println(tareatmp.getTag());
        boolean queda=false;
        String temp;
        String otemp;

        if (tareatmp.getTag().equals("tarea")){
        // para las tareas hay que provar

          //si algún objeto se encuentra en alguna condicion
          if(!jCheckBoxObjetosNoImporta.isSelected()){
            for (i=0;i<condiciones.size();i++)
                for (j=0;j<objetos.size();j++)
                    {
                    otemp = (String)objetos.elementAt(j);
                    temp = (String)tareatmp.getProperty((String)condiciones.elementAt(i));
                    if (temp!=null)
                      if (temp.indexOf(otemp)>0)
                        queda=true;
                    }
          }
          else
              queda=true;

          if (Integer.valueOf((String)tareatmp.getProperty("Prioridad")).intValue()>=prioridad & queda)
              queda=true;
          else
              queda=false;

          if (!jCheckBoxNoimportaInterrupcion.isSelected())
            if (interrupciones.contains(tareatmp.getProperty("Interrumpible"))& queda)
              queda=true;
            else
```

```
                queda=false;

            if(jComboBoxTrabajador.getSelectedIndex()!=0)
              if (tareatmp.getProperty("Operadores").equals(trabajador) & queda)
                  queda=true;
              else
                  queda=false;

            if(jComboBoxFiltraModo.getSelectedIndex()!=0)
              if (tareatmp.getProperty("Modal").equals(modo) & queda)
                  queda=true;
              else
                  queda=false;

            if (queda){
                Vector vtemp = new Vector();
                vtemp.addElement((Object)tareatmp.getID());
                for (i=1; i<Columnas.length; i++)
                  vtemp.addElement(tareatmp.getProperty(Columnas[i]));
                vtareas.addElement(vtemp);
                enginetareas.setSelected(tareatmp,true);
            }
          }
          else
            enginetareas.setSelected(tareatmp,true);
        }
```

Fragmento de Código 5

Por otro lado, los resultados obtenidos son mostrados en la tabla de la parte inferior. Las tareas resultantes son seleccionadas y el usuario puede marcarlas en algún color. El código que hace esto se muestra en el Fragmento de Código 6.

```
IlvSDMEngine engine = TheViewer.getEngine();
      Enumeration enum =engine.getSelectedObjects();
      while (enum.hasMoreElements()){
            IlvDefaultSDMNode nodo = (IlvDefaultSDMNode)enum.nextElement();
            if (nodo.getTag().equals("tarea"))
              nodo.setProperty("Color",jComboBoxColores.getSelectedItem());
      }
```

Fragmento de Código 6

Como se puede apreciar, el código únicamente establece una propiedad "Color" a las tareas seleccionadas. Es la aplicación de la hoja de estilos la que hace que visualmente se coloreen las tareas. El fragmento de la hoja de estilos que indica esto último se muestra en el Fragmento de Código 7.

```
node[Color="Rojo"]{
        fillColor1 : "Red";
 }

 node[Color="Azul"]{
        fillColor1 : "Blue";
 }
...
...
```

Fragmento de Código 7

5.5.1.5 Sección de visualización general del árbol de la tarea

En esta última *sección* es posible visualizar el árbol de la tarea y, mediante la ventana de navegación (Figura 30), realizar acercamientos a las secciones que se desee. Esta sección de la interfaz es implementada en el modelo de datos de jViews y la herramienta se limita a

utilizar el método showOverview() del modelo de datos. La tecnología ofrecida por jViews será detallada posteriormente en este trabajo (Ver 5.6.1).

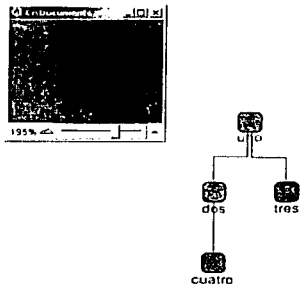


Figura 30 "Vista de Navegación"

5.5.2 Archivos de entrada

Una vez explicadas las distintas funciones del sistema, así como la manera en que el usuario puede acceder a ellas, mostraremos más a detalle los archivos que procesa el bloque de "entrada y salida de datos", así como los que genera.

Como ya se explicó, se tienen tres archivos de entrada en la herramienta, además de uno que puede ser de entrada o de salida (Figura 31). Estos archivos contienen al árbol de la tarea, a los objetos del árbol y a las preferencias de visualización del mismo. Los archivos mencionados son explicados a continuación.

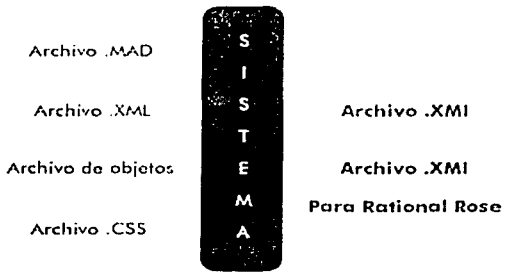


Figura 31 "Archivos de entrada y salida de la herramienta"

5.5.2.1 Archivo del árbol de la tarea

El principal archivo de entrada es el archivo .MAD generado por la herramienta IMAD*. Este archivo contiene las tareas correspondientes al "árbol de la tarea del usuario", y todas sus propiedades, así como las ligas entre las tareas.

El archivo .MAD leido está formado por renglones que dan la información de las tareas o ligas entre tareas del árbol. Como se puede observar en él. Las líneas con la palabra "IlvTache" en el quinto elemento del renglón son tareas y las que llevan la palabra IlvDoubleLinkImage son ligas entre tareas. El resto de la información, tanto de ligas como de tareas está en el resto de los elementos del renglón.

```
0 0 { 0 0 IlvTache 100 130 "uno" "1" "uno" …
0 0 { 1 0 IlvTache 150 230 "dos" "2" "dos"…
0 0 { 2 0 IlvTache 150 330 "tres" "3" "tres  …
0 0 { 3 0 IlvTache 150 430 "cuatro" "4" "dos   …
3 1 { 4 0 IlvDoubleLinkImage 1 0 1 } 24
3 1 { 5 0 IlvDoubleLinkImage 1 0 2 } 24
3 1 { 6 0 IlvDoubleLinkImage 1 1 3 } 24
```

Fragmento de Código 8

La herramienta lee el archivo .MAD y lo convierte al formato XML que comprende el modelo de datos.

El formato del archivo XML es el siguiente:

```
Después del encabezado se abre un gran elemento <SDM> y enseguida se tienen nodos con dos formatos
diferentes. El primero de ellos es el que representa las tareas:

<tarea ID="valorID">
<property name="PROP1">valor1</property>
<property name="PROP2">valor2</property>
   …                                    …
<property name="PROPn">valorn</property>
</tarea>
```

Las propiedades intrínsecas al modelo son las que se tienen dentro de la etiqueta, es decir: tarea, que será el valor de la propiedad *TAG*, e *ID* cuyo valor es *ValorID*.

El resto de las propiedades, es decir *PROP1*, al *PROPn*, son las propiedades que se agregan al nodo y es mediante ellas que se establecen todas las propiedades del árbol MAD.

En el caso de las ligas se tiene el mismo formato; sin embargo, a diferencia de los nodos, en este caso sólo se utilizan propiedades intrínsecas como lo son el origen y el destino de la liga. El elemento XML queda de la siguiente manera:

```
<transition Id="ValorID" islink="true/false" from="IDNodoOrigen" to="IDNodoDestino" />
```

Aquí, *transition* es el valor de la etiqueta *TAG* del elemento, *ValorID* es el valor de la propiedad *ID*, la propiedad *islink* indica si se trata o no de una liga, y las propiedades *FROM* y *TO* deberán tener establecidos los valores de la propiedad *ID* de los respectivos nodos.

Finalmente se cierra la etiqueta <SDM> del archivo XML con una etiqueta </SDM> que cierra el archivo.

Mediante este archivo XML el modelo de datos puede guardar y volver a leer la información a disco duro. Para ello el modelo cuenta con métodos tan sencillos como OpenXML o SaveXML.

5.5.2.2 Archivo de objetos

El segundo archivo de entrada que utiliza la herramienta es el archivo de objetos que también genera IMAD*. Este archivo contiene la información de los objetos usados en las condiciones del árbol de la tarea y también es transformado al formato de XML, aunque únicamente para ser leído. Esto se debe a que la herramienta no permite la modificación de los objetos del árbol y, por tanto, guardar nuevamente este archivo pierde sentido.

El formato de éste archivo es el siguiente:

Por cada clase se escribe un renglón en el archivo XML. Cada renglón tiene los siguientes elementos, divididos por un espacio:

Nombre de la clase	El nombre asignado a la clase
Numero de atributos	El número de atributos que tendrá esta clase
Numero de instancias	El número de objetos que serán creados a partir de esta clase.
Es único o no?	Determina si únicamente existe un objeto derivado de la clase o son varios.
Es del sistema o no?	Determina si la clase es del sistema (no creada por el usuario)
Esta siendo usado o no?	Determina si la clase es utilizada dentro de las condiciones de las tareas del árbol de la tarea del usuario.

Después de cada uno de los renglones de las clases se escribe un renglón por cada uno de los atributos de la clase. Cada una de estos atributos tiene las siguientes propiedades también separadas por un espacio.

Nombre	El nombre del atributo de la clase
Tipo	El tipo de atributo, (String, Numérico, Boolean o Time)
Es del sistema o no?	Determina si la clase es del sistema (no creada por el usuario)
Es usado o no?	Determina si la clase es utilizada dentro de las condiciones de las tareas del árbol de la tarea del usuario.

Finalmente, al terminar los renglones con la información de los atributos de cada clase se escribe un renglón por cada instancia que se tenga de la clase. Para cada una de estas instancias se tienen propiedades escritas de la misma manera que en el caso de las clases y de los atributos.

Nombre	El nombre de la instancia de la clase
Es del sistema o no?	Determina si la clase es del sistema (no creada por el usuario)
Es usado o no?	Determina si la clase es utilizada dentro de las condiciones de las tareas del árbol de la tarea del usuario.

Todo este ciclo se repite por cuantas clases se tengan en el archivo, de tal forma que el archivo queda como se muestra en el Fragmento de Código 9:

```
Clase 1    props
Atrib1Clase1  props
Atrib2Clase1 props
...          ...
AtribnClase1 props
Instancia1Clase1   props
Instancia2Clase1   props
...          ...
InstanciaNClase1   props

Clase 2    props
Atrib1Clase2  props
Atrib2Clase2 props
...          ...
AtribnClase2 props
Instancia1Clase2   props
Instancia2Clase2   props
...          ...
InstanciaNClase2   props

.......        ......      .........

Clase N    props
Atrib1ClaseN  props
Atrib2ClaseN props
...          ...
AtribnClaseN props
Instancia1ClaseN   props
Instancia2ClaseN   props
...          ...
InstanciaNClaseN   props
```

Fragmento de Código 9

5.5.2.3 Archivo de preferencias de visualización

Finalmente, el tercer archivo de entrada que se utiliza es el archivo de preferencias de visualización.

La visualización del árbol de la tarea en la pantalla está basada en la tecnología de hojas de estilo CSS. Esta tecnología tiene como principios de funcionamiento los llamados "selectores" y las "acciones". *Los selectores* son condiciones que, en caso de cumplirse, dan lugar al establecimiento de una serie de valores en una serie de propiedades, a esto último se le llama *acciones*.

Por ejemplo, si se tiene algo como:

```
[x=y]    -> Selector
{          ->Acciones
        propiedad1=valor1
        propiedad2=valor2
            .....          .........
}
```

Fragmento de Código 10

Implica que, en caso de cumplirse la condición del *selector* (que la propiedad x tenga el valor y). Se ejecutará la *acción* (la propiedad 1 será establecida en el valor 1, la propiedad2 en el valor2, etc). Las propiedades usadas en el selector son propiedades del elemento

examinado. Las propiedades establecidas en las acciones, por otro lado, son características de visualización como el color, el grosor, la forma, etc.

La instancia del modelo de datos que guarda la información del árbol de la tarea forma parte de un objeto llamado *viewer* que provee de un método que toma el archivo *.css*, lo recorre, y busca en el modelo de datos a las tareas que cumplen con los distintos selectores. Una vez ubicadas, define las características de visualización que correspondan y lo muestra al usuario.

En esta herramienta se manejó el archivo example.css que se muestra en el Apéndice A "Hoja de estilo del programa". En él hay algunos segmentos que son de importancia para el manejo de algunos conceptos manejados en anteriores secciones. Estos segmentos se explican a continuación:

El primer segmento (Fragmento de Código 11) muestra los llamados "*renderers*" utilizados en el archivo. Los "renders" son métodos de visualización de las gráficas, es decir, distintas maneras de dibujar las gráficas que muestran los datos, al elegir distintos renders los datos del modelo de datos no cambian, lo único que cambia es la manera en que estos datos se muestran al usuario. Cada uno de los *renderers* de que se dispone gracias a la tecnología *JViews,* permite establecer distintas propiedades de la visualización. Solo aquellos *renderers* establecidos en la sección SDM serán habilitados.

```
SDM {

 graphLayout:"ilog.views.graphlayout.tree.IlvTreeLayout";
 LinkLayout : true;

 StyleSheet: true;
 Coloring: true;
 Interactor: true;
}
```

Fragmento de Código 11 "Sección de habilitación de renderers"

En los siguientes segmentos (Fragmento de Código 12) del archivo se establecen propiedades exclusivas de cada uno de los *renderers* habilitados. El caso del *render StyleSheet* es muy importante ya que, de no establecerse la propiedad, se tomaría la hoja de estilos por defecto y el resto de la presente hoja de estilos sería ignorada.

```
StyleSheet {
        styleSheets : @styleSheets;
}
```

Fragmento de Código 12 "Sección de renderer StyleSheet"

Una vez establecidas las propiedades de los *renderers* se pasa al segmento de *selectores->acciones* (Fragmento de Código 13). En éste segmento se establecen las características visuales que tendrán las tareas seleccionadas, las tareas transferidas, aquellas que sean objetivos fundamentales, etc.

Un ejemplo de cómo se establecen estas características es el siguiente:

```
node:selected {
        strokeColor      : "Red";
 }
```

```
node[Transferida="true"]{
        strokeColor     : "black";
        strokeWidth     : 3;
}
```

Fragmento de Código 13"Selectores->Acciones"

En el Fragmento de Código 13 se estableció que las tareas que hubiesen sido transferidas tendrían un marco más grueso y que las tareas seleccionadas tendrían un color de marco en rojo.

Modificando este segmento del archivo, el analista puede cambiar la forma en que destaca a las tareas transferidas o la manera en que se visualizan las tareas seleccionadas.

Finalmente, en el Fragmento de Código 14, podemos observar como en un segmento del archivo se revisa la propiedad "*color*" y se establece el color de relleno al valor respectivo. Esta sección es la que permite implementar la función *marcar* que provee la herramienta.

```
node[Color="Rojo"]{
        fillColor1 : "Red";
}

node[Color="Azul"]{
        fillColor1 : "Blue";
}

node[Color="Negro"]{
        fillColor1 : "Black";
}

node[Color="Verde"]{
        fillColor1 : "Green";
}

node[Color="Amarillo"]{
        fillColor1 : "Yellow";
}
```

Fragmento de Código 14

5.5.3 Archivos de salida

En lo que se refiere a los archivos de salida, se tiene el que se genera en XML (que , como se había mencionado, hace posible guardar el modelo para abrirlo más tarde tal como se tenía) y otro archivo de fundamental importancia para la herramienta, ya que mediante él se podrán exportar los resultados de la herramienta a un formato estandar internacional y, en particular, a otra herramienta muy utilizada en el resto del desarrollo de un sistema de software.

En efecto, el archivo de salida debe ser leído en Rational Rose®. Dado que el formato en que Rational Rose® guarda su información es propietario, resultó imposible escribir la información en el mencionado tipo de archivos. Por ello se buscó otra manera en que Rational Rose® pudiese importar datos. Así, se encontró una herramienta de Unisys® que se incorporaba a Rational Rose®, y que le permite importar o exportar datos a un archivo en un formato particular de XML llamado XMI. Esta opción fue definitivamente adoptada

para la herramienta ya que el formato XMI es el formato estándar definido para la representación del lenguaje UML en XML.

Sin embargo, el tener la posibilidad de manejar un archivo compatible con Racional, y que se encontraba en formato texto, con posibilidad de manejarlo debido estar escrito en XMI, no nos resolvía el problema completamente. En efecto, aún así, el orden en que se debía escribir cada uno de los elementos de UML era desconocido para nosotros, ya que Unisys® tampoco explica la forma en que se codifica la información.

Se procedió entonces a identificar la forma en que se codifican los elementos UML (como el caso de uso y la asociación entre ellos) exportando, de Rational Rose®, modelos con únicamente dos casos de uso y una asociación. A estos elementos se les establecieron nombres que permitieran su localización en el archivo generado y, de esta manera (y conociendo el formato general de XML), se logró la identificación de los bloques que se necesitaba escribir para cada uno de los elementos, así como la localización en el archivo que éstos debían respetar.

Una vez identificados estos elementos la labor fue clara: se colocó un archivo plantilla en el directorio de la herramienta MAD2UML que no contenía ningún elemento pero que sí incluía al resto del archivo que Rational Rose® podía importar. Mediante Java se leyó el archivo, se buscó el punto en que se debían incluir los casos de uso y se colocaron ahí. Esto último se hizo mediante un recorrido sobre el árbol de la tarea modificado: Para cada tarea, se colocó un bloque "caso de uso" y para cada liga entre tareas, un bloque asociación.

Los únicos datos que cambian en los bloques de cada caso de uso son el nombre de la tarea y el ID (Generado automáticamente por la herramienta), ambos marcados en el Fragmento de código 15:

```
<Behavioral_Elements.Use_Cases.UseCase xmi.id = 'S.10116'>
                <Foundation.Core.ModelElement.name>Nombre tarea</Foundation.Core.ModelElement.name>
                <Foundation.Core.ModelElement.visibility xmi.value = 'private'/>
                <Foundation.Core.GeneralizableElement.isRoot xmi.value = 'false'/>
                <Foundation.Core.GeneralizableElement.isLeaf xmi.value = 'false'/>
                <Foundation.Core.GeneralizableElement.isAbstract xmi.value = 'false'/>
                <Behavioral_Elements.Use_Cases.UseCase.extensionPoint>
                </Behavioral_Elements.Use_Cases.UseCase.extensionPoint>
                <Foundation.Core.ModelElement.namespace>
                  <Model_Management.Model xmi.idref = 'G.33'/> <!-- Use Case View -->
                </Foundation.Core.ModelElement.namespace>
                <Foundation.Core.Classifier.associationEnd>
                  <Foundation.Core.AssociationEnd xmi.idref = 'G.4'/> <!-- {3D88A4BB0238} -->
                  <Foundation.Core.AssociationEnd xmi.idref = 'G.7'/> <!-- {3D88A4BF028D} -->
                </Foundation.Core.Classifier.associationEnd>
</Behavioral_Elements.Use_Cases.UseCase>
```

Fragmento de código 15: agregado para cada tarea del árbol

Para el caso de las asociaciones, es necesario establecer el identificador de la asociación, el identificador de cada uno de los casos de uso que son asociados así como el identificador que se estableció en los bloques respectivos a los mencionados casos de uso.

Es importante diferenciar entre estos dos últimos identificadores, pues el primero es el identificador que, dentro de la asociación, tiene el nodo. Además de ello, se establece el

identificador que se le asignó al nodo durante su codificación. Esto se puede observar en el Fragmento de código 16.

```
<Foundation.Core.Association xmi.id = 'G.2'>
              <Foundation.Core.ModelElement.name></Foundation.Core.ModelElement.name>
              <Foundation.Core.ModelElement.visibility xmi.value = 'private'/>
              <Foundation.Core.GeneralizableElement.isRoot xmi.value = 'false'/>
              <Foundation.Core.GeneralizableElement.isLeaf xmi.value = 'false'/>
              <Foundation.Core.GeneralizableElement.isAbstract xmi.value = 'false'/>
              <Foundation.Core.Association.connection>
                <Foundation.Core.AssociationEnd xmi.id = 'G.3'>
                  <Foundation.Core.ModelElement.name></Foundation.Core.ModelElement.name>
                  <Foundation.Core.ModelElement.visibility xmi.value = 'public'/>
                  <Foundation.Core.AssociationEnd.isNavigable xmi.value = 'true'/>
                  <Foundation.Core.AssociationEnd.isOrdered xmi.value = 'false'/>
                  <Foundation.Core.AssociationEnd.aggregation xmi.value = 'none'/>

<Foundation.Core.AssociationEnd.multiplicity>0..*</Foundation.Core.AssociationEnd.multiplicity>
                  <Foundation.Core.AssociationEnd.changeable xmi.value = 'none'/>
                  <Foundation.Core.AssociationEnd.targetScope xmi.value = 'instance'/>
                  <Foundation.Core.AssociationEnd.type>
                    <Behavioral_Elements.Use_Cases.UseCase xmi.idref = 'S.10018'/> <!-- blabla3 -->
                  </Foundation.Core.AssociationEnd.type>
                </Foundation.Core.AssociationEnd>
                <Foundation.Core.AssociationEnd xmi.id = 'G.4'>
                  <Foundation.Core.ModelElement.name></Foundation.Core.ModelElement.name>
                  <Foundation.Core.ModelElement.visibility xmi.value = 'public'/>
                  <Foundation.Core.AssociationEnd.isNavigable xmi.value = 'false'/>
                  <Foundation.Core.AssociationEnd.isOrdered xmi.value = 'false'/>
                  <Foundation.Core.AssociationEnd.aggregation xmi.value = 'none'/>

<Foundation.Core.AssociationEnd.multiplicity>0..*</Foundation.Core.AssociationEnd.multiplicity>
                  <Foundation.Core.AssociationEnd.changeable xmi.value = 'none'/>
                  <Foundation.Core.AssociationEnd.targetScope xmi.value = 'instance'/>
                  <Foundation.Core.AssociationEnd.type>
                    <Behavioral_Elements.Use_Cases.UseCase xmi.idref = 'S.10016'/> <!-- blabla1 -->
                  </Foundation.Core.AssociationEnd.type>
                </Foundation.Core.AssociationEnd>
              </Foundation.Core.Association.connection>
            </Foundation.Core.Association>
```

Fragmento de código 16: agregado para cada liga del árbol

Este archivo es de fundamental importancia para la trascendencia que la herramienta propuesta tenga en la industria del software. Esto debido a que, gracias a este archivo, la información obtenida con ayuda de la herramienta, puede ser integrada al desarrollo de sistemas realizados con la Rational Rose®. De esta manera, se logra también aumentar las posibilidades de aceptación de la herramienta por parte de los usuarios de Rational Rose®.

5.6 Tecnología utilizada

Si bien se ha explicado todo lo referente a la herramienta, vale la pena hablar un poco sobre la tecnología que se empleó para su implementación. A continuación se dará una breve explicación de esta tecnología.

5.6.1 Jviews

ILOG es una compañía dedicada a la construcción de herramientas para la graficación de diagramas de todos tipos. La compañía produce herramientas en varios lenguajes orientados a objetos, y en el caso particular de java, ofrece el paquete Jviews, homologo a su anterior paquete Iviews dirigido a desarrolladores de C++. Así, Jviews es un conjunto de paquetes de clases que conforman módulos para distintos tipos de graficación, así como algunos

independientes como el SDM (del ingles: Stylable Data Mapper) que contiene todas las clases que permiten el manejo del modelo de datos.

5.6.1.1 Módulos

Los módulos que ofrece Jviews son:

Graphics Framework: Este módulo es necesario para que cualquier otro funcione ya que provee de todo el marco de trabajo sobre el cual el resto de los módulos realizan sus gráficas.

Graph Layout: Este módulo permite realizar gráficas de redes, flujos de trabajos, jerarquías (como en el caso del árbol MAD) y todo aquello que en términos matemáticos se entienda como una gráfica.

Maps: Gracias a este módulo es posible la realización de mapas geográficos

Gantt Chart: Este módulo permite realizar gráficas de Gantt

Stylable Data Mapper **(SDM)**: Este módulo realice la liga entre la visualización que provee *Jviews* y su modelo de datos, es decir, implementa la arquitectura modelo-vista-controlador.

Charts: El módulo más reciente de *Jviews* permite la realización de gráficas de pie, de barras, etc de tal manera que se representen grupos de datos X-Y.

La herramienta provee de un modelo de datos por defecto. Este modelo consta de nodos y ligas, y en ambos casos es posible agregar las propiedades que se deseen, además de aquellas que permiten su manejo visual.

El modelo de datos cumple con el estándar DOM (*Domain Object Model*) de tal manera que en cualquier momento puede ser guardado o tomado de archivos XML. Esto permite una gran conectividad de los programas que se hagan con *Jviews* hacia el resto de los programas e inclusive hacia la Internet.

El modelo de datos por defecto es el usado por la mayor parte de las aplicaciones. Aún así existen muchos casos en que el modelo no se ajusta a las necesidades del programador. Para estos casos *jviews* no pone limitantes, ya que provee una interfaz estándar que puede ser implementada por el programador, de tal manera que desarrolle su propio modelo de datos y, mientras respete las convenciones de la mencionada interfaz, el modelo será aceptado por el resto de *jViews*, es decir, podrá utilizar el resto del marco de trabajo de la misma manera que si usara el modelo de datos por defecto.

5.6.1.2 Visualización

En lo que se refiere a la visualización, describiremos brevemente el módulo *Graph Layout*, debido a que es el módulo que usaremos para la graficación de los árboles MAD*.

El módulo *Graph Layout* provee la posibilidad de graficar un modelo de datos de distintas formas mediante algoritmos que se listan a continuación:

• *Topological Mesh Layout* (TML) • *Spring Embedder Layout* • *Uniform Length Edges Layout* • *Circular Layout (Ring/Star)* • *Hierarchical Layout* • *Link Layout* • *Tree Layout* • *Random Layout* • *Bus Layout* • *Grid Layout*	Cada uno de estos algoritmos organizan a la gráfica de distintas formas geométricamente hablando, por ejemplo, si se utiliza el *Circular Layout*, la gráfica se distribuirá a manera de circulo, de usarse el *Uniform Length Edger*, el algoritmo buscará colocar los nodos de tal manera que la distancia entre ellos sea similar. En general, cada uno de los *Layouts* es útil para alguna aplicación en particular.

De todas estas formas de organizar el modelo de datos, es claro que para nuestro caso se usará el *Tree Layout* dado que se trata, precisamente de un árbol lo que se desea modelar.

Ahora bien, si bien es cierto que el *Layout* elegido controlará la distribución de los nodos y las ligas, la apariencia de éstos será controlada por un archivo .css, es decir una hoja de estilo. Como en el caso de XML, el uso del estándar *Hojas de Estilo en Cascada* (CSS del ingles: *Cascade Style Sheets*) permite una comunicación mayor con otras aplicaciones así como con el Internet. El uso de estas tecnologías le permite a *jViews* crecer junto con estándares internacionales de enorme fuerza que también están creciendo.

Para usar las hojas de estilo se cuenta con una serie de propiedades tanto para los nodos como para las ligas que permiten al usuario, o al programador, cambiar de manera dramática la forma en que se visualizan los datos. A continuación mencionamos las propiedades más notables:

Nodo:

Forma Tamaño Tipo de relleno Color de relleno Borde • Color • Grosor • Tipo de línea (punteada, continua, etc)	Estas propiedades controlan las características de la forma geométrica que presentará el nodo al desplegarse en la gráfica. Así, propiedades como el tipo de relleno, asignarán la manera en que se coloreará el interior del nodo, ya sea en gradiente, en textura o en color sólido. De la misma manera existen propiedades para establecer el o los colores de este relleno, el tamaño y forma del nodo, las características del

Etiqueta • Cadena • Tipo de letra • Posición • Multilínea • Autowrap • Color	borde, de la etiqueta o letrero que presentará.
Icono	
Forma de redimensionar	

Ligas:

Tipo de relleno	Como en el caso del nodo, estas propiedades controlan las características de la forma geométrica que presentará la liga al desplegarse en la gráfica. Así, propiedades como el tipo de relleno, asignarán la manera en que se coloreará el interior de la liga, ya sea en gradiente, en textura o en color sólido. De la misma manera existen propiedades para establecer el o los colores de este relleno. Para el caso de la liga, por otro lado, también se establecen propiedades especiales para una línea, como es el caso de su grosor, color y tipo de línea. Finalmente se establecen las propiedades que presenta en cuanto a la flecha que da dirección a la línea, esto es el color, tamaño, forma, posición y orientación (dirección) que tendrá la mencionada flecha. Finalmente, se establece el estilo que tendrán las líneas de las ligas, este puede ser en ondas, curveado, o recto.
Color de relleno	
Borde • Color • Grosor • Tipo de línea (punteada, continua, etc)	
Flecha • Color • Tamaño • Forma • Posición • Orientación	
Estilo (curvo, en ondas, etc)	

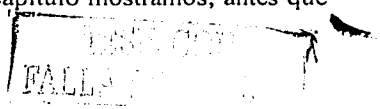
No es objetivo del presente trabajo el ir a fondo en la descripción de éstas propiedades. Aún así, la referencia completa se encuentra en el Manual de Usuario del SDM dentro del producto.

Es de destacar que ésta forma de controlar el despliegue de los nodos resultó determinante para la realización de nuestra herramienta. Mediante ella y con la posibilidad de agregar propiedades a los nodos resultó posible agregar importantes funciones a la herramienta lo cual le dio una gran usabilidad.

Para más información sobre *jViews* y el resto de las herramientas de ILOG referirse a la página web de ILOG en www.ilog.com.

5.7 Discusión Final

Como se ha visto a lo largo de este Capítulo, y del anterior, las herramientas han cobrado gran importancia en el desarrollo de software actual debido a las necesidades de acelerar y dotar de mayor calidad a los procesos de desarrollo. En este capítulo mostramos, antes que



nada, las características del proceso al que asistirá la herramienta; de esto último se derivaron los requerimientos que se tendrían para la herramienta. Con estos requerimientos definidos presentamos la manera en que se diseñó e implementó con el fin de cumplir con dichos requerimientos.

La herramienta obtenida deberá permitir, por lo tanto, cubrir las necesidades de los analistas de la tarea en lo que se refiere a la traducción de este análisis, a casos de uso. Estos casos de uso que deben ser utilizados en el desarrollo de software, principalmente por aquellos desarrolladores que se guíen por el proceso unificado de desarrollo de software y que se apoyen en la herramienta Rational Rose® para la documentación del este proceso.

El objetivo general al que pretende ayudar el presente trabajo es el ayudar a la incorporación del análisis de la tarea a los desarrollos de software actuales. La herramienta, sin embargo, persigue dos objetivos más particulares.

El primer objetivo particular que se busca alcanzar es la asistencia al analista de la tarea para que, a partir del árbol de la tarea que obtenga, pueda obtener un árbol de la tarea modificado enfocado a la tarea que se realizará mediante el sistema.

El segundo objetivo particular se enfoca en la incorporación de estos resultados a los procesos de desarrollo de software actuales mediante la traducción del árbol modificado a casos de uso y su incorporación a la herramienta Rational Rose®.

Ambos objetivos fueron cumplidos por el desarrollo por lo que consideramos que las perspectivas que se tienen para el uso real de la herramienta son amplios. Por un lado, consideramos que para el analista de la tarea la herramienta le permitirá realizar su tarea de una manera mucho más eficiente, sobre todo en árboles de gran tamaño. Por otro lado, el cumplimiento del segundo objetivo permitirá que los desarrollos que no consideraban al análisis de la tarea lo incorporen al ya no tener los inconvenientes de incompatibilidad de lenguajes y de lentitud que tenía anteriormente.

6 Caso de estudio

El objetivo del presente Capítulo es mostrar de manera clara la forma en que se utiliza la herramienta propuesta en un caso real; así mismo, se muestra como algunas necesidades del analista para integrar al Análisis de la Tarea al Proceso Unificado, son resueltas mediante la herramienta. Este Capítulo es de suma importancia para nuestro trabajo, ya que nos permite afirmar que la herramienta es útil en un caso concreto y evita que el trabajo quede únicamente como una propuesta teórica, sin saber si es aplicable realmente o de que manera se haría esto.

En la primera parte del Capítulo presentaremos la forma en que el caso de estudio surgió, la manera en que se analizó el problema y cómo fue modelado el árbol de la tarea del usuario utilizando IMAD*. Por otro lado, se analizará a fondo el uso de la herramienta MAD2UML comenzando con la manera como se obtuvo el *árbol de la tarea del usuario* a partir del archivo generado con IMAD*. También se describirá cómo asistió la herramienta en el análisis del árbol obtenido y en la creación del *árbol modificado*. Finalmente se describirá cómo, a partir del *árbol modificado*, la herramienta generó casos de uso preliminares del sistema mismos que son leídos en Rational Rose® donde el analista de casos de uso podrá obtener el modelo de casos de uso del sistema. Todo este proceso se muestra en la Figura 32, donde se puede observar en el recuadro la parte del proceso de captura de requerimientos que es asistida por MAD2UML.

6.1 Planteamiento

El caso a revisar consiste en la captura de requerimientos para el sistema de contabilidad del Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la UNAM.

Este caso de estudio fue tomado por cumplir con los siguientes aspectos: El primer punto es que se trata de un ejemplo real; es decir, el sistema del que se habla entrará en operación con los datos reales del CCADET. Por otro lado se tomó en cuenta el tamaño del proyecto: la tarea del usuario a analizar es de un tamaño suficiente como para mostrar las características más importantes de la herramienta y al mismo tiempo es lo suficientemente acotado para permitir mostrar los conceptos de manera clara y en el tiempo de una tesis. Finalmente, se tomó en cuenta la accesibilidad que teníamos para consultar con el analista en persona sobre los criterios utilizados en su análisis; esto con el fin de aplicar esos mismos criterios al momento de utilizar la herramienta, además de poder contar con sus observaciones sobre la misma.

Los requerimientos de este sistema fueron levantados por Alejandro López Kolkovsky [5]; por ello, nos basaremos en el *árbol de la tarea del usuario*, generado en el trabajo referido y que se muestra en la Figura 33, para mostrar la aplicación de la herramienta. El sistema a desarrollar debe mejorar un sistema ya existente en el CCADET que permite realizar la contabilidad, mantener un registro de los préstamos que se realizan (prestamos que se entregan para los proyectos del CCADET), así como de los pagos recibidos; además de

ello, permite generar los reportes pertinentes sobre las acciones anteriores; a estos grandes rubros los llamaremos las tareas "Realizar Préstamos", "Revisar Registros" y "Elaborar Reportes" dado que así se le llamó en el árbol realizado por el analista.

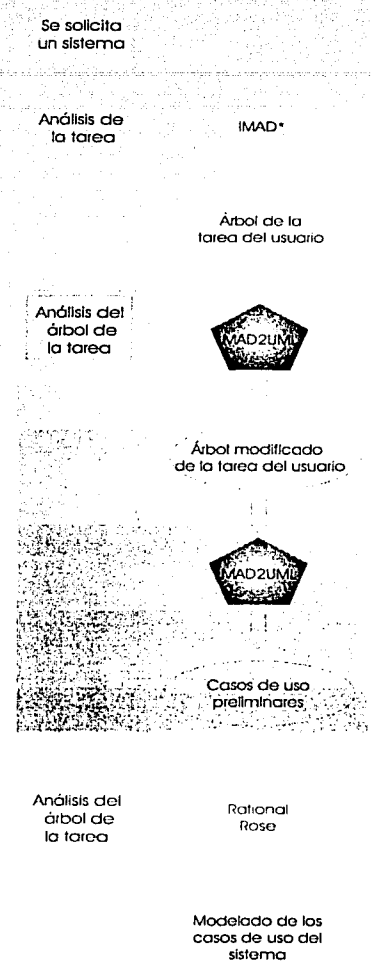


Figura 32 "Diagrama general del proceso de Captura de Requerimientos"

Dentro de la tarea *Realizar préstamos* se incluye la serie de tareas burocráticas que se requiere realizar secuencialmente para obtener el préstamo, desde la solicitud del cheque, hasta el registro de su entrega. Por otro lado, la tarea *Revisar registros* (la revisión de los

estados de cuenta de los deudores, de la institución, etc.) es una de las más tardadas para el usuario y esto se debe a que es completamente manual.

Finalmente, en lo concerniente a *Elaborar reportes*, las actividades englobadas en esta tarea se refieren al llenado de formatos para la UNAM, la formación de paquetes de documentación y la impresión de reportes.

López Kolkovsky [5]realiza en su análisis de la tarea del usuario entrevistas no dirigidas y entrevistas dirigidas, que le permiten conformar el árbol de la tarea mostrado en la Figura 33. Como se puede observar, el análisis obtuvo un buen detalle de las tareas y esto se refleja en un árbol de la tarea amplio, que llega a un nivel de detalle en el que las tareas son del tipo *entrar al sistema de caja* o *entregar solicitud*.

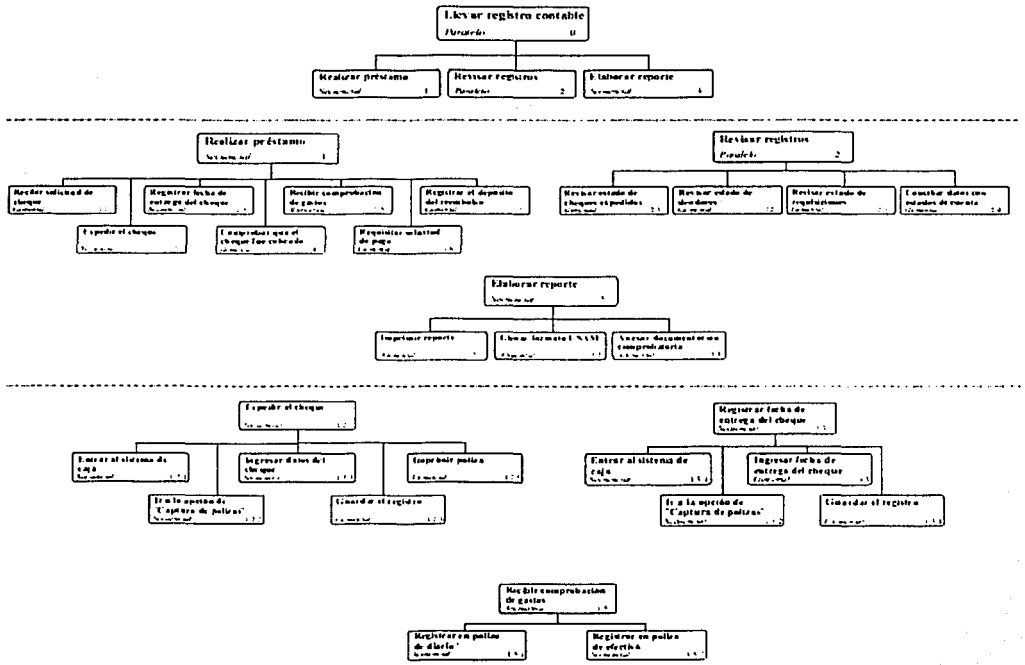


Figura 33 "Árbol de la tarea del usuario de López Kolkovsky"

6.2 Modelado MAD*

El trabajo antes referido presenta las fichas del árbol de la tarea del usuario en su apéndice A. A partir de esta información, las tareas fueron capturadas en la herramienta IMAD*, de tal manera que se pudieran leer posteriormente desde MAD2UML.

El árbol del que se parte presenta una serie de objetos, atributos y "*sub atributos*" que son usados como un elemento para expresar las condiciones de cada una de las tareas del árbol. El formato de IMAD*, por otro lado, incluye clases con objetos y atributos, pero no contempla que estos atributos encierren sub-atributos. Por lo anterior, los *sub atributos* del árbol de la tarea del usuario tuvieron que ser ajustados a atributos sencillos para ser capturadas en IMAD*.

La manera en que se resolvió esta incompatibilidad entre la manera en que se expresan los atributos en el árbol original y la manera en que lo modela IMAD*, fue mediante el cambio del tipo de dato de algunos atributos; esto es, si un objeto como *cheque* tenía un atributo *póliza* y éste a su vez tiene varios atributos como Firma, fecha de entrega, etc; se trasladaba este esquema a uno en donde el objeto cheque tenía sus atributos normales, más otros que se forman del atributo y sus sub-atributos de tal manera que Póliza_firma y Póliza_fechaEntrega pasan a ser atributos del objeto cheque y el atributo Póliza desaparece.

Modelo anterior	Modelo actual
Cheque • Poliza o Firma o Fecha de entrega	Cheque • Poliza_Firma • Poliza_FechadeEntrega

El árbol ya capturado en IMAD* se presenta en la Figura 34.

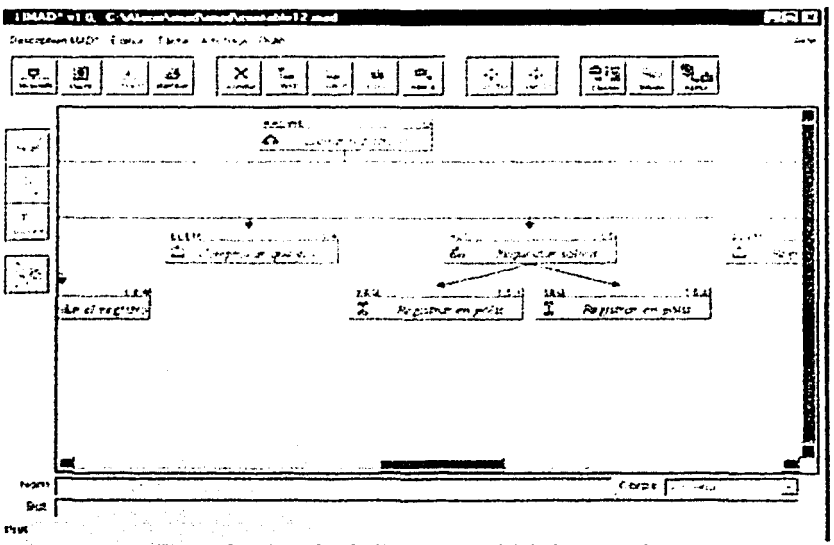


Figura 34 "Interfaz de IMAD con el árbol capturado"*

6.3 Uso de la herramienta

La herramienta propuesta en este trabajo, llamada MAD2UML, partirá del archivo generado por IMAD*. El primer paso a realizar fue la lectura del mencionado archivo, así como de los objetos involucrados en él -guardados en un archivo *.obj* (ver 5.5.2.2)-. Para ello, únicamente se utilizó la función "*Abrir MAD*" en el menú *Archivo*. Mediante esta función la herramienta lee los dos archivos generados por IMAD* (el árbol de la tarea y el de los objetos) y carga la información en las instancias del modelo de datos correspondientes (Ver 5.4.2).

Al terminar la carga de la información, el árbol de la tarea del usuario es mostrado como se muestra en la Figura 35 .

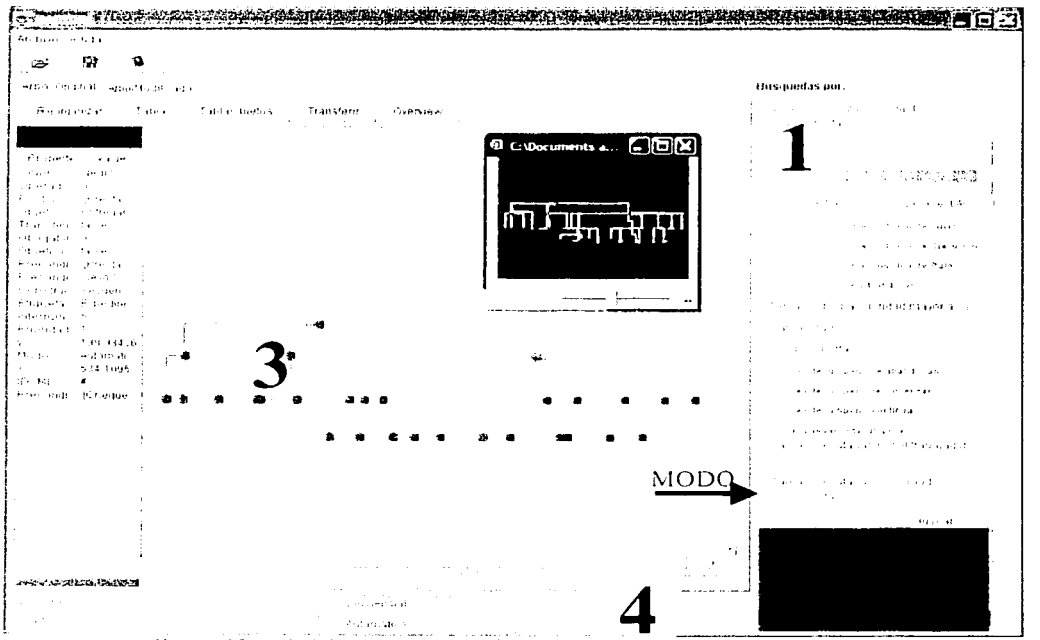


Figura 35 "Árbol de la tarea del caso de estudio cargado en la herramienta"

Con esta información en el sistema, el analista cuenta con todos los elementos para analizar el árbol de la tarea y, basado en los resultados de este análisis, conformar el árbol de la tarea modificado.

6.3.1 Análisis del árbol de la tarea del usuario y creación del árbol modificado.

Una vez cargada la información del árbol de la tarea del usuario en la herramienta, se procede a su análisis de acuerdo con diversos criterios.

Un primer criterio que interesó al análisis que nos ocupa es el modo en que se realizan las tareas, puesto que consideramos que las tareas manuales serán las primeras en descartarse para la conformación del árbol de la tarea modificado. Para ello, el analista puede realizar una búsqueda (sección 1 de la Figura 35) en el árbol de aquellas tareas que cumplan con la mencionada condición. Esto se realiza al seleccionar, en la zona señalada como "modo" en la imagen, el valor "manual". Con esto se obtiene en la tabla de resultados (seccion 2), una lista de las tareas manuales en el árbol, además de que todas ellas son seleccionadas dentro de la zona de visualización(sección 3). Al tenerse seleccionadas a las tareas, se da la oportunidad al usuario de marcar con un color todas estas tareas(parte baja de la sección 2), lo que resulta muy útil para visualizar globalmente las tareas con cierta característica (en este caso las manuales) dentro de todo el árbol. De quererse así, también podrían ser transferidas en grupo al árbol modificado. En este caso sólo se utilizó la herramienta "marcar" para colorear en gris claro a todas aquellas tareas manuales, debido a que las tareas "no manuales", que son las que realmente nos interesan, aún tendrán que ser pasadas por varios filtros más antes de pasarlas al árbol modificado. Además, las tareas manuales también tendrán que ser analizadas con más cuidado ya que es posible que algunas de ellas deban conservarse y modificarse a automáticas o interactivas. Esto último se puede hacer mediante las listas desplegables que se tienen en la sección 4 de la Figura 35 .

El árbol del ejemplo contiene muchas tareas manuales. En la Figura 36 se muestran las tareas mencionadas marcadas en gris claro.

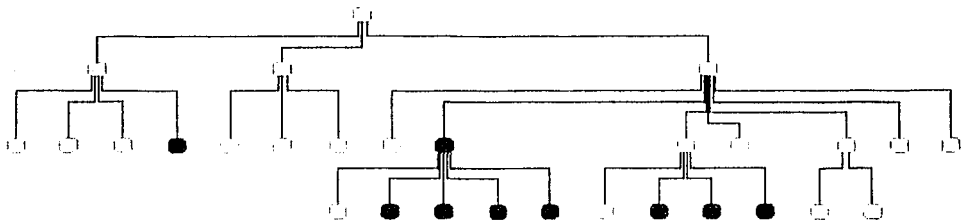


Figura 36 "Tareas manuales del árbol"

El siguiente criterio que se utilizó en el análisis del árbol, y que es comúnmente usado en los análisis de la tarea, es buscar aquellas tareas relacionadas con un objeto particular, y tomarlo como parámetro para la definición del nuevo árbol de la tarea. En este caso se tomó la decisión de eliminar del árbol modificado todas aquellas tareas que tuvieran que ver con el manejo de la interfaz del sistema que se manejaba anteriormente; esto debido a que el sistema que se tiene en operación desaparecerá y, por lo tanto, las tareas que se refieren al manejo de su interfaz, (ej. "seleccionar la opción 1 del menú") no serán realizadas de la

misma manera en el nuevo sistema. Un ejemplo de esto fue que para detectar este tipo de tareas se buscaron las tareas que involucraran al objeto "sistema caja". Estas tareas fueron marcadas en rojo de la misma manera, y por la misma razón, que las tareas manuales. Después de marcar estas tareas el árbol quedó como se muestra en la Figura 37. Es de notarse que las tareas aún verdes (en este color aparecen todas las tareas al iniciar el proceso) no son ni manuales ni tienen que ver con la interfaz del sistema anterior.

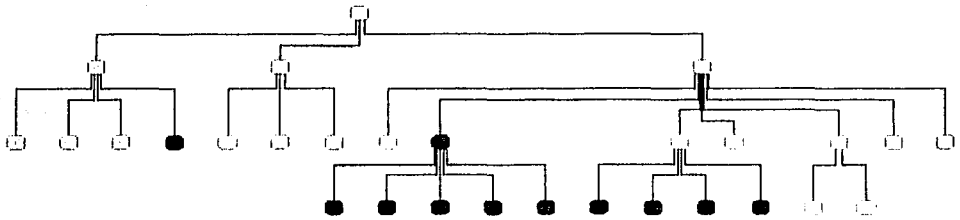


Figura 37 "Tareas relacionadas con el manejo del sistema actual"

Otro caso en el que se buscó con base en un objeto es el siguiente: se decidió que para el sistema que se desarrollará, las tareas relacionadas con los préstamos eran de fundamental importancia ya que se trata de un elemento presente en los objetivos de las tareas de la más alta jerarquía del árbol, por lo que sin este elemento la tarea completa pierde sentido. Por lo anterior, el analista deseaba agrupar en una sola una rama a las tareas relacionadas con dicho objeto. Con ayuda de la herramienta, el analista puede realizar la búsqueda de la misma manera que lo hizo para las tareas relacionadas con "sistema caja" pero buscando aquellas relacionadas con el objeto "préstamo". El resultado de esta búsqueda será un grupo de tareas que estarán seleccionadas y pueden ser trasladadas, con un solo clic del ratón, al árbol modificado. Una vez ahí, es posible agruparlas a conveniencia.

Al realizar lo anterior, nos damos cuenta de que las tareas relacionadas con él objeto "préstamo" se encuentran principalmente en las ramas de subtarea "realizar préstamo"[4]. De ahí podemos deducir que ese sector del árbol es muy importante. Para poder distinguir estas tareas posteriormente, las marcamos en azul. Además de ello, las transferimos al árbol modificado.

El árbol de la tarea queda, con esta última modificación, como se muestra en la Figura 38.

[4] Debido a que el nivel de acercamiento de la gráfica no permite leer los textos de las tareas se señala en un círculo a la subtarea "Realizar Préstamo"

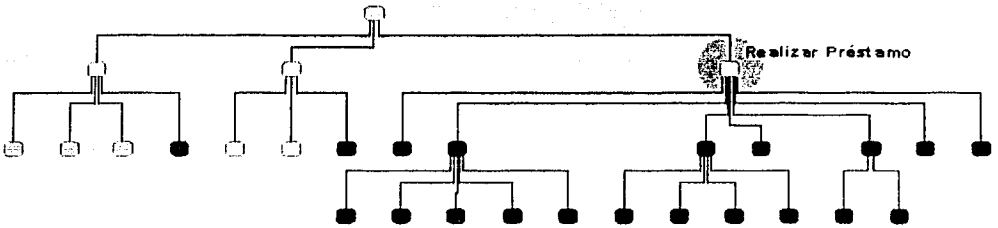


Figura 38 "Tareas relacionadas con el objeto Préstamo"

La búsqueda por objetos también resultó útil para buscar el objeto "cheque". Este objeto dejará de ser utilizado cuando se tenga el nuevo sistema, por lo que se desea eliminarlas: en este caso, la herramienta permite buscar las tareas relacionadas con este objeto, y marcarlas en determinado color que indique que son candidatas a desaparecer. Al realizar el ejercicio, sin embargo, el resultado de la búsqueda nos selecciona tres tareas, como se aprecia en la Figura 39. En todas ellas vemos el color azul que denota a las tareas relacionadas con "préstamo" por lo que sabemos que estas tareas ya fueron consideradas en el árbol modificado y por lo tanto ya no son marcadas.

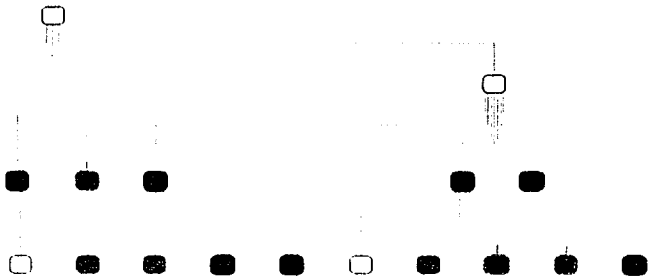


Figura 39 "Figuras relacionadas con el objeto "Cheque"

El último caso en que se requirió la búsqueda por objetos fue para el objeto "reporte": se buscaron las tareas relacionadas con este objeto y se encontró que se trataba de aquellas pertenecientes al bloque de la tarea "Elaborar reporte". Al igual que en el caso de las tareas relacionadas con el objeto "préstamo", se marcaron estas tareas en azul y se les transfirió al árbol modificado. Se decidió que este sector del árbol debía mantenerse debido a que todo el bloque está encaminado a cumplir con uno de los tres principales objetivos del sistema; a saber, la obtención de reportes. Con estas tareas marcadas, el árbol de la tarea del usuario queda como se aprecia en la Figura 40.

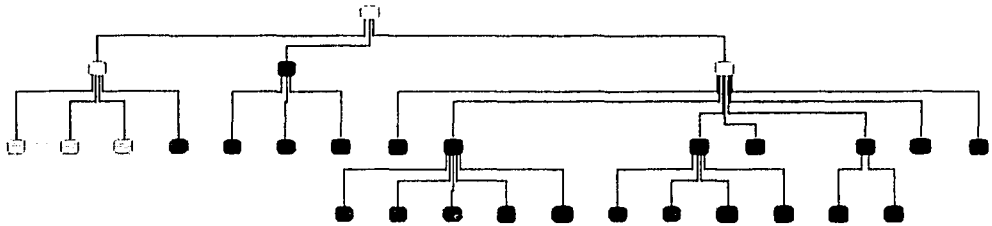


Figura 40 "Árbol de la tarea del usuario completamente marcado"

Como último paso de la transferencia de tareas al árbol modificado, se transfieren las ligas necesarias para unir a las tareas que fueron enviadas separadamente. Con esto, el árbol modificado queda como se muestra en la Figura 41.

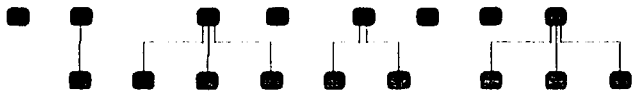


Figura 41 "Árbol de la tarea modificado antes de ligar tareas aisladas"

En caso de tenerse tareas aisladas que se desee unir de distinta manera a como estaban originalmente es posible crear estas ligas directamente en el árbol modificado, también es posible transferir tareas que no se había contemplado por ser necesarias para la estructura del árbol. El árbol modificado, con estos cambios se muestra en la Figura 42.

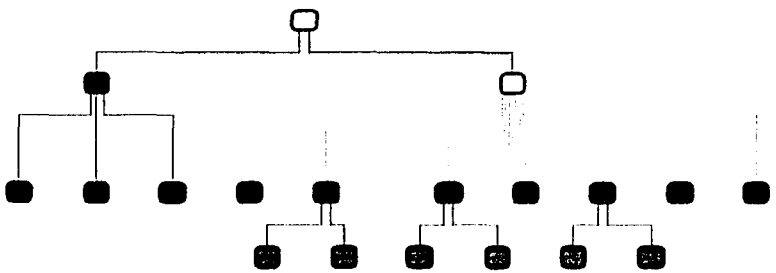


Figura 42 "Árbol de la tarea modificado"

El siguiente paso para la obtención del árbol modificado definitivo es la creación de tareas y ligas nuevas, así como la eliminación de aquellas tareas que hubiesen sido transferidas pero que finalmente se decide que no se requieren en el nuevo árbol.

Para el caso de nuestro ejemplo, no consideramos la creación de tareas directamente en el árbol modificado ya que en una fase tan temprana del proyecto no resulta clara su necesidad, aún así, es conveniente mencionar que esto puede hacerse en los casos en que ya se tiene una idea clara de lo que el nuevo sistema debe agregar a la tarea. En esos casos deberá hacerse uso de esta función que provee la herramienta. En este punto es necesario destacar que una vez obtenidos los casos de uso iniciales, el desarrollo del nuevo sistema continuará su proceso encontrando incremental e iterativamente los casos de uso definitivos del sistema por lo que todas las tareas que aquí falten deberán ser incorporadas eventualmente.

Un último aspecto a destacar antes de continuar con el desarrollo del ejemplo son las posibilidades de búsqueda dentro de un árbol que la herramienta ofrece. El ejemplo, por su tamaño, no requiere de búsquedas complejas; sin embargo, es posible hacer búsquedas de gran complejidad: por ejemplo, si se tuviese que desarrollar un sistema para una realizar los trámites del INFONAVIT, tendríamos tareas generales como *captura de viviendas a ofertar*, *asignación de peritos*, *captura de avaluos*, *captura de derecho habientes*, *reportes de pagos*, etc; estas tareas serían realizadas por muy diversos actores como lo son *Bancos*, *Infonavit*, *Solicitante*, *Constructoras* (*Oferentes*), etc.

Para un sistema como éste podríamos pensar en organizar, módulos con respecto a algunos objetos como el avalúo, pero también con respecto a quien realiza la tarea. Por ejemplo, se pensaría en un módulo de *avaluos* que se presentaría a los peritos, para lo cual se buscaría entre todo el árbol aquellas tareas relacionadas con el trabajador "perito" y con el objeto "avaluo" en el modo "manual". Estas tareas serían marcadas y agrupadas en el árbol modificado con el fin de obtener un módulo que automatice todo ese proceso que realicen los peritos. Por otro lado, probablemente se buscarían las tareas que realice el trabajador "banco" y dentro de ellas las relacionadas con el objeto "avaluo" para obtener otra parte del módulo "avaluos" que se realizaría por el trabajador "banco".

Estas posibilidades, en un proyecto de gran magnitud, resultarían en información de mucha utilidad para el analista ya que se harían búsquedas sobre cientos de tareas que permitieran encontrar tareas con características muy particulares que de manera manual resultaría un trabajo costoso en tiempo y recursos humanos. En el caso que nos ocupa, realizar búsquedas de este tipo hubiera resultado ocioso dado que, por ejemplo, las tareas tienen la misma prioridad y son realizadas por un solo trabajador.

6.3.2 Generación de casos de uso

Una vez que se definió el "árbol de la tarea modificado", se procedió a obtener los casos de uso preeliminares del sistema. Este punto es de fundamental importancia debido a que es precisamente mediante la generación de estos casos de uso que se liga el análisis de la tarea del usuario a los procesos de desarrollo de software. Como ya se comentó, los casos de uso son el elemento básico para expresar los requerimientos del usuario dentro del Proceso Unificado de Desarrollo de Software , así como en la mayoría de los procesos de este tipo. Los casos de uso son generados mediante el método de mapeo de casos de uso en torno a

las tareas propuesto en . Este método propone la creación de un caso de uso por cada tarea del árbol. Aquellas tareas que tienen subtareas reflejarán esta jerarquía mediante una "relación"[5] entre los casos de uso respectivos.

Los casos de uso generados son escritos en el lenguaje XMI, (un sublenguaje de XML para codificar UML) en un archivo .XML gracias a la función de "Exportar casos de uso" del menú "Archivo". Al estar escrito en el estándar internacional XMI, el archivo generado podrá ser leído por aquellas herramientas que implementen la lectura de dicho estándar.

En el caso que estamos estudiando, tomamos el árbol modificado obtenido mediante la herramienta y la exportamos en un archivo llamado "casosdeuso.xml".

6.3.3 Resultados

El propósito de la presente sección es mostrar como se leen los resultados del caso de estudio en la herramienta líder del mercado de documentación de desarrollos de software en UML, Rational Rose®.

Efectivamente, el archivo generado por MAD2UML es legible desde Rational Rose® gracias a la implementación hecha por Unisys para la lectura y escritura de XMI en Rational Rose®. Para lograr esto es necesario instalar el add-in de Unysis para Rational Rose (el ejemplo fue probado en Rational Rose 2000 Enterprise Edition, pero debe funcionar en versiones posteriores). Con el add-in instalado, el menú "archivo" de Rational agrega las opciones importar y exportar a XMI; utilizando la primera opción se importó el archivo generado en MAD2UML. Por otro lado es importante hacer notar que la definición del lenguaje UML en XMI se encuentra en un archivo llamado UML.dtd; por ello, para la correcta lectura de cualquier archivo desde Rational Rose es necesario tener el UML.dtd en el mismo directorio del archivo que se desea leer.

Lo que se obtiene en Rational Rose al importar el archivo es una serie de casos de uso y las asociaciones entre ellos (esto se observa en la Figura 43). En la Figura 44 se puede observar que cada caso de uso tiene asociaciones con sus subcasos de uso, así como con su caso de uso padre. Todo esto tiene una relación directa con el árbol de la tarea modificado que se tenía; al examinar la lista de casos de uso podemos notar que se tiene uno para cada una de las tareas de las que constaba el árbol referido.

Por otro lado, en la Figura 44 se observa que cada caso de uso tiene asociaciones. Estas asociaciones son obtenidas a razón de dos para cada una de las ligas que se tenían en el árbol. Si se tenía, por ejemplo, una liga desde la tarea "Llevar registro contable" a una tarea "Elaborar Reporte", aparecerán ahora una asociación en el primer caso de uso (indicando que es padre del segundo) y una más en el segundo caso de uso (indicando que es hijo del primero). En la Figura 44 se puede apreciar que las asociaciones de padre a hijo son dibujadas en tonos tenues y la relación con el padre es denotada con color azul intenso.

[5] "Relación" es una liga entre casos de uso definida en UML

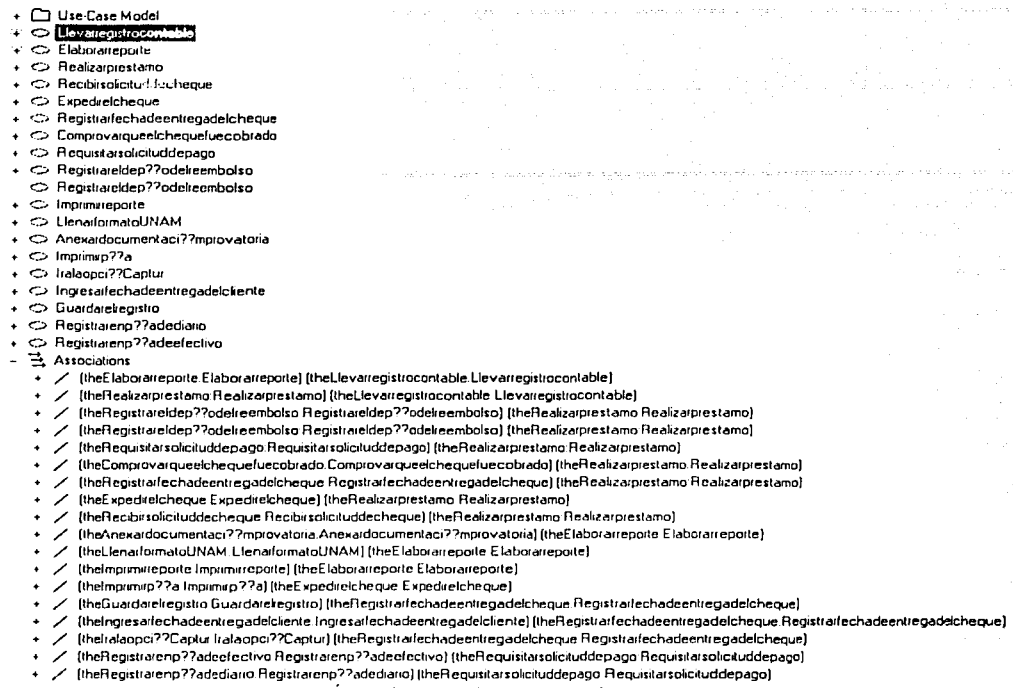


Figura 43 "Árbol de casos de uso obtenido en Rational Rose"

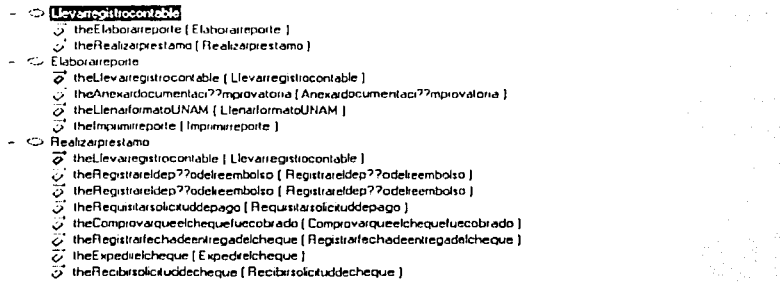


Figura 44 "Casos de uso con sus relaciones"

Para poder formar el diagrama de casos de uso equivalente al árbol modificado obtenido en MAD2UML, solo es necesario arrastrar los casos de uso hacia un diagrama y las asociaciones se dibujan automáticamente. En el caso del ejemplo, arrastramos todos los casos de uso hacia un diagrama, obteniendo es el mostrado en la Figura 45. Aquí podemos observar como las asociaciones se dibujaron con algunos ornamentos.

El primer aspecto a observar en el diagrama es la flecha, como se puede observar la flecha siempre apunta hacia el caso de uso padre. Se recomienda colocar a el caso de uso correspondiente a la raíz del árbol de la tarea, en este caso "Llevar registro contable" al centro del diagrama ya que generalmente facilita el distribuir al resto de los casos de uso a su alrededor.

También se puede observar que en los extremos de la asociación se escribe "X..X", donde las X pueden ser 0,1 o *. Esto se utiliza en la notación de UML para denotar la multiplicidad en las relaciones, es decir, en una relación 1..1 por ejemplo implica que para cada elemento de la parte izquierda de la relación existe uno y solo uno de la parte derecha. Tal es el caso de la mayoría de los casos de uso que es conformado por uno de cada uno de sus sub-casos de uso. Una relación como la que tiene el caso de uso principal con sus subcasos de uso (0..* a 0..*) implica que es posible que el llevar el registro contable se realice muchas ocasiones y la actividad consiste en elaboren ninguno o más reportes y ninguno o más préstamos.

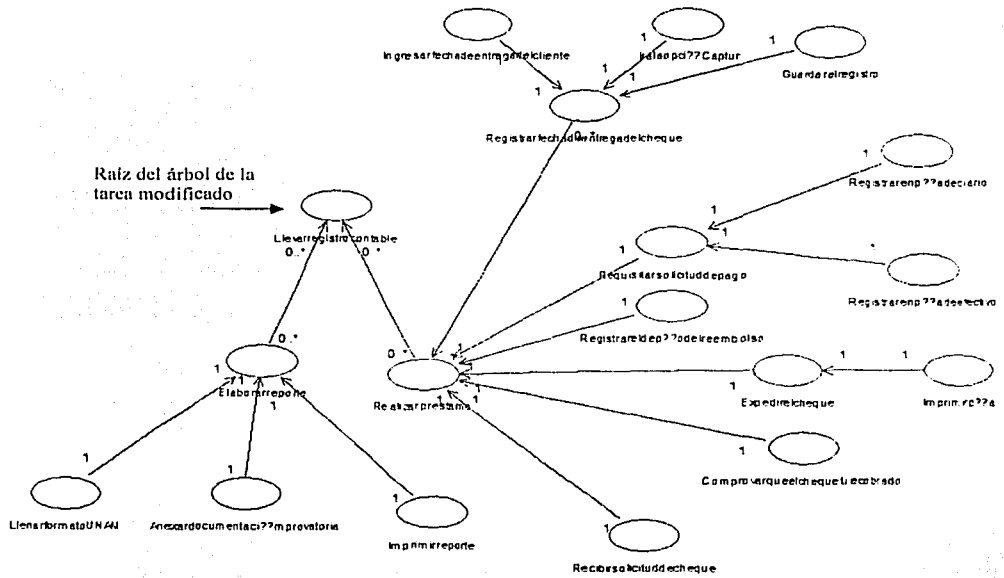


Figura 45 "Diagrama de casos de uso obtenido"

Con este diagrama es posible continuar con el desarrollo del software mediante cualquiera de los varios procesos de desarrollo de software disponibles hoy día. En particular, el Proceso Unificado ha sido adaptado para utilizar los artefactos del análisis de la tarea en los trabajos de Kolkovsky y Pinzón .

6.4 Discusión

Como se pudo observar a través del desarrollo del caso de estudio, la herramienta puede resultar de enorme utilidad para el análisis de la tarea de sistemas de gran tamaño. Se mostraron aquí algunas de las posibles búsquedas que se pueden realizar con la herramienta sin dejar de remarcar las mayores posibilidades de exploración del árbol que ofrece la herramienta.

El ejemplo permitió también mostrar la manera en que los criterios de búsqueda pueden ser útiles para un análisis concienzudo de la tarea del usuario y de qué manera las herramientas de marcado en colores permiten clarificar el contenido del árbol para el analista.

Por otro lado, el ejemplo mostró la manera en que los resultados del análisis son traducidos en un árbol de la tarea modificado y nos permitió demostrar como la herramienta ayuda en la conformación de este nuevo árbol.

Finalmente, pudimos ver como la herramienta nos permite exportar el árbol de la tarea modificado a un archivo XMI que, a su vez, puede ser leído en Rational Rose con lo que se completa la liga hacia los procesos de desarrollo de software actuales.

El proceso que se siguió desde el inicio del ejemplo se puede visualizar en la Figura 46. El proceso consistió de los siguientes pasos:

- Se analizó la tarea.
    - De este análisis se generó un árbol de la tarea.
- Este árbol fue capturado en IMAD*
    - El árbol fue guardado en un archivo .MAD.
- El archivo generado por IMAD* se leyó en MAD2UML.
- Se analizó la información.
- Se generó el árbol de la tarea modificado.
- Se generaron los casos de uso del árbol modificado.
    - Estos casos de uso se guardaron en un archivo XML.
- El archivo XML fue leído desde Rational Rose.
- Se realizó el diagrama de casos de uso preliminares en Rational Rose.

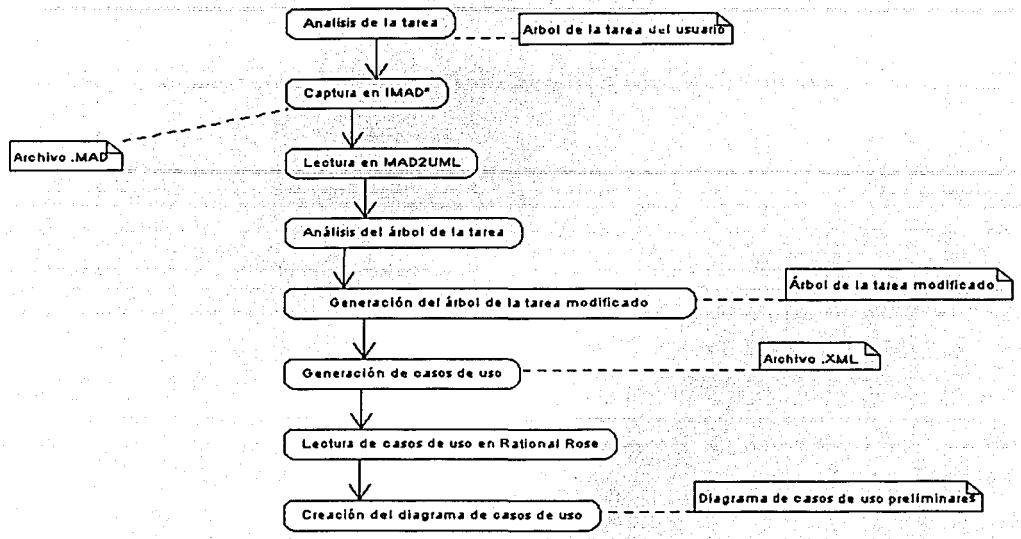


Figura 46 "Diagrama de actividades del proceso de obtención de los casos de uso preliminares"

A partir del diagrama de casos de uso preliminares obtenido, es posible continuar con la mayor parte de los procesos de desarrollo de software actuales.

En términos generales el ejemplo nos permitió visualizar más claramente como se puede llevar el proceso completo, desde el análisis de la tarea del usuario, hasta la generación de casos de uso preliminares para el desarrollo de software. Todo este proceso es asistido por herramientas, tanto existentes como la propuesta en este trabajo, lo que le brinda eficacia y calidad al mismo. Por todo lo anterior consideramos que el caso de uso resultó exitoso y nos permite afirmar que la herramienta cumple con los objetivos de asistencia al analista para los que fue diseñada.

7 Conclusiones

En la herramienta que se propone en el presente trabajo se parte de los siguientes objetivos:

La presente es parte de los esfuerzos que se realizan en el área de la ingeniería de software por lograr un objetivo general, lograr mejorar la calidad de la captura de requerimientos en un desarrollo de software. Este objetivo puede y debe ser atacado desde distintos ángulos, nuestro trabajo pretende atacar uno de ellos.

La manera en que este trabajo pretende aportar es asistiendo a la integración de técnicas de "Análisis de la Tarea" a los procesos de desarrollo actuales; en particular a los procesos realizados mediante el "Proceso Unificado de Desarrollo de Software".

Uno de los problemas por lo que la integración antes mencionada no se ha dado es la cantidad de tiempo que implica debido a la falta de herramientas que soporten este proceso. El trabajo que aquí se desarrolló esta enfocado en lograr que esta integración pueda realizarse con mayor eficiencia, esto mediante el desarrollo de una herramienta que asista al equipo de desarrollo en la obtención de una serie de casos de uso preliminares a partir de un análisis de la tarea.

Para lograr el objetivo anterior, se estudió el proceso que actualmente realiza un desarrollador cuando desea obtener casos de uso a partir de un análisis de la tarea. De este estudio se concluyó que era necesario que la herramienta asistiera en la creación de un "árbol de la tarea modificada". Este árbol se enfoca ya a la tarea del sistema y es obtenido partiendo de un análisis de la tarea del usuario. A partir de este árbol podrán ser obtenidos los casos de uso preliminares.

Por la manera metódica y asistida por una herramienta en que se realizaría el proceso, consideramos que los casos de uso obtenidos así tendrán una mayor calidad, cumpliendo así con el objetivo de aportar en este sentido.

Otro de los problemas por los que el análisis de la tarea puede no ser integrado a los procesos de desarrollo de software actuales es debido a que las herramientas que surjan no se integren a las herramientas existentes, creando así un problema que los desarrolladores tendrían que solucionar manualmente (ej. capturando manualmente los resultados de una herramienta en otra). Este problema también es atacado en el presente trabajo mediante la exportación de los casos de uso obtenidos al lenguaje XMI (sub lenguaje de XML para la especificación de UML). Esto es de fundamental importancia ya que la tendencia actual de las herramientas es la de plasmar sus resultados en XML y en particular las herramientas de modelado UML guardarán y leerán la información en XMI, de tal suerte que la información que genera la presente herramienta será integrada a las nuevas herramientas con mayor facilidad.

De las herramientas líderes en el mercado actualmente se tiene a Rational Rose. Esta herramienta cuenta ya con la posibilidad de leer XMI (a través de un add-in desarrollado

por Unysis) por lo que el archivo generado por nuestra herramienta puede ser leído desde Rational. Esto último, aunado con la posibilidad de leer archivos generados por la herramienta IMAD* (que permite capturar árboles de la tarea del usuario) permite integrar un proceso completo, desde la captura del árbol de la tarea del usuario hasta la conclusión del desarrollo del software, soportado por herramientas.

Para cumplir con todos estos objetivos la herramienta asiste en el análisis del árbol de la tarea del usuario y en el modelado del árbol de la tarea modificado.

La herramienta fue desarrollada conforme al ciclo de desarrollo de cualquier software: Primero, se tuvo una fase de planteamiento del problema, en esta fase se investigó sobre la manera en que se ayudaría de mejor manera al usuario. Se contemplaron múltiples posibilidades para múltiples aspectos de la herramienta.

Para la elección de la plataforma se tomó en cuenta la portabilidad que provee Java sobre cualquier otra plataforma gracias a su código multiplataforma. Además de esto, se tomó en cuenta la existencia de la herramienta ILOG JViews, que permitiría facilitar la diagramación de árboles de la tarea y que se encontraba disponible en JAVA.

Otra razón por la que se decidió desarrollar en este lenguaje fue la escalabilidad que tendría la aplicación en caso de que en posteriores trabajos se decidiera hacer la herramienta Web.

Otro aspecto que se evaluó es el punto del que se partiría en el análisis de la tarea. Para esto la decisión fue definitivamente a favor de tomar el árbol de la tarea que genera la herramienta IMAD* porque, por un lado, se tenía el acceso al diseñador mismo de la herramienta y por otro lado, porque se utilizaba el modelo de la tarea MAD* mismo que consideramos el más completo para la captura de información de la tarea del usuario.

La salida que tendría el programa fue un punto importante en el diseño de la herramienta. Se discutieron opciones como la de obtener diagramas UML de actividades o de secuencia ya que la información para ello también se encuentra en el modelo MAD*. Sin embargo, se determinó que los casos de uso eran el elemento de mayor utilidad. Esto se decidió considerando que son precisamente los casos de uso el elemento mediante el que se expresan los requerimientos en un proceso de desarrollo actual y, dado el objetivo de mejorar esta etapa del proceso, se decidió asistir en su obtención.

También fue importante la manera en que se obtendrían los casos de uso mencionados a partir del "árbol de la tarea modificado" que modelaría el analista. Para ello se consideraron las distintas posibles traducciones que plantea Kolkovsky [5]. De ellas, se consideró que la "basada en el árbol de la tarea" era la adecuada para los propósitos de la herramienta debido a que la traducción es directa y deja al analista las decisiones de re organización que desee hacer.

Los métodos de obtención en torno a "objetivos", "acciones" u "objetos" implican tomar decisiones que corresponden al analista, mediante el método elegido estas decisiones pueden ser tomadas posteriormente en el programa en que se continué el desarrollo del sistema.

Por otro lado, en caso de definir un método de los anteriores, la traducción de todo el árbol se haría con base en él; mediante la traducción propuesta, en cambio, el analista puede organizar distintas partes del árbol en torno a distintos criterios dentro de la herramienta de modelado que utilice en el resto del desarrollo.

Otro aspecto de diseño importante fueron las herramientas que requeriría el analista para obtener la información que deseaba del *árbol de la tarea del usuario* y para realizar el modelado del *árbol de la tarea modificado*. Para esto se partió de las solicitudes de algunos analistas de la tarea y de un análisis de la tarea realizado sobre uno de ellos.

De esto último obtuvimos que las siguientes funciones eran necesarias:

- Búsqueda sobre el árbol de la tarea con base en los siguientes atributos de la tarea:
    - Trabajador que la realiza
    - Prioridad
    - Objetos utilizados en las condiciones
    - Interruptibilidad
    - Modo en que se realiza
- Marcado de las tareas
- Transferencia de tareas, y grupos de tareas
- Creación de nuevas tareas en el árbol modificado
- Reorganización de las tareas en el árbol modificado

Finalmente se trató el asunto de la manera en que se debía obtener el resultado de la herramienta, para esto, se pensó en obtener un archivo que describiera los casos de uso en un formato propietario de la herramienta. Esto tenía el evidente problema de que si alguien quería usar este resultado en un proceso de desarrollo actual o simplemente imprimir un diagrama que los mostrara, necesitaría capturar estos casos de uso en otra herramienta. Por otro lado, se pensó en lograr un diagrama de casos de uso impreso, esto tenía la ventaja de que se generaría un documento final que podría ser usado como un artefacto del sistema directamente. El problema continuaba siendo que para usar los resultados dentro de un proceso de desarrollo de software aún tendría que capturar en otra herramienta los casos de uso generados por MAD2UML.

Por lo anterior se buscó la manera de obtener un resultado legible directamente en otra herramienta que condujese el resto del desarrollo de un software. En búsqueda de este objetivo se encontró el lenguaje XMI, derivado de XML para la especificación de UML. Por otro lado se encontró la manera de leer definiciones en este sublenguaje dentro de Rational Rose, herramienta líder para modelado de sistemas de software. Por lo anterior, se decidió encontrar la manera de generar los resultados de la herramienta en el formato XMI, de tal manera que Rational Rose pudiese leer los casos de uso generados por MAD2UML.

Una vez que se tuvo una herramienta que logró asistir en el proceso desde la lectura del resultado generado por IMAD* hasta la generación de un resultado legible desde Rational Rose (permitiendo además la exploración del árbol de la tarea del usuario y el modelado del árbol modificado); se procedió a probar su utilidad mediante su uso en un caso práctico de estudio.

El caso de uso fue elegido por tener un tamaño que permitiera mostrar con claridad las ventajas de la herramienta, es decir, que no fuera tan sencillo que pareciese mas sencillo analizar el árbol sin herramienta ni que fuese tan complejo que no se comprendiera y por lo tanto no se comprendiera la utilidad de la herramienta. Además de lo anterior se requería un ejemplo para el cual hubiese habido un análisis de la tarea al cual tomar como punto de partida. Debido a estas características, se tomó el caso de un sistema de contabilidad que se utiliza actualmente en el CCADET de la UNAM y para el cual Alejandro Lopez Kolkovsky realizó un análisis de la tarea en su trabajo de Tesis [5].

Como ya se comentó, existía ya un análisis de la tarea del caso de estudio y de éste análisis se tenía un árbol de la tarea del usuario resultante. Para ejemplificar el uso de nuestra herramienta fue necesario, previamente, capturar el árbol de la tarea en la herramienta IMAD*, una vez hecho esto se generó el archivo .MAD y a partir de el se comenzó a utilizar MAD2UML.

Ya en lo referente a la herramienta se logró leer el archivo .MAD y analizarlo mediante MAD2UML. Lo importante de este análisis fue que se mostró cómo las funciones de búsqueda y coloreado de MAD2UML permiten diseccionar el árbol de la tarea del usuario con base en las características que interesen al usuario.

Por otro lado, se mostró de que manera las herramientas de creación de tareas y ligas en el *árbol modificado*, junto con las posibilidades de transferencia de tareas desde el *árbol de la tarea del usuario*, permiten al analista modelar *el árbol modificado*.

Finalmente, se mostró la forma en que la herramienta genera casos de uso a partir del árbol de la tarea modificado obtenido y de que forma se forma un diagrama de casos de uso en Rational Rose con los casos de uso obtenidos en MAD2UML.

Trabajos Futuros

Con todo esto se muestra el proceso completo asistido por la herramienta así como los beneficios que brinda al analista de la tarea. Por otro lado, se identificaron algunas necesidades que persisten aún con la herramienta. Un ejemplo de ello consiste en la imposibilidad de obtener artefactos impresos del árbol de la tarea del usuario y de la tarea modificado.

Otro problema es que las fichas de la tarea son difíciles de capturar y tampoco es posible sacarlas del sistema de tal manera que la documentación de las mismas sigue siendo manual. Finalmente, es necesario homogeneizar las herramientas IMAD* y MAD2UML en una sola herramienta lo más portable posible, de tal forma que pudiese ser usado en todos los sistemas operativos y, de ser posible, en cualquier plataforma. Para ello, se sugiere re-

codificar IMAD* en el lenguaje Java y agregarle funcionalidades que permitan cubrir las deficiencias aquí citadas, es decir, que permitan imprimir los árboles de la tarea así como las fichas de usuario.

Otro aspecto en el que se puede trabajar con el propósito de asistir de mayor manera a la captura de requerimientos de los procesos de desarrollo de software es la obtención de diagramas de actividades o de secuencia que muestren la dinámica del sistema a partir del árbol de la tarea del usuario. Para esto, se sugiere realizar una herramienta que asista (como en el caso de MAD2UML) en el análisis del árbol de la tarea del usuario y la edición del árbol de la tarea modificado pero enfocada en las características dinámicas del árbol, de esta herramienta se deberían poder obtener diagramas de secuencia o de actividades preliminares que permitan tener una dinámica del sistema inicial basada en el estudio de la tarea del usuario.

8 Bibliografía

[1] Sommerville, I., *Ingeniería de Software.* 2ª Edición ed. 1988, México: Addison Wesley.
[2] Pressman, R., *Software engeeniering, A beginner's guide.* 1988, E.U.A.: Mc Graw Hill.
[3] I. Jacobson, G.B., J. Rumbaugh, *El proceso unificado de desarrollo de software.* 1a en español ed. 2000, Madrid: Pearson Educación S.A.
[4] I. Jacobson, G.B., J. Rumbaugh, *El Lenguaje Unificado de Modelado.* 1a en español ed. 1999, Madrid: Addison Wesley Iberoamérica.
[5] Kolkovsky, A.L., *Integración del Análisis Tarea-Usuario al Proceso Unificado.* 2003, México: UNAM, IIMAS.
[6] Pinzon, R.A., *Evaluación de interfaces de usuario en El Proceso Unificado.* 2002, México: UNAM, IIMAS.
[7] Institute, S.E., 2002, www.sei.cmu.edu/tsp/psp.html
[8] SEI, 2002, http://www.sei.cmu.edu/tsp/tsp.html
[9] 2002, http://www.extremeprogramming.org/index.html
[10] Rodriguez, F.G., *Spécification et impleméntation d'ALACIE, Atelier Logiciel d'Aide à la Conception d'Interfaces Ergonomiques.* 1998, Université de Paris-Sud: Paris. p. 266.
[11] Oktaba, H., *Introducción a Team Software Process SM (TSPi SM).* 2002, Dep. de Matemáticas, Fac. de Ciencias,UNAM.
[13] OMG, www.omg.com/mda
[14] IBM, http://www.rational.com/products/reqpro/index.jsp
[15] Microsoft, http://www.microsoft.com/net/
[16] Peter Johnson, S.W., P. Markopoulos, J. Pycock, ADEPT- Advanced Design Environment for Prototyping with Task Models. Proceedings of INTERCHI'93, Amsterdam, 24-29 Abril 1993.
[17] C.Wiecha, S.B., *Generating user interfaces: principles and use of ITS style rules.* Proceedings of UIST'90. 1990: ACM Press. 21-30.
[18] C.Wiecha, S.B., *ITS: A tool for Rapidly Developing Interactive Applications.* ACM Transactions on Information Systems. 1990: ACM Press.
[19] Tarby, J.-C. and M.-F. Barthet, http://www.cutsys.com/CHI97/Tarby.html
[20] Tarby, J.-C. and M.-F. Barthet, The DIANE+ Method. Computer-Aided Design of User Interfaces. 1996: Namur University Press.
[21] Myers, B., Mayo de 1996, http://www.cs.ucl.ac.uk/staff/J.Dowell/cf/myers.pdf
[22] Wilson, D., *Programming with MacApp.* Addison-Wesley, 1990.
[23] X Business Group, I., *Interface Development Technology.* 1994.
[24] Francois Bodart, e.a., 1994, http://citeseer.nj.nec.com/bodart94modelbased.html
[26] P. Markopoulos, J.P., S. Wilson, and P. Johnson., *Adept A task based design environment.* Proceedings of the 25th Hawaii International Conference on System Sciences. 1992: IEEE Computer Society Press.
[27] P. Szekely, P., Luo, and R. Neches, Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design. Proceedings SIGCHI'92., May 1992.

[28] Mori G., P.F., Santoro C., CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. IEEE Transactions on Software Engineering, August 2002.
[29] F. Bodart, A.H., J. Leheureux, I. Provot, B. Sacre, and J. Vanderdonckt, Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide. Design, Specification and Verificationof Interactive Systems. Vienna, 1995.
[30] Brinksma., T.B.a.E., Introduction to the ISO specification language LOTOS. Computer Network ISDN Systems, 1987.
[31] Boies., C.W.a.S., *Generating user interfaces: principles and use of ITS style rules.* Proceedings of UIST'90. October 1990: ACM Press.
[32] Schlungbaum., T.E.a.E., Modelling and Generation of Graphical User Interfaces in the TADEUS Approach. Designing, Specification and Verification of Interactive Systems, 1995.
[33] Humphrey, W.S., *Introduction to the Team Software Process.* Series in Software Engeniering. 2000, EUA: Addison-Wesley.

9 Apéndice A "Hoja de estilo del programa"

```
SDM {

graphLayout:"ilog.views.graphlayout.tree.IlvTreeLayout";
    LinkLayout : true;
    StyleSheet: true;
    Coloring: true;
    Interactor: true;
}

StyleSheet {
        styleSheets : @styleSheets;
}

Coloring {
        colorProperty : "foreground";
        hue: 0.0 ;
        saturation: 0.5;
        brightness: 0.0;
}

LinkLayout {
        class: ilog.views.sdm.renderer.graphlayout.IlvLinkLayoutRenderer:
        performingLayoutOnZoom: true;
}

GraphLayout {
   enabled       :"true";
   savingNodePositions :"true";
   globalLinkStyle : "ORTHOGONAL_LINKS";
   flowDirection : "NORTH";
}


link {
        curved     : "false" ;
        mode       : "MODE_UNICOLOR" ;
        endCap : "CAP_ROUND";
        lineJoin : "JOIN_ROUND";
        linkWidth : 10;
        borderWidth : 4;
        borderUpColor : "white";
        borderDownColor : "black";
        foreground : "pink";
        strokeWidth : 2 ;
}

node {
        class : "ilog.views.sdm.graphic.IlvGeneralNode";
```

```
        shapeType : "RoundRectangle";
        label : "@Etiqueta";
        fillStyle:"SOLID_COLOR";
    fillColor1 : "Brown";
}

link.transition {
        class : "ilog.views.sdm.graphic.IlvGeneralLink";
        mode : "MODE_GRADIENT";
        foreground : orange;
        oriented : true;
}

node[Transferida="true"]{
        strokeColor    : "black";
        strokeWidth    : 3;
}

node[ObjetivoFundamental="true"]{
        shapeType : "Diamond";
}

node[Color="Rojo"]{
        fillColor1 : "Red";
}

node[Color="Azul"]{
        fillColor1 : "Blue";
}

node[Color="Negro"]{
        fillColor1 : "Black";
}

node[Color="Verde"]{
        fillColor1 : "Green";
}

node[Color="Amarillo"]{
        fillColor1 : "Yellow";
}

node:selected {
        fillColor1 : white;
}

link:selected {
        foreground : red;
        mode : "MODE_NEON";
}
```