UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

HERRAMIENTA DE IMPLEMENTACIÓN DE PROCESO

TESIS

QUE PARA OBTENER EL TÍTULO DE INGENIERO EN COMPUTACIÓN

PRESENTA:

FRANCISCO ALFREDO ADAM DAJER

DIRECTOR DE TESIS: M. I. ADOLFO MILLÁN NÁJERA

México, DF. ..... Octubre, 2005

Índice

Índice de Diagramas y Tablas

Introducción

El desarrollo de software como cualquier actividad desarrollada en la industria requiere de ser cuantificable, susceptible de ser optimizada y sobre todo estimable. Cualquier organización que se dedique al desarrollo de software necesita saber la cantidad de tiempo que sus empleados laboran en cada proyecto en el que están trabajando para poder definir los costos de cada uno de ellos. De la misma manera, requiere de datos de proyectos pasados para poder estimar tiempos y costos de proyectos futuros. Otra gran necesidad, es la de darle seguimiento a los errores encontrados durante el desarrollo de los proyectos y ver que se completen con el menor costo posible.

Debido a que en el desarrollo de cualquier proyecto de software no trivial muchas veces es imposible anticipar la naturaleza de los problemas a encontrar, no digamos su tiempo de resolución, es imposible decir a ciencia cierta cualquier dato exacto acerca de las actividades de desarrollo antes de que éstas se lleven a cabo. La mejor solución hasta el momento, es la utilización de métodos estadísticos, pero para poder sacar estadísticas acerca de algo se requiere de datos mesurables. Este proyecto pretende crear una herramienta que facilite la obtención de estos datos.

El problema del seguimiento del esfuerzo de desarrollo de software ha sido ampliamente tratado por Watts S. Humprey en el Instituto de Ingeniería de Software ("Software Engineering Institute" o SEI) quien desarrollo un método para poder capturar la información pertinente y utilizar ésta para estimar desarrollos futuros. El método en cuestión fue llamado Proceso Personal de Software ("Personal Software Process" o PSP) y lo expone en su libro "A Discipline for Software Engineering". Este proceso como su nombre lo indica, es un proceso personal, es decir de un solo individuo, pero si utilizamos los mismos principios a nivel organizacional, se puede llegar a resultados incluso más exactos, ya que las muestras son mucho más amplias.

Para el desarrollo de esta herramienta, se va a seguir un proceso centrado en documentos ya que éste es el que mejor se ajusta a las características de madures en el desarrollo según lo expone el SEI en su Modelo de Madures de Capacidades. Para esto se van a desarrollar los siguientes documentos:

Establecimiento del Proyecto

Para el desarrollo estructurado de cualquier sistema de software, es necesario definir un ciclo de vida para el sistema o modificar alguno ya existente. Esta elección define muchas de las actividades principales en todo el proceso de desarrollo, pero antes de poder hacer esta elección se tienen que establecer ciertos puntos acerca del proyecto. Todas estas definiciones y elecciones iniciales van a ser englobadas en este documento

Requisitos de Usuario

Una de las fases más complicadas de cualquier proyecto, es la definición de los requisitos. La principal dificultad consiste usualmente en la comunicación entre el o los clientes y los desarrolladores, especialmente en que ambos grupos tengan claros los requisitos y lo

que éstos conllevan. Es también común que algunos de ellos ni siquiera sean conocidos cuando se da inicio al proyecto. Este documento pretende ser el repositorio de todos estos requisitos y en el curso de un proyecto es actualizado constantemente para reflejar lo mejor posible lo que el sistema desarrollado deberá de ser capaz de hacer.

Requisitos de Sistema

En el documento de Requisitos de Sistema se hace un análisis de los requisitos del sistema según las funcionalidades descritas en los requisitos de usuario. Se analizan las interfaces en términos de hardware, software y de su interacción con el usuario. Adicionalmente, se analizan los requisitos no funcionales, es decir, los de desempeño, facilidad de uso y de modificación, escalabilidad…

Arquitectura

La definición de la arquitectura de un proyecto de software es uno de los pasos mas críticos, ya que es aquí donde se definen los grandes bloques de software, su distribución en los recursos de hardware y la forma en la que se han de comunicar entre si. De esta arquitectura depende que sea asequible satisfacer los requisitos no funcionales.

Diseño

En el documento de diseño se hace un análisis detallado de cada uno de los componentes de software y se definen sus características. Si este diseño es llevado a cabo con la suficiente calidad, se pueden minimizar los problemas que se han de encontrar en la fase de implementación.

Para efectos de esta tesis, cada uno de los documentos previamente descritos va a integrar un capítulo. Adicionalmente, detalles muy específicos como el código fuente del producto final, se van a quedar relegados a un apéndice y solamente como muestra, ya que de otra forma ocuparían un gran espacio.

Capítulo 1. Establecimiento del Proyecto

Objetivo

El documento de Establecimiento del Proyecto es utilizado para definir algunos conceptos básicos acerca del mismo, tales como:

- La organización general del equipo responsable del proyecto
- Los procesos que van a ser utilizados en el proyecto
- Las herramientas que se van a requerir
- Planeación inicial del proyecto

Cabe señalar que todas las consideraciones mencionadas en los anteriores puntos son solamente estimaciones e ideas iniciales, ya que a estas alturas el proyecto no se ha estudiado o desarrollado lo suficiente para asegurar nada.

Alcance

Esta sección define ideas iniciales acerca del proyecto en general pero no abunda en ellas. Podemos decir por ejemplo, que cuando se habla de los procesos que van a ser utilizados, no se pretende que estos estén completamente definidos con todas las actividades y resultados necesarios. Se pretende que se enlisten y describan brevemente los conceptos mas amplios como en el caso del proceso, el ciclo de vida a usar, los procesos específicos para el funcionamiento del equipo de desarrollo, los procesos estándares que van a ser considerados (CMM, ISO, …), procesos para inspecciones y revisiones, etc.…

Referencias

Los documentos referenciados en la preparación del presente, o aquellos en los que se basa son:

- *Carnegie Mellon University. "A Discipline for Software Engineering" Watts S. Humprey. 1995 Addison-Wesley*
- *Carnegie Mellon University. "Introduction to the Team Software Process" Watts S. Humprey. 1999 Addison-Wesley*
- *"Extreme Programming Explained: Embrace the Change" Kent Beck. 2000 Addison-Wesley*
- *Rational Software Corporation."Software Project Management: A Unified Framework" Walker Royce 1998 Addison-Wesley*
- *Proceeding, IEEE WESCON "Managing the Development of Large Software Systems" 1970 Winston W Royce*
- *"A spiral model of software development and enhancement" Boehm, B. W.Computer, Vol.21, Iss.5, Mayo 1988*

- *"Evolutionary Delivery versus the Waterfall Model" Tom Gilb. ACM SIGSOFT, Vol. 10, Iss. 3 Julio 1985*
- *"Object-Oriented and Classical Software Engineering" Stephen R. Schach. 2002 McGraw-Hill*
- *"Agile Model Driven Development (AMDD)" Scott W. Ambler. 2004 The Official Agile Modeling (AM) Sitio Web http://www.agilemodeling.com*

1.1 Objetivo del Proyecto

El proyecto tiene como objetivo implementar una herramienta de registro de las actividades de desarrollo de software llamada "Herramienta de Implementación de Proceso" ("Process Enactment Tool" o PET). Otro objetivo fundamental es el de desarrollar software de alta calidad utilizando las mejores técnicas y procesos de la ingeniería de software actual.

1.2 Organización y Recursos del Proyecto

Para su organización el equipo de desarrollo usó los roles especificados en TSP. El equipo consistió de 5 desarrolladores los cales tuvieron que dividirse los siguientes papeles:

- Líder de Equipo
  Es encargado de mantener al equipo funcionando como una unidad, de asegurarse que las actividades programadas se terminen a tiempo, servir de enlace con los supervisores y de organizar las reuniones de equipo.
- Encargado de Desarrollo
  Se encarga de asegurarse que el proyecto llevado a cabo cumpla con todos los requerimientos de forma consistente durante todas las fases del desarrollo. Otra responsabilidad es la de que todas las habilidades de los miembros del equipo se apliquen de la manera más eficiente para bien del proyecto.
- Encargado de Planeación
  Es el que define el plan detallado de las actividades del equipo, de darle seguimiento a dicho plan y de hacer los reportes de avance semanal.
- Encargado de Calidad y del Proceso
  Encargado de asegurar que todos los miembros del equipo reporten y usen de manera precisa todas las mediciones pertinentes al proceso y de que todo esto este debidamente documentado. Otra responsabilidad que tiene es la de asegurarse que todos los documentos y el producto desarrollado tengan la calidad adecuada y de que se sigan todos los procesos fielmente. También, lidera las inspecciones y revisiones de los productos y documentos desarrollados. Finalmente, reporta los temas tratados y los resultados de todas las reuniones de equipo y mantiene al día la bitácora del equipo.
- Encargado del Soporte
  Es el encargado de asegurarse que el equipo tiene a su alcance todas las herramientas y métodos necesarios para el desarrollo y de todas las actividades de control de versión, tanto de las herramientas como de los documentos desarrollados. Relacionado con el punto anterior, es el encargado de aprobar cualquier cambio a los productos y documentos puestos sobre control de versión. Adicionalmente, es responsable de registrar y darle seguimiento a todos los problemas surgidos desde que se hace un reporte hasta que se solucionan. Finalmente, es encargado de procurar el máximo re-uso de los productos y documentos producidos.

Adicionalmente a las actividades propias de cada rol, todos los integrantes del equipo deben de ser desarrolladores, es decir, todos son encargados de elaborar los productos y documentos pertinentes.

1.3 Estrategia de Proceso

Ciclo de Vida

Debido a las muchas implicaciones que tiene la elección del ciclo de vida (o modelo de desarrollo) en la planeación y estructuración del proyecto, éste tiene que ser definido lo antes posible. En este caso, la elección se llevó a cabo en esta sección inicial del proyecto.

Se analizaron los siguientes ciclos de vida en relación a su utilidad para este proyecto: modelo en cascada, de prototipo rápido, evolutivo, ágil, espiral y finalmente el modelo incremental. A continuación una brevísima descripción de cada uno de ellos.

Modelo en Cascada

Modelo secuencial donde para iniciar cada fase se requiere que la anterior haya sido completada. El diagrama incluye las fases como las definió Winston Royce en 1970.

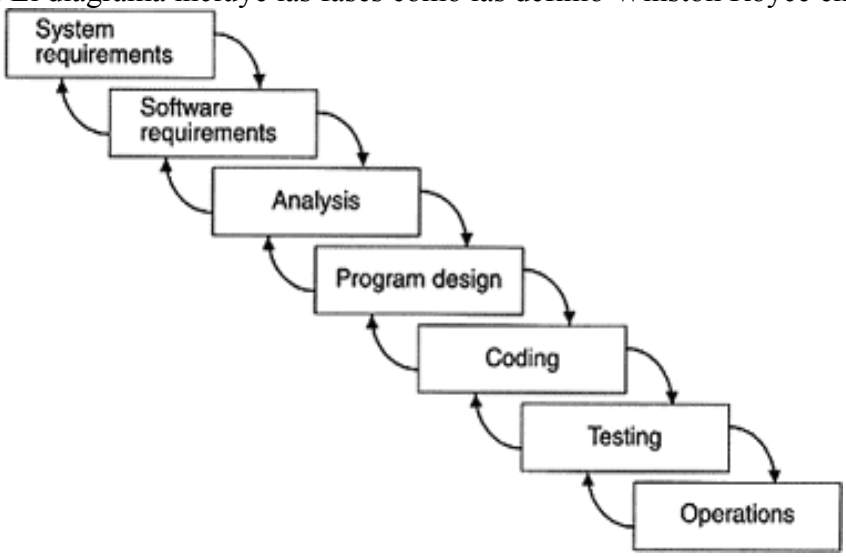

Diagrama 1 - Modelo en cascada[1]

Ventajas	Desventajas
Modelo más conocido y familiar para la mayoría de los programadores	Al tener que terminar con una fase antes de iniciar la siguiente, se restringe el trabajo paralelo que puede ser útil para evitar retrasos

[1] Tal como lo definió Winston Royce en su artículo "Managing the development of large software systems: concepts and techniques" y posteriormente actualizado por su hijo Walker Royce en su libro "Software Project Management: A Unified Framework"

Sus fases corresponden perfectamente con la forma en que los documentos fueron evaluados	Hace muy difícil la incorporación de los cambios solicitados por los supervisores una ves que los documentos de cada fase son terminados
	Si los recursos son limitados y específicamente hay una fecha límite para la entrega, se corre el riesgo que al final no se haya alcanzada la fase de implementación con lo que el equipo puede quedarse solamente con documentos y ningún producto
	Es un proceso que requiere mucho esfuerzo en si mismo, es decir, una proporción importante del tiempo de desarrollo es dedicada a actividades propias del manejo del proceso. Esta última cualidad normalmente es referida como el peso de un proceso, en este caso es un proceso pesado

Tabla 1 - Análisis del modelo en cascada

Modelo Espiral

En el modelo espiral, no hay fases como tales que deban de ser completadas en cierto orden, mas bien hay una progresión de actividades que se van repitiendo de manera cíclica. Boehm define el modelo espiral de la siguiente forma "El modelo refleja el concepto subyacente de que cada ciclo involucra una progresión que trata la misma secuencia de pasos, para cada porción del producto y para cada uno de sus niveles de elaboración, desde un concepto global de documentos operativos hasta la codificación de cada programa individual".

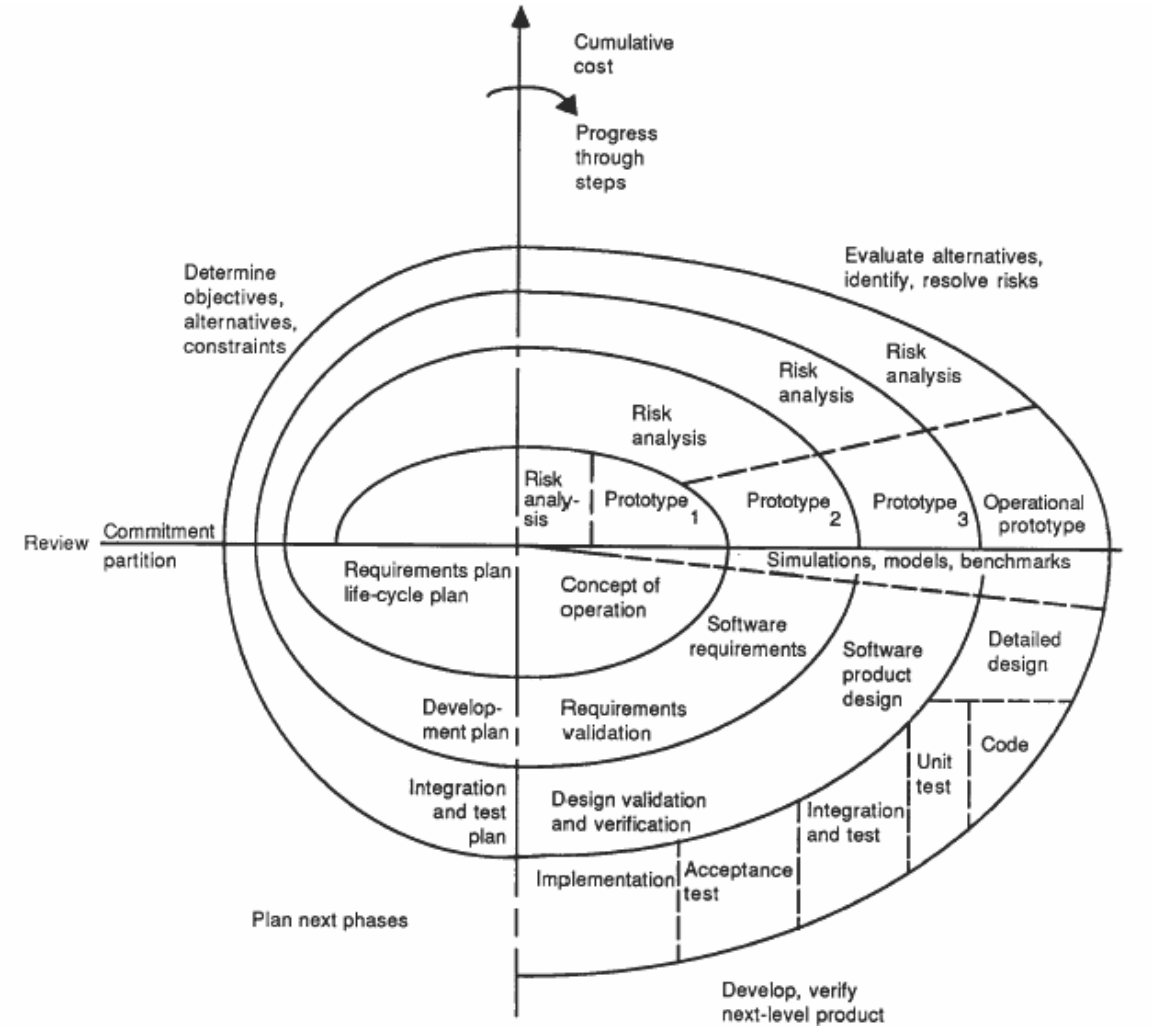

Diagrama 2 - Modelo en espiral[2]

Ventajas	Desventajas
Permite que el proyecto se mantenga haciendo espirales en cada problema hasta que la solución necesaria sea encontrada	Si los criterios de entrada y de salida de cada espiral no están bien definidos, se puede perder mucho tiempo en cada una de ellas
	Si no se cuenta con experiencia en el tipo de problemas y los riesgos que éstos entrañan, es difícil que los criterios de entrada y salida sean correctamente establecidos

Tabla 2 - Análisis del modelo en espiral

[2] "A spiral model of software development and enhancement" Boehm, B. W.Computer, Vol.21, Iss.5, Mayo 1988

Modelo Evolutivo

En este modelo, se desarrolla un prototipo al inicio, y éste es constantemente evolucionado hasta conseguir que contenga toda la funcionalidad del producto final. En este modelo, se planea solamente para el siguiente horizonte, es decir hasta donde realmente se tiene conocido. El resultado de este modelo es que se tiene una serie de micro-incrementos. El modelo fue definido y sigue siendo evolucionado por Tom Gilb. A continuación un diagrama de uno de sus primeros artículos al respecto:

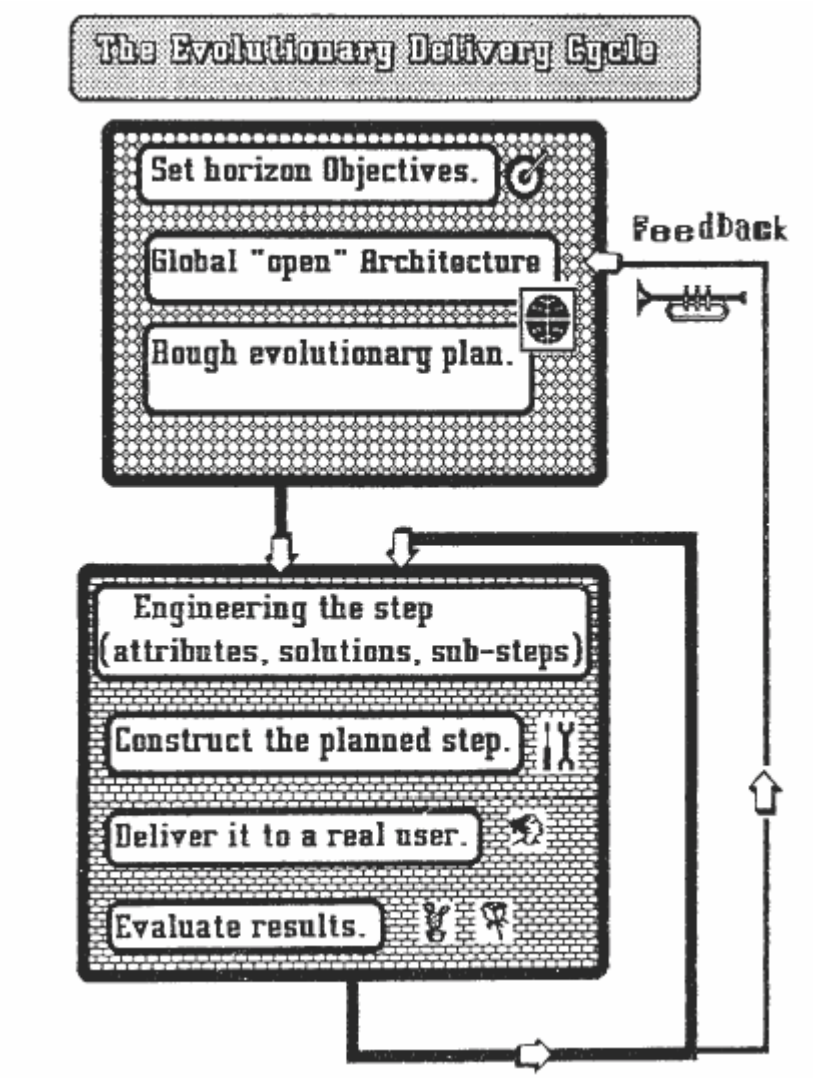

Diagrama 3 - Modelo evolutivo[3]

Ventajas	Desventajas
Siempre existe un sistema que en caso de cumplir con un conjunto suficientemente	Se corre el riesgo de que en sistemas complejos, la fase de limpieza o

[3] "Evolutionary Delivery versus the Waterfall Model" Tom Gilb. ACM SIGSOFT, Vol. 10, Iss. 3 Julio 1985

grande de los criterios de aceptación puede servir de producto final	reestructuración del código se vuelva extremadamente laboriosa o que requiera tanto tiempo que sea más fácil rehacer el sistema con lo que el modelo pierde sentido
Es un proceso muy ligero	

Tabla 3 - Análisis del modelo evolutivo

Modelo de Prototipo Rápido

Modelo en el que al igual que en el evolutivo se genera un prototipo, pero a diferencia de éste, el prototipo solamente es utilizado para encontrar los requerimientos del sistema. Una ves que se ha terminado esta fase, el prototipo es desechado.

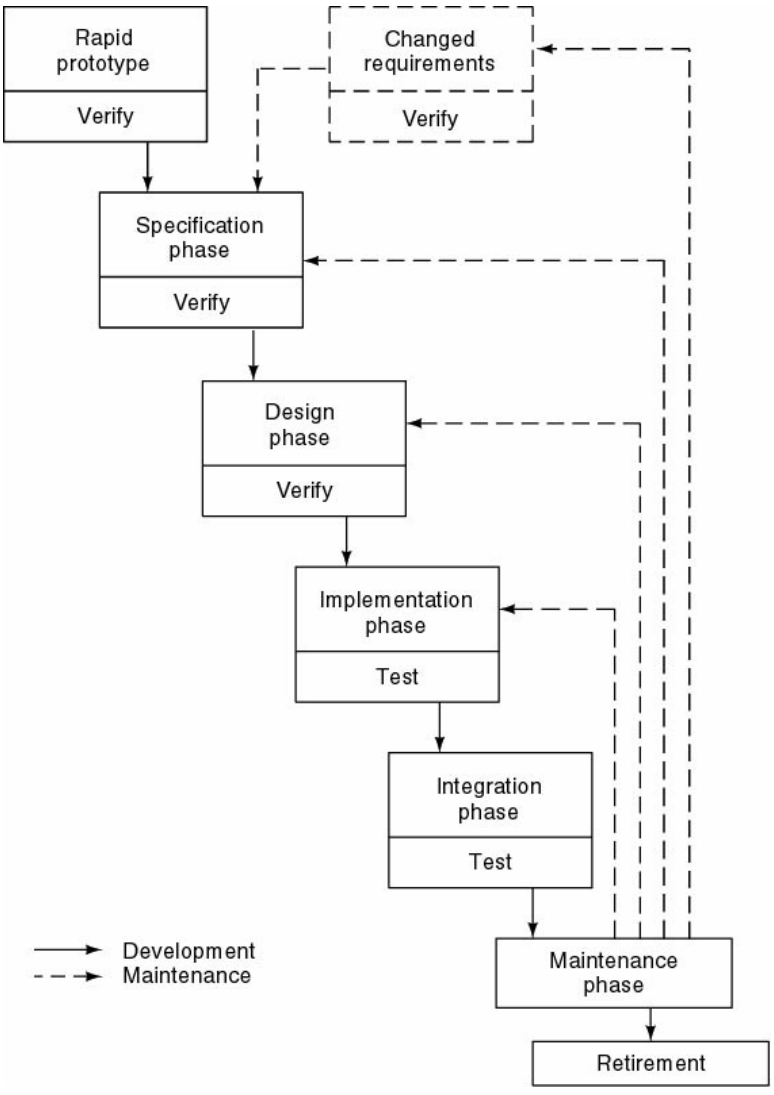

Diagrama 4 - Modelo de prototipo rápido[4]

[4] "Object-Oriented and Classical Software Engineering" Stephen R. Schach. 2002 McGraw-Hill

Ventajas	Desventajas
El prototipo conduce a fases de análisis y diseño muy precisas	En un proyecto con el tiempo limitado, se corre el riesgo que el esfuerzo empleado y luego desechado en el prototipo sea demasiado
Permite solucionar los problemas inherentes a la tecnología a aplicar desde muy temprano en el desarrollo	Es un proceso pesado

Tabla 4 - Análisis del modelo de prototipo rápido

Modelo Ágil

Modelo que utiliza todos los principios, valores y prácticas de la programación extrema ("Extreme Programming" o XP), tal como la define Kent Beck en su libro "Extreme Programming Explained". Una de lás principales prácticas que Beck propone, es que la programación sea el centro del desarrollo. Propone ciclos cortos de desarrollo siguiendo un modelo evolutivo pero con la adición de ciertas prácticas como las revisiones y pruebas de integración y desempeño automatizadas que deben de ser practicadas constantemente.

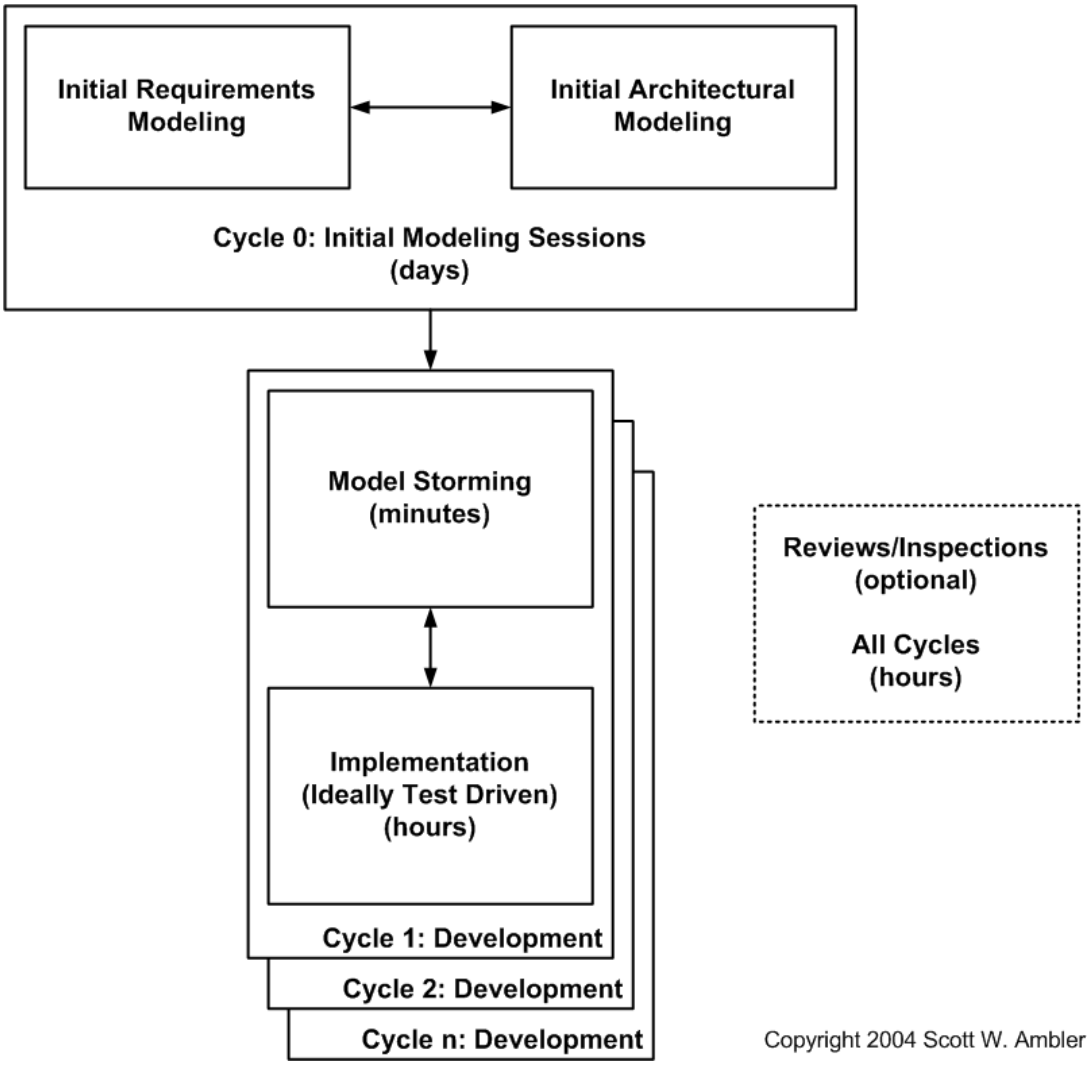

Diagrama 5 - Modelo ágil[5]

Ventajas	Desventajas
Es un proceso extremadamente ligero	Los conceptos de XP no son muy conocidos por la mayoría de los desarrolladores
Se acomoda muy bien a las preferencias de los desarrolladores ya que mantiene buena parte del esfuerzo en la implementación	

Tabla 5 - Análisis del modelo ágil

[5] "Agile Model Driven Development (AMDD)" Scott W. Ambler. 2004 The Official Agile Modeling (AM) Sitio Web http://www.agilemodeling.com

Modelo Incremental

Modelo en el que se define un conjunto mínimo de requerimientos a satisfacer para que el sistema tenga sentido. Se desarrolla este conjunto de requerimientos usando un modelo en cascada. Los requerimientos restantes son programados en grupos para que por medio de la aplicación sucesiva de ciclos de desarrollo en cascada, permiten que el sistema evolucione hasta completar su desarrollo.

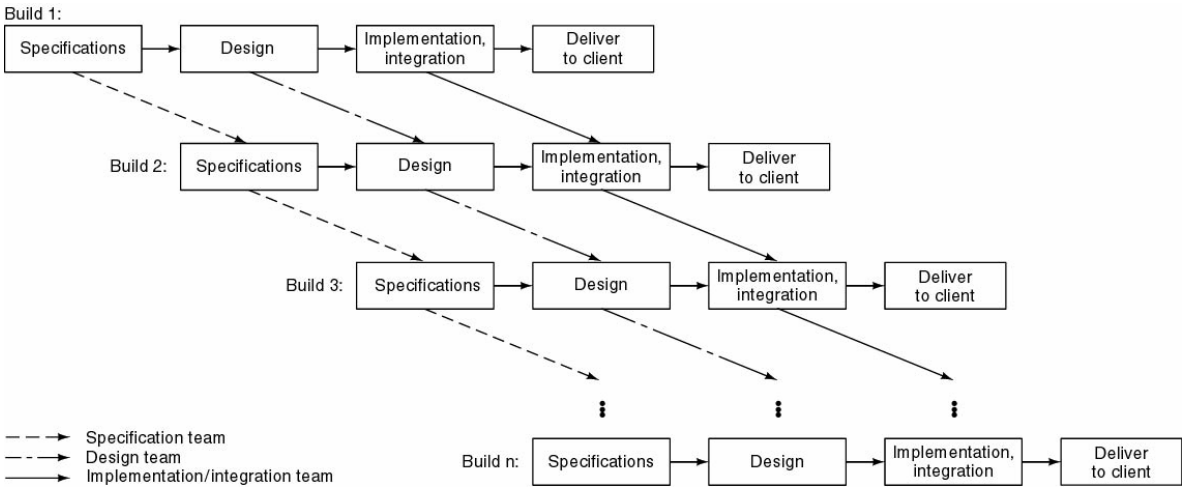

Diagrama 6 - Modelo incremental[6]

Ventajas	Desventajas
Es relativamente conocido al ser una aplicación sucesiva de modelos en cascada	Se corre el riesgo de añadir demasiados requerimientos al incremento inicial con lo que se puede terminar con un modelo en cascada
Permite presentar los resultados de cada incremento a los supervisores y clientes potenciales asegurando así la mayor utilidad del sistema final	Puede ser complicada la elección de que requerimientos incluir en cada incremento
Ya que los requerimientos indispensables son desarrollados al inicio, en caso de requerirse, los subsecuentes incrementos pueden recortarse para acomodar problemas de tiempo	Se puede perder mucho tiempo en la revisión y ampliación de los documentos pertenecientes a las mismas fases pero en incrementos anteriores
	Es un proceso pesado

Tabla 6 - Análisis del modelo incremental

[6] Rational Software Corporation."Software Project Management: A Unified Framework" Walker Royce 1998 Addison-Wesley

Elección del Modelo

Para la elección del modelo se consideraron todas las ventajas y desventajas antes expuestas en relación con el tipo de proyecto a desarrollar y los recursos disponibles. La elección fue realizada en acuerdo con los supervisores del equipo y fue la de modificar el modelo incremental de la siguiente forma:

- Primer Incremento. Hacer las fases de análisis de requerimientos y diseño como si se tratara del modelo en cascada
- Del segundo incremento en adelante no se hace un análisis de requerimientos.
- Del segundo incremento en adelante la fase de diseño es muy reducida y solamente se usa para incorporar los cambios resultantes de la evaluación/presentación al final del incremento anterior

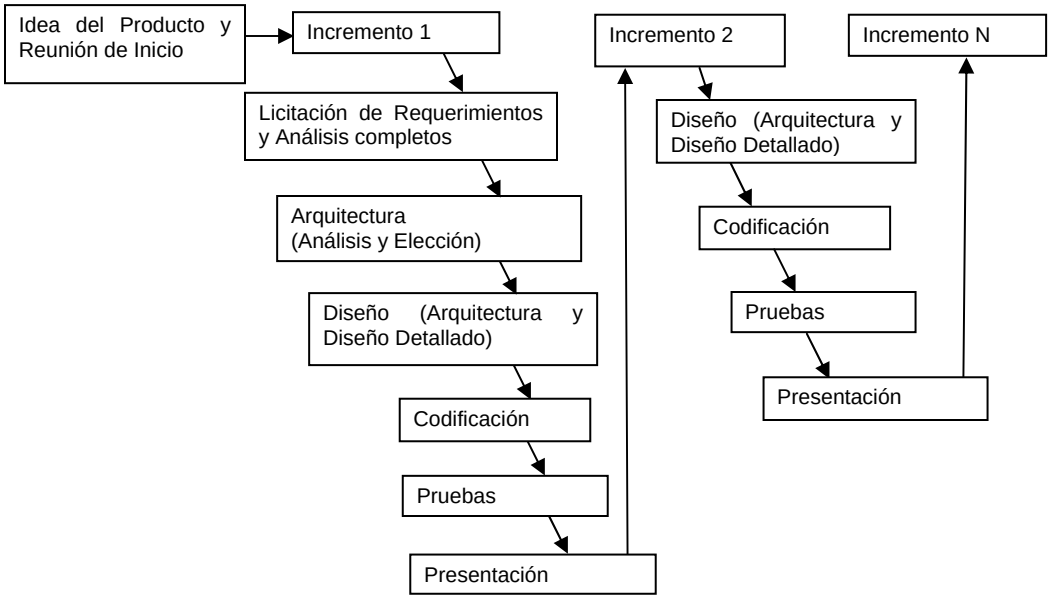

Diagrama 7 - Modelo incremental modificado

El modelo elegido tuvo como ventajas principales las siguientes:

- Al ser más largos los incrementos, se da la flexibilidad de acomodar curvas de aprendizaje mayores, necesidad fundamental en un equipo inexperto
- Al tener una fase de análisis de requerimientos completa al inicio, permite que éstos sean divididos según prioridades en incrementos o entregas más eficientemente
- Dado que se planeó una presentación al final de cada incremento, se pueden incorporar las observaciones que se obtengan de ésta para los subsecuentes incrementos

Capítulo 2. Requisitos de Usuario

Objetivo

El documento de Requisitos de Usuario (URD por sus siglas en inglés) sirve como establecimiento formal de los requisitos del sistema desde el punto de vista tanto del cliente como de los desarrolladores.

Alcance

En este documento se definen los requisitos tanto funcionales como no funcionales (de desempeño, facilidad de mantenimiento, escalabilidad,…) que el sistema debe de tener. Tanto la estrategia como los detalles de la implementación de todos estos requisitos se dejan para documentos subsecuentes.

Referencias

Los documentos referenciados en la preparación del presente, o aquellos en los que se basa son:

Carnegie Mellon University. "A Discipline for Software Engineering" Watts S. Humprey. 1995 Addison-Wesley

2.1 Requisitos Generales

El Sistema en Contexto

A grandes rasgos, el sistema en cuestión es una herramienta que permite a los supervisores de una organización que desarrolla software, monitorear el desempeño actual e histórico de la organización, así como ayuda en la estimación de proyectos futuros. Estas funciones son realizadas en modo cercano a tiempo real para múltiples usuarios simultaneos.

El sistema utiliza una base de datos relacional (RDBMS por sus siglas en inglés) para almacenar todos los datos tales como: usuarios, su desempeño, proyectos en curso, proyectos pasados y ciclos de vida para proyectos

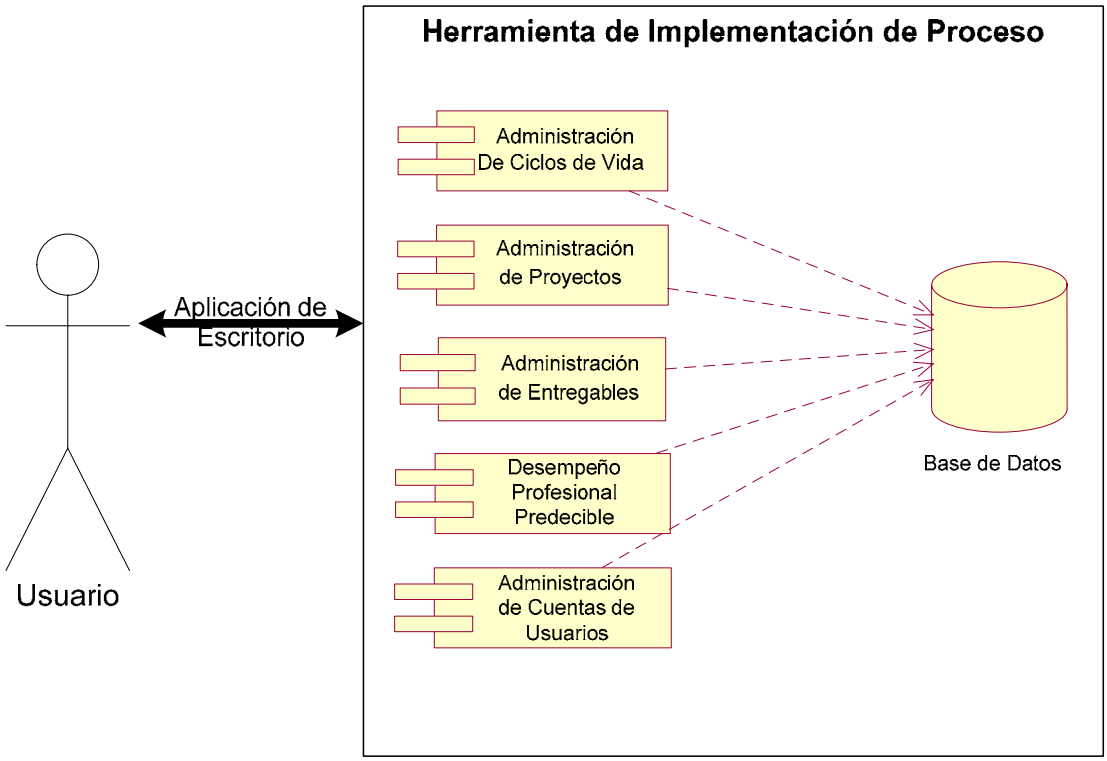

Diagrama 8 - Descripción general del sistema

El sistema es diseñado como una aplicación de escritorio que consta de 5 grupos de funciones que almacenan su información en una base de datos. Los grupos son los siguientes:

- Administración de Ciclos de Vida
  Administración de los procesos generales utilizados para el desarrollo y mantenimiento de software
- Administración de Proyectos
  Administración de los proyectos que utilizan los ciclos de vida del sistema

- Administración de Entregables
  Administración de los archivos y documentos que se generan como parte de los proyectos
- Desempeño Profesional Predecible
  Administración de las mediciones y datos recolectados durante el desempeño individual de los desarrolladores
- Administración de Usuarios
  Administración de los usuarios del sistema

Usuario Objetivo

El sistema fue diseñado como herramienta para el desarrollo de software, es por esto que los usuarios objetivos son desarrolladores de software y sus supervisores.

Se hicieron las siguientes suposiciones:

- Los supervisores administran los ciclos de vida
- Los supervisores tienen bajo su responsabilidad la planeación y el monitoreo de los proyectos
- Los supervisores asignan desarrolladores y estiman las diferentes actividades
- Los desarrolladores registran defectos y tiempo dedicado para cada actividad a la que son asignados
- Los desarrolladores pueden ver reportes y status de los entregables a los que son asignados
- Los desarrolladores registran el tamaño real de cada entregable y pueden ver la comparación de tamaño planeado contra el tamaño real de cada entregable
- Todos los usuarios utilizan el sistema de buena fe

Restricciones

- El equipo de desarrollo estuvo limitado a los integrantes iniciales, se inició con 7 desarrolladores y se terminó con 5
- El sistema tubo que estar terminado para en un periodo de 6 meses

Requisitos Globales

El sistema debe de procesar los siguientes tipos de errores de forma diferente:

- Errores de Usuario
  El sistema despliega un mensaje de error con una explicación de éste
  El sistema permite al usuario corregir el error sin tener que reiniciar la operación
- Errores de Sistema
  El sistema despliega un mensaje con los detalles del error

Entidades que va a Registrar el Sistema

Entidad	Descripción
Usuarios	Los usuarios registrados en el sistema, tomando en cuenta su nombre completo, un nombre de usuario (que debe ser único) y una contraseña
Unidades de Medición	Unidades de medición de tamaño para los Entregables
Ciclos de Vida	Los ciclos de vida en uso y/o aprobados por la organización que usa el sistema
Fases de Ciclo de Vida	Las fases de desarrollo de cada ciclo de vida
Entregables Prototipo	Documentos y/o archivos de código que se generan forzosamente en cada una de las fases del ciclo de vida
Proyectos	Los proyectos que se desarrollan según algún ciclo de vida registrado
Fases de Proyecto	Las fases del proyecto según el ciclo de vida en uso para el proyecto. Son una copia de las fases de ciclo de vida
Entregables de Proyecto	Los entregables de proyecto son copias de los entregables prototipos, pero que también incluyen la estimación de esfuerzo por parte del supervisor, así como la unidad de medición. Estos entregables pueden ser asignados a cualquier desarrollador
Entregables Ad hoc	Entregables similares a los de proyecto pero que no dependen de un entregable prototipo. El uso de estos es para casos donde se requieren otros entregables que no estaban previstos o que son específicos al proyecto en cuestión. Estos entregables no son usados para mantener registros históricos
Pasos de Proceso	Son las fases de desarrollo de cada entregable y son usadas por los desarrolladores para subdividir las tareas de cada entregable y así llevar un mejor registro de sus actividades para poder estimarlas mejor. Los pasos de proceso son a los entregables lo que las fases de proyecto son al proyecto.
Lista de Tareas que Hacer	Lista de entregables asignados a cada usuario
Lista de Tareas Completadas	Lista de entregables ya completados por cada desarrollador
Estimación de Esfuerzo	Registro de las estimaciones de esfuerzo y tamaño para los entregables
Registro de Esfuerzo	Registro del esfuerzo dedicado y el tamaño final de los entregables

Tabla 7 - Entidades del sistema

Consideraciones sobre las operaciones que el sistema va a permitir:

Administración de Ciclos de Vida

1. Un ciclo de vida puede ser modificado o eliminado solamente si no tiene proyectos asociados
2. A un ciclo de vida se le pueden añadir, modificar o eliminar fases solamente en el caso de que no haya proyectos relacionados con éste

Administración de Proyectos y Entregables

1. Un proyecto puede ser modificado o eliminado solamente cuando ninguno de sus entregables tienen registrado tiempo dedicado por algún desarrollador
2. Los entregables de proyecto pueden ser modificados solamente cuando no tienen registrado tiempo dedicado por algún desarrollador
3. Un entregable de proyecto puede ser asignado a un desarrollador solamente si no ha sido asignado a algún otro desarrollador
4. Un entregable de proyecto puede ser asignado solamente si existe tiempo para elaborarlo entre la fecha designada como inicio y la fecha de conclusión
5. Los pasos de desarrollo de un entregable pueden ser modificados o eliminados solamente si no se ha registrado tiempo dedicado a el por el desarrollador a cargo

Desempeño Profesional Predecible

1. La tarea de reflexión de cualquier tarea asignada va a ser realizada antes del envío de ésta como terminada

Administración de Usuarios

1. El sistema sólo permite el acceso a los usuarios mediante el ingreso de un nombre de usuario y una contraseña
2. Sólo los usuarios registrados como administradores pueden crear, modificar o eliminar cuentas de usuario
3. Las funciones reservadas para usuarios registrados como administradores son:
    i. Ver, crear, modificar y eliminar ciclos de vida
    ii. Ver, crear, modificar y eliminar fases para cada ciclo de vida
    iii. Ver, crear, modificar y eliminar entregables prototipo para cada fase de los ciclos de vida
    iv. Ver, crear, modificar y eliminar pasos del proceso para cada entregable prototipo
    v. Ver, crear, modificar y eliminar proyectos
    vi. Ver las fases del proyecto
    vii. Ver, crear, modificar y eliminar entregables ad hoc a este proyecto en específico
    viii. Ver, crear, modificar y eliminar estimaciones de tiempo requerido y tamaño para cada entregable de proyecto
4. Las funciones abiertas para los usuarios registrados como desarrolladores son:
    i. Ver su lista de tareas por hacer
    ii. Ver los pasos del proceso para realizar cada tarea y hacer sus estimaciones
    iii. Para cada paso del punto “ii”, empezar y detener el cronómetro
    iv. Para cada paso del punto “ii”, agregar el tiempo que fue interrumpido
    v. Para cada paso del punto “ii”, enviar los datos del tiempo dedicado
    vi. Al terminar todos los pasos, registrar el tamaño final del Entregable

Plan de Entregas

Para acomodar el ciclo de vida elegido (incremental), se negoció con los supervisores del proyecto el siguiente plan de entregas:

1er INCREMENTO

Núcleo de PET

- Rediseño de la Interfaz Gráfica de Usuario del prototipo
- Re-implementación de las siguientes secciones
    - Administración de Ciclos de Vida
    - Administración de Proyectos
    - Administración de Entregables

Desempeño Profesional Predecible

- Planeación
- Lista de Tareas por hacer
- Lista de Tareas Completadas
- Recopilación de Datos Históricos
- Pasos de Proceso para cada Entregable
- Registro de Esfuerzo
- Reflexión sobre las tareas realizadas

Administración de Usuarios

- Acceso personalizado y autentificado
- Administración de Cuentas de Usuarios

2° INCREMENTO

Núcleo de PET

- Asignación de un entregable a múltiples desarrolladores
- Uso de métodos estadísticos tales como regresión lineal para la estimación
- Posibilidad de adjuntar archivos de plantillas y resultados a los entregables asignados y completados respectivamente

Desempeño Profesional Predecible

- Registro y Seguimiento de Errores
- Modificación de los pasos de proceso para Entregables individuales por los desarrolladores asignados a éste
- Creación de tareas no forzosamente relacionadas con entregables directamente por los desarrolladores

Administración de Usuarios

- Privilegios de Acceso por medio de Roles o Papeles
- Organización de Equipos
- Equipos múltiples por usuario

Debido a las restricciones de tiempo y recursos, y después de hacer las estimaciones iniciales, se determinó, que el segundo incremento quedaría como opcional. Finalmente, en fases posteriores del desarrollo, se determinó que el segundo incremento sería imposible de realizar en el tiempo pactado.

2.2 Requisitos Detallados

La sección de Requisitos Detallados define los requisitos funcionales del sistema, es decir, que debe de permitir hacer el sistema. Finalmente, en esta misma sección, los requisitos técnicos son especificados. Estos últimos, definen a tanto detalle como se requiera por parte de quien haya encargado el desarrollo, los requisitos no funcionales tales como: la plataforma de uso y los atributos cualitativos o metas de desempeño.

Para definir los requisitos funcionales del sistema, en este caso fueron usados los siguientes medios:

- Texto que describe a grandes rasgos la funcionalidad objetivo por grupos relacionados lógicamente
- Diagramas conceptuales, que asemejan a un diagrama de clases, pero que no necesariamente representan clases u objetos
- Tablas resúmenes de los Casos de Uso con pequeñas descripciones de que se tratan
- Una tabla por cada caso de uso que define tanto como sea necesario el caso de uso. Estas tablas son usadas como referencia a lo largo del desarrollo del sistema, ya que definen exactamente como debe de comportarse el sistema desde el punto de vista de los usuarios u otros actores que interactúen con el

Debido a la naturaleza de este reporte, no se incluirán todas las tablas de caso de uso, ya que es información en extremo técnica, extensa y repetitiva. Sin embargo, se incluirá una tabla plantilla y algunos ejemplos específicos de casos de uso.

ID del Caso de Uso: Clave	**Nombre del Caso de Uso:** Nombre
Descripción:	Descripción breve del caso de uso
Condiciones Previas:	Condiciones del sistema necesarias para iniciar este caso de uso
Condiciones Posteriores:	Condiciones del sistema al finalizar este caso de uso
Frecuencia de Uso:	Frecuencia de uso esperada medida de forma general (baja, mediana, alta)
Curso Normal de los Eventos:	1. Paso1 2. Paso2 3. … 4. Paso n
Cursos Alternativos:	Secuencias alterna de pasos
Errores:	• Error 1 1. Acción tomada 1 2. Acción tomada 2 3. … 4. Acción tomada n • Error n
Incluye:	Referencia a los casos de usos incluidos
Requerimientos Asociados:	Lista de los requerimientos involucrados si es que este caso de uso refleje mas de uno

Tabla 8 - Modelo de tabla de caso de uso

Administración de Ciclos de Vida y Proyectos

La administración de ciclos de vida trata con todas las funciones referentes a la creación, modificación y borrado de los ciclos de vida y sus fases. La administración de proyectos trata con las funciones referentes a la creación de proyectos según algún ciclo de vida, la modificación de éstos y su eliminación. Cabe destacar que las fases que tiene un proyecto son copia de las del ciclo de vida elegido por lo que ya están definidas y no pueden ser cambiadas.

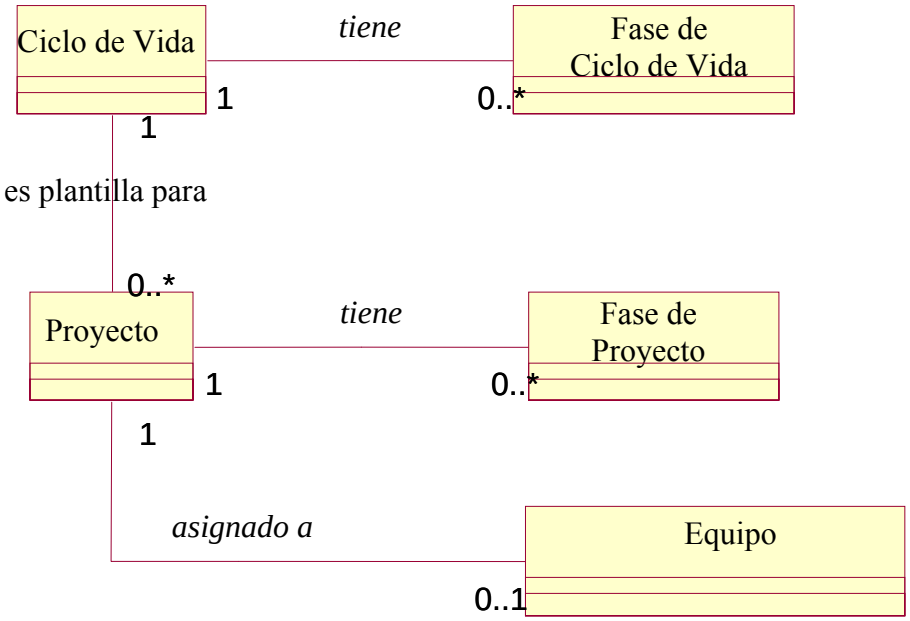

Diagrama 9 - Diagrama conceptual de la administración de los ciclos de vida y de proyectos

Características:

- Un Ciclo de Vida tiene cero o más Fases de Ciclo de Vida
- Un Ciclo de Vida sirve como plantilla para cero o más Proyectos
- Un Proyecto tiene cero o más Fases de Proyecto
- Las Fases de Proyecto son copias de las Fases de Ciclo de Vida

Errores de Usuario Verificados:

- Un Ciclo de Vida debe de tener un nombre único y no vacío
- Una Fase de Ciclo de Vida debe de tener un número de secuencia natural, único y diferente de cero
- Un Proyecto debe de tener un nombre único y no vació por cada ciclo de vida al cual esté asociado

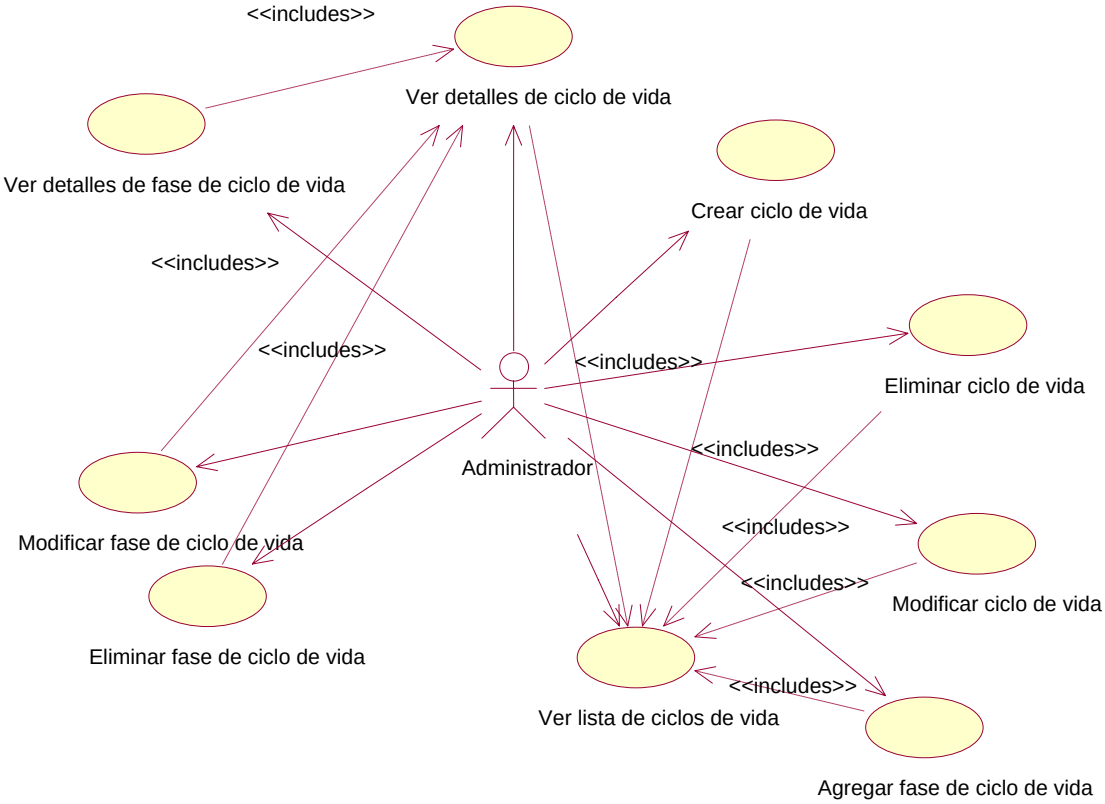

Diagrama 10 - Diagrama de casos de uso de la administración de ciclos de vida

	ID del Caso de Uso:	Nombre del Caso de Uso:
1.	UC-PET-LC-01	Crear Ciclo de Vida
2.	UC-PET-LC-02	Modificar Ciclo de Vida
3.	UC-PET-LC-03	Eliminar Ciclo de Vida
4.	UC-PET-LC-04	Agregar Fase de Ciclo de Vida
5.	UC-PET-LC-05	Modificar Fase de Ciclo de Vida
6.	UC-PET-LC-06	Eliminar Fase de Ciclo de Vida
7.	UC-PET-LC-07	Ver Detalles del Ciclo de Vida
8.	UC-PET-LC-08	Ver Detalles de la Fase de Ciclo de Vida
9.	UC-PET-LC-09	Ver Lista de Ciclos de Vida
10.	UC-PET-PR-01	Crear Proyecto
11.	UC-PET-PR-02	Modificar Proyecto
12.	UC-PET-PR-03	Eliminar Proyecto
13.	UC-PET-PR-04	Ver Detalles de Proyecto
14.	UC-PET-PR-05	Ver Detalles de Fase de Proyecto
15.	UC-PET-PR-06	Ver Lista de Proyectos

Tabla 9 - Casos de uso de la administración de ciclos de vida

ID del Caso de Uso: UC-PET-LC-01	**Nombre del Caso de Uso:** Crear Ciclo de Vida
Descripción:	Crea un ciclo de vida nuevo para ser usado en la planeación de proyectos
Condiciones Previas:	Ninguna
Condiciones Posteriores:	El nuevo ciclo de vida es guardado en el sistema
Frecuencia de Uso:	Baja
Curso Normal de los Eventos:	1. El usuario selecciona crear un nuevo ciclo de vida 2. El usuario escribe los datos del ciclo de vida: a. Nombre del ciclo de vida b. Descripción del ciclo de vida 3. El usuario envía la información 4. El sistema guarda la información 5. El sistema despliega un mensaje de éxito en la operación
Cursos Alternativos:	Ninguno
Errores:	• El usuario utiliza un nombre de ciclo de vida ya existente 1. El sistema despliega un mensaje indicando el error y regresa el control al usuario para que cambie el nombre
Incluye:	Refiérase al UC-PET-LC-09 Ver Lista de Ciclos de Vida
Requerimientos Asociados:	Ninguna

Tabla 10 - Ejemplo 1 de caso de uso: crear ciclo de vida

Administración de Entregables

	ID del Caso de Uso:	**Nombre del Caso de Uso:**
16.	UC-PET-DM-01	Crear Entregable Prototípico
17.	UC-PET-DM-02	Crear Entregable Ad-hoc
18.	UC-PET-DM-03	Crear Entregable de Proyecto
19.	UC-PET-DM-04	Modificar Entregable Prototípico
20.	UC-PET-DM-05	Modificar Entregable de Proyecto
21.	UC-PET-DM-06	Modificar Entregable Ad-hoc
22.	UC-PET-DM-07	Eliminar Entregable
23.	UC-PET-DM-08	Crear Paso de Proceso de Entregable
24.	UC-PET-DM-09	Modificar Paso de Proceso de Entregable
25.	UC-PET-DM-10	Eliminar Paso de Proceso de Entregable
26.	UC-PET-DM-11	Asignar Entregable
27.	UC-PET-DM-12	Ver Entregable Prototípico
28.	UC-PET-DM-13	Ver Entregable de Proyecto
29.	UC-PET-DM-14	Ver Entregable Ad-hoc

Tabla 11 – Casos de uso de la administración de entregables

ID del Caso de Uso: UC-PET-DM-08	**Nombre del Caso de Uso:** Crear Paso de Proceso de Entregable
Descripción:	Crean un paso de proceso para un entregable existente
Condiciones Previas:	El usuario ya ha seleccionado un entregable
Condiciones Posteriores:	Ninguna
Frecuencia de Uso:	Media
Curso Normal de los Eventos:	1. El usuario elige crear un nuevo paso de proceso 2. El usuario escribe los datos del paso de proceso: c. Nombre del paso de proceso d. Descripción del paso de proceso 3. El paso 2 es repetido para cada uno de los pasos de proceso que se requiera introducir 4. El usuario envía la información 5. El sistema guarda la información 6. El sistema despliega un mensaje de éxito en la operación
Cursos Alternativos:	Ninguno
Errores:	• El usuario utiliza un nombre de paso de proceso ya existente para este entregable 2. El sistema despliega un mensaje indicando el error y regresa el control al usuario para que cambia el nombre • El entregable ya tiene esfuerzo registrado 3. El sistema rechaza la operación
Incluye:	Refiérase al UC-PET-DM-01 Crear Entregable Prototípico Refiérase al UC-PET-DM-02 Crear Entregable Ad-hoc Refiérase al UC-PET-DM-03 Crear Entregable de Proyecto Refiérase al UC-PET-DM-04 Modificar Entregable Prototípico Refiérase al UC-PET-DM-05 Modificar Entregable de Proyecto Refiérase al UC-PET-DM-06 Modificar Entregable Ad-hoc
Requerimientos Asociados:	Ninguna

Tabla 12 - Ejemplo 2 de caso de uso: crear paso de proceso de entregable

Desempeño Profesional Predecible

	ID del Caso de Uso:	**Nombre del Caso de Uso:**
30.	UC-PET-PPP-01	Ver Lista de Tareas por Hacer
31.	UC-PET-PPP-02	Ver Lista de Tareas Completadas
32.	UC-PET-PPP-03	Ver Tarea Completada
33.	UC-PET-PPP-04	Reflexión acerca de la Tarea Completada
34.	UC-PET-PPP-05	Enviar Tarea por Hacer Completada
35.	UC-PET-PPP-06	Ver Tarea por Hacer
36.	UC-PET-PPP-07	Crear Paso de Proceso
37.	UC-PET-PPP-08	Modificar Paso de Proceso
38.	UC-PET-PPP-09	Eliminar Paso de Proceso

	ID del Caso de Uso:	Nombre del Caso de Uso:
39.	UC-PET-PPP-10	Ver Paso de Proceso
40.	UC-PET-PPP-11	Registro de Esfuerzo por Paso de Proceso
41.	UC-PET-PPP-12	Registra Defecto
42.	UC-PET-PPP-13	Modificar Defecto
43.	UC-PET-PPP-14	Ver Defecto
44.	UC-PET-PPP-15	Registro de Esfuerzo por Defecto

Tabla 13 - Casos de uso del modulo de desempeño profesional predecible

ID del Caso de Uso: UC-PET-PPP-11	**Nombre del Caso de Uso:** Registro de Esfuerzo por Paso de Proceso
Descripción:	El usuario registra el esfuerzo aplicado para un paso específico del proceso de una Tarea por Hacer
Condiciones Previas:	Se ha seleccionado un Paso de Proceso
Condiciones Posteriores:	El esfuerzo es registrado como aplicado a el paso seleccionado
Frecuencia de Uso:	Alta
Curso Normal de los Eventos:	7. El usuario selecciona la opción de registrar esfuerzo para un paso de proceso 8. El usuario registra el esfuerzo al oprimir el botón de inicio/fin del cronómetro 9. El usuario registra el tiempo que estima fue interrumpido 10. El usuario envía la información 11. El sistema guarda la información 12. El sistema despliega un mensaje de éxito en la operación
Cursos Alternativos:	Ninguno
Errores:	• El usuario ingresa un tiempo de interrupción mayor al tiempo cronometrado 4. El sistema despliega un mensaje indicando el error y regresa el control al usuario para que cambia el tiempo de interrupción
Incluye:	Refiérase al UC-PET-PPP-06 Ver Tarea por Hacer
Requerimientos Asociados:	Ninguna

Tabla 14 - Ejemplo 3 de caso de uso: registro de esfuerzo por paso de proceso

Administración de Usuarios

No.	ID del Caso de Uso:	Nombre del Caso de Uso:
45.	UC-PET-UAM-01	Login
46.	UC-PET-UAM-02	Logout
47.	UC-PET-UAM-03	Crear Cuenta de Usuario
48.	UC-PET-UAM-04	Mostrar Cuenta de Usuario
49.	UC-PET-UAM-05	Modificar Cuenta de Usuario
50.	UC-PET-UAM-06	Eliminar Cuenta de Usuario

Tabla 15 - Casos de uso de la administración de usuarios

ID del Caso de Uso: UC-PET-UAM-03	**Nombre del Caso de Uso:** Crear Cuenta de Usuario
Descripción:	Creación de una cuenta de usuario para ingreso al sistema
Condiciones Previas:	Ninguna
Condiciones Posteriores:	La cuenta de usuario queda registrada en el sistema
Frecuencia de Uso:	Alta
Curso Normal de los Eventos:	13. El usuario elige la opción de crear una cuenta de usuario 14. El usuario escribe el nombre de cuenta del nuevo usuario 15. El usuario escribe el nombre completo del nuevo usuario 16. El usuario escribe la contraseña del usuario 17. El usuario envía la información 18. El sistema guarda la información 19. El sistema despliega un mensaje de éxito en la operación
Cursos Alternativos:	Ninguno
Errores:	• El usuario ingresa un nombre de usuario ya existente 5. El sistema despliega un mensaje indicando el error y regresa el control al usuario para que cambia el nombre de usuario
Incluye:	Ninguna
Requerimientos Asociados:	Ninguna

Tabla 16 - Ejemplo 4 de casos de uso: crear cuenta de usuario

Requisitos Técnicos

Plataforma

La plataforma del proyecto debe de ser analizada desde dos puntos de vista distintos: el servidor y los clientes.

Para el servidor, la plataforma utilizada fue la siguiente:

Concepto	**Tecnología**
Tipo de Computadora	PC de alto rendimiento
Sistema Operativo	UNIX o Windows XP (Se utilizó Windows XP)
Base de Datos	MySQL 4.0.17
Servidor de Aplicaciones	JBoss 3.2.3
Lenguaje de Programación	Se uso Java, y específicamente las siguientes tecnologías: Java 2 Enterprise Edition 1.4 (J2EE 1.4) Enterprise Java Beans 2.0 (EJB2.0) Java Message Service (JMS)

Tabla 17 - Plataforma del servidor

Para el cliente:

Concepto	Tecnología
Tipo de Computadora	PC de alto rendimiento
Sistema Operativo	UNIX o Windows XP (Se utilizó Windows XP)
Lenguaje de Programación	Se uso Java, y específicamente las siguientes tecnologías: Java 2 Standard Edition 1.4.2_01 (J2SE 1.4.2) Java Foundation Classes – Swing (JFCSwing) Java 2 Enterprise Edition 1.4 (J2EE 1.4) Java Message Service (JMS)

Tabla 18 - Plataforma del cliente

Ambiente de Desarrollo

El ambiente de desarrollo varió a lo largo del proyecto, pero las herramientas que se utilizaron durante la mayor parte de éste fueron las siguientes:

Concepto	Tecnología
Tipo de Computadora	PC de alto rendimiento
Sistema Operativo	Windows XP
Ambiente Integral de Desarrollo (IDE)	JBuilder X Enterprise Edition
Kit de Desarrollo de Software (SDK)	J2SE1.4.2_01
Herramienta de Desarrollo de Interfaces Gráficas	JBuilder X Enterprise Edition

Tabla 19 - Ambiente de desarrollo

Capítulo 3. Requisitos de Sistema

Objetivo

El documento de Requisitos de Sistema (SRS por sus siglas en inglés) contiene el análisis de los requisitos especificados en el documento de requisitos de usuario. En este documento también se especifican los requisitos específicos de las interfaces.

Alcance

En este documento se analizan los requisitos tanto funcionales como no funcionales (de desempeño, facilidad de mantenimiento, escalabilidad,…) que el sistema debe de tener. En este documento no se define como van a ser implementados los requisitos. Tampoco se diseñan las interfaces, sino que más bien se especifican los lineamientos generales.

Referencias

Los documentos referenciados en la preparación del presente, o aquellos en los que se basa son:

Carnegie Mellon University. "A Discipline for Software Engineering" Watts S. Humprey. 1995 Addison-Wesley

3.1 Requisitos de Interfaces

Interfaz de Usuario

A grandes rasgos, la interfaz de usuario deberá tener lo siguiente:

- Ventana de registro
  Ventana independiente de la interfaz principal que permitirá al usuario introducir su nombre y contraseña para ingresar al sistema
- Menú principal
  Menú con las opciones iniciales. Por ejemplo, ver las opciones de ciclos de vida.
- Menús expandidos
  Menús con las opciones correspondientes a la opción del menú principal seleccionada. Por ejemplo añadir un ciclo de vida
- Área de trabajo
  Los detalles de objeto o función seleccionada, así como los campos y botones necesarios para manipularla

Interfaces de Hardware

El sistema debe de tener acceso a una plataforma con:

- conectividad en red según el protocolo TCP/IP
- Máquina Virtual de Java (JVM)

Interfaces de Software

- Una base de datos accesible a través de una interfaz estándar tipo JDBC

Interfaces de Comunicaciones

- Los cliente y el servidor van a intercomunicarse a través del protocolo TCP/IP

3.2 Requisitos Funcionales

Análisis de Características

Debido a la naturaleza de los requisitos funcionales de este sistema, éstos han sido divididos en las siguientes 5 categorías:

1. Administración de Ciclos de Vida
2. Administración de Proyectos
3. Administración de Entregables
4. Predicción de Desempeño Profesional
5. Manejo de Cuentas de Usuario

El análisis de los requisitos, independientemente de a que categoría correspondan se va a hacer mediante diagramas de actividad según son descritos en el Lenguaje de Modelado Unificado (UML). Muchos de los diagramas tienen patrones que se repiten, por lo que éstos son recogidos en diagramas generales.

Los diagramas que son expuestos en este documento son de los siguientes tres tipos:

1. Diagramas generales de actividades
   Patrones de actividades que son sintetizados para evitar la repetición
2. Diagramas selectos de actividades
   Algunos ejemplos selectos de diagramas de actividad
3. Diagrama de actividad de alto nivel
   diagrama de alto nivel que cubre las actividades según un escenario típico de uso

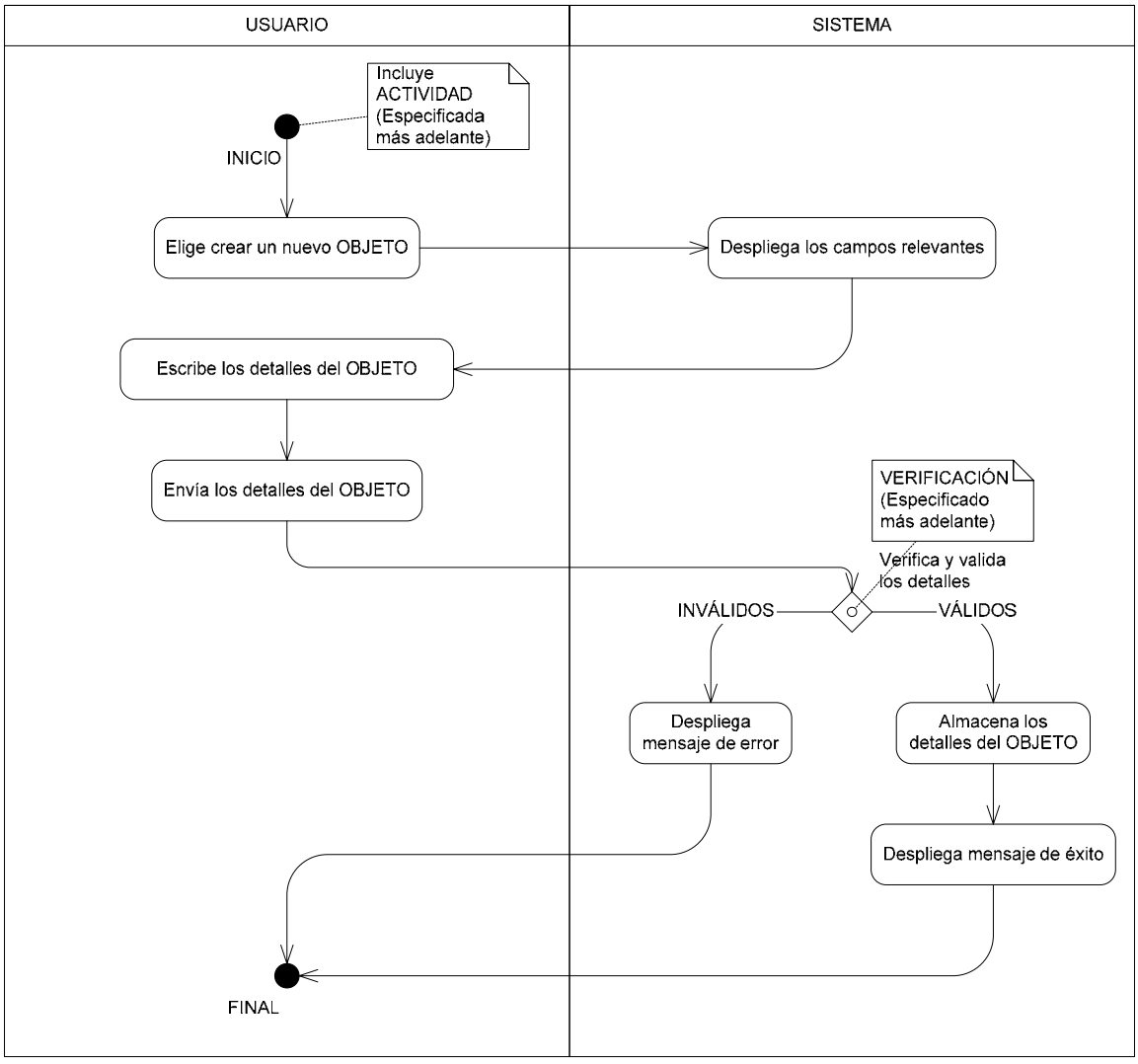

Diagrama 11 - Diagrama de actividades de creación de objeto

El patrón anterior fue sintetizado de los casos de uso de creación de algún objeto, entendiéndose por objeto los que se especifican a continuación.

Nombre	**Actividad Incluida**	**Verificación**	**Objeto**
Crear ciclo de vida	Ver lista de ciclos de vida	El nombre del ciclo de vida debe de ser único y no vacío	Ciclo de Vida

Nombre	Actividad Incluida	Verificación	Objeto
Agregar fase al ciclo de vida	Ver lista de ciclos de vida	El número de secuencia de la fase debe de ser único y un número natural	Fase de ciclo de vida
Crear entregable ad-hoc	Ver detalles del proyecto	El nombre del entregable debe de ser único para esa fase de proyecto y no vacío	Entregable ad-hoc
Crear entregable prototipo	Ver detalles del ciclo de vida	El nombre del entregable debe de ser único para esa fase de ciclo de vida y no vacío	Entregable prototipo

Tabla 20 - Objetos para el modelo de creación de objetos

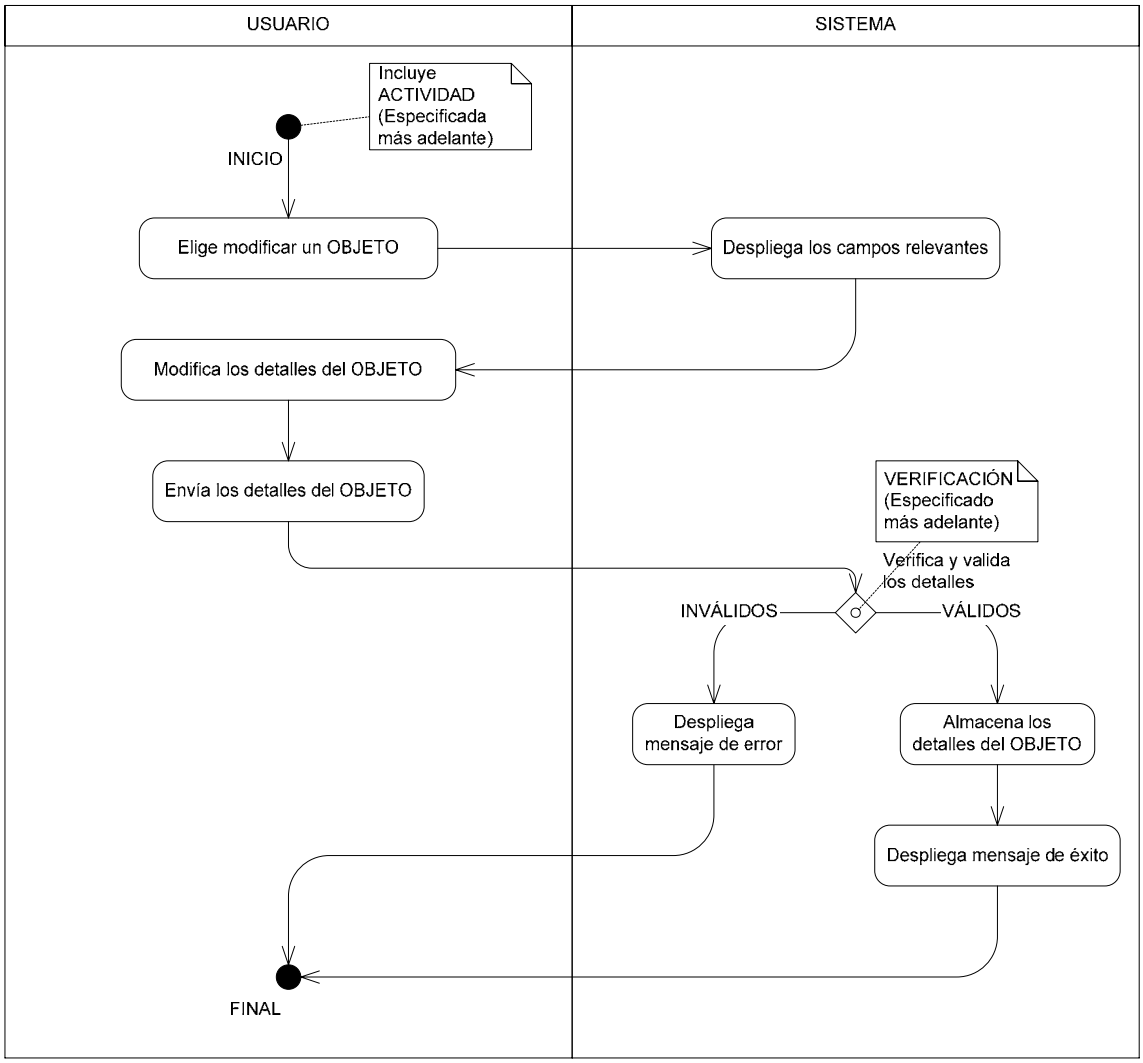

Diagrama 12 - Diagrama de actividades de modificación de objeto

El patrón anterior fue sintetizado de los casos de uso de modificación de algún objeto, entendiéndose por objeto los que se especifican a continuación.

Nombre	Actividad Incluida	Verificación	Objeto
Modificar ciclo de vida	Ver lista de ciclos de vida	El nombre del ciclo de vida debe de ser único y no vacío	Ciclo de Vida
Modificar fase del ciclo de vida	Ver lista de ciclos de vida	El número de secuencia de la fase debe de ser único y un número natural	Fase de ciclo de vida
Modificar Proyecto	Ver lista de proyectos	El nombre del proyecto debe de ser único y no vacío	Proyecto
Modificar entregable ad-hoc	Ver detalles del proyecto	El nombre debe de ser único para esa fase de proyecto y no vacío. El entregable no puede ser modificado si tiene esfuerzo registrado	Entregable ad-hoc
Modificar entregable prototipo	Ver detalles del ciclo de vida	El nombre debe de ser único y no vacío para esa fase de ciclo de vida El entregable no puede ser modificado si tiene esfuerzo registrado	Entregable prototipo
Modificar paso de proceso	Ver tarea por hacer	El nombre debe de ser único para el entregable y no vacío	

Tabla 21 - Objetos para el modelo de modificación de objetos

Los anteriores son dos ejemplos de patrones de actividades que sintetizan diagramas de actividades. Una generalización similar se hizo para los siguientes casos:

- Ver objeto
- Ver lista de objetos
- Eliminar objeto

Los diagramas que no seguían un patrón fueron detallados independientemente, y en este caso se muestra un ejemplo.

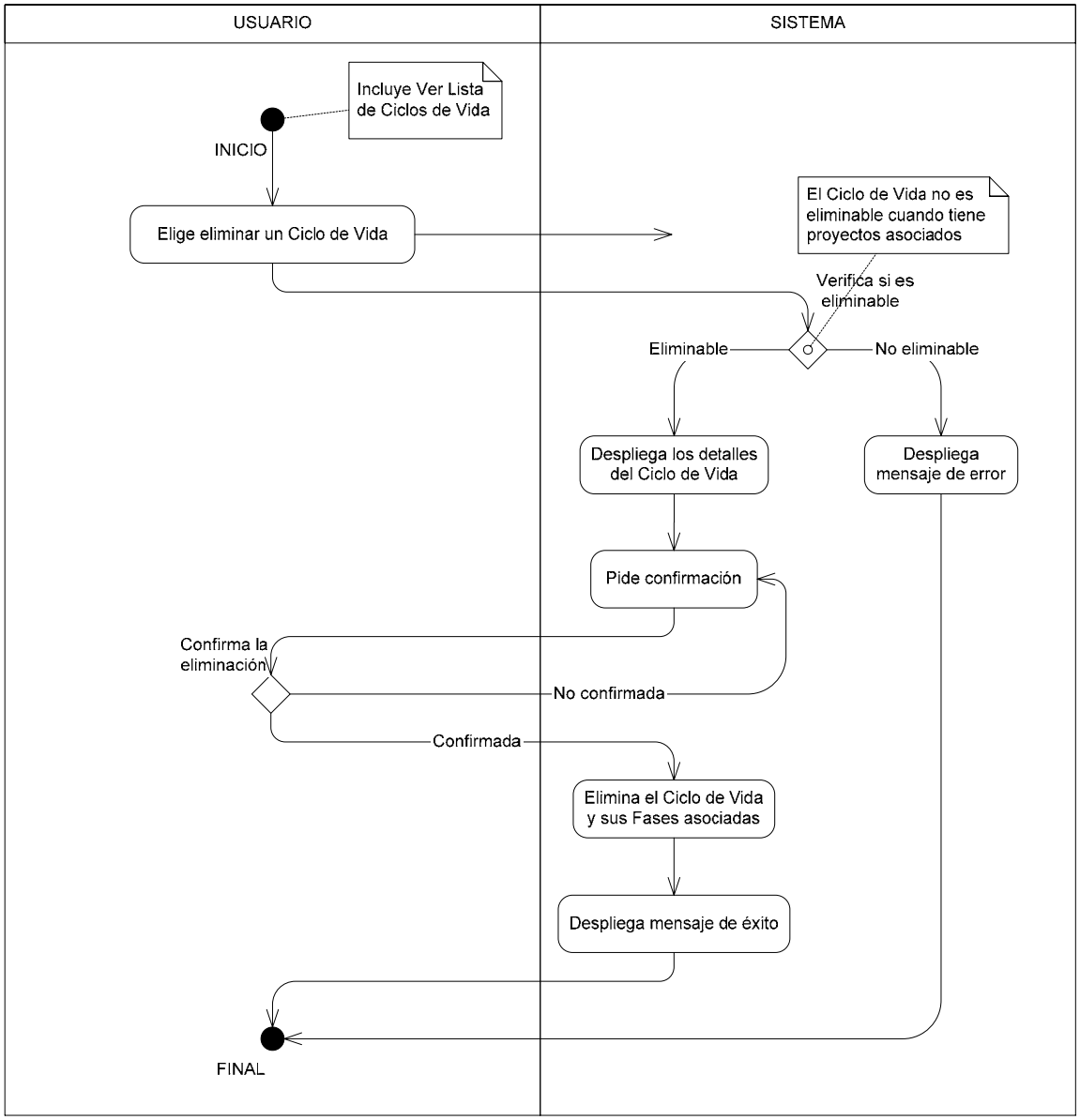

Diagrama 13 - Diagrama de actividad de eliminar ciclo de vida

Finalmente se especifica un escenario de uso y se agrega un diagrama de actividad de alto nivel.

El escenario:

Un usuario que administra un proyecto ya está registrado en el sistema. Este usuario crea un entregable de proyecto usando como plantilla un entregable prototipo o decide mejor crear un entregable ad-hoc. Finalmente asigna este entregable a un desarrollador.

El desarrollador también está ya registrado en el sistema. El desarrollador ve su lista de tareas por hacer y luego ve los detalles del entregable que le fue asignado. Decide

empezar a trabajar en esa tarea y empieza a registrar tiempo para el primer paso de proceso. Cuando termina ese paso, el sistema registra el tiempo dedicado. Este tiempo registrado puede ser visto por el administrador del proyecto y por el desarrollador. Cuando termina con la tarea, confirma los datos recopilados de esfuerzo e introduce datos extras como tiempo estimado de interrupciones,… A esta actividad, se le denomina reflexión. Una vez terminada ésta, marca el entregable como terminado y lo almacena.

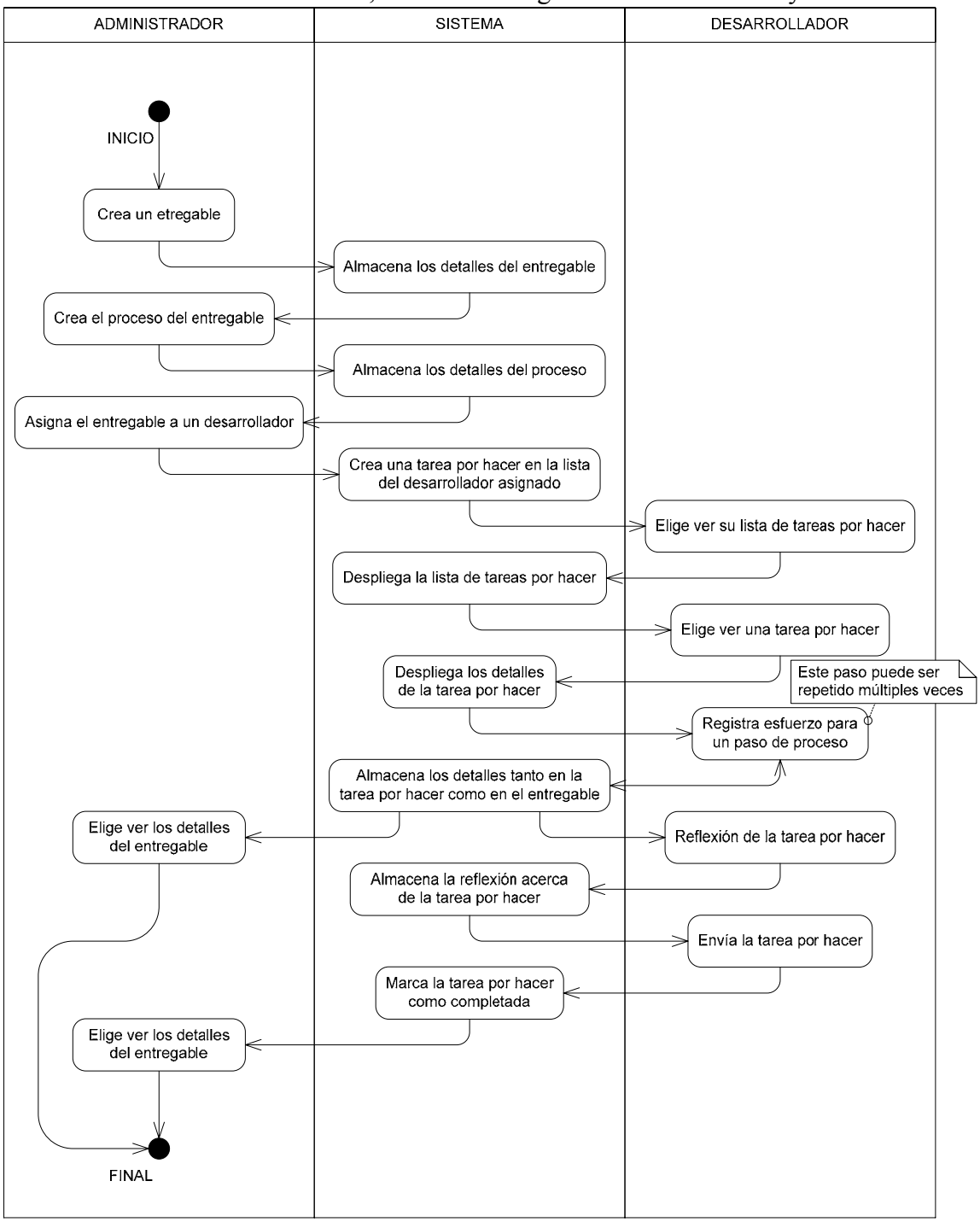

Diagrama 14 - Diagrama de actividades de alto nivel para el escenario

Análisis de Objetos

A estas alturas del proyecto, las clases no han sido definidas, por lo que solamente se puede hablar de conceptos que pueden o no ser clases o incluso comprender más de una clase en el diseño.

Numero.	Nombre del Concepto	Responsabilidad
1.	Ciclo de vida	Administra un ciclo de vida incluido un conjunto de fases de ciclo de vida
2.	Fase de ciclo de vida	Administra una sola fase de ciclo de vida incluido un conjunto de entregables prototipo
3.	Proyecto	Administra un solo proyecto incluido el conjunto de fases de proyecto
4.	Fase de proyecto	Administra una sola fase de proyecto incluido un conjunto de entregables de proyecto o ad-hoc
5.	Entregable prototipo	Administra un solo entregable prototipo incluido un conjunto de pasos de proceso
6.	Entregable de proyecto	Administra un solo entregable de proyecto incluido un conjunto de pasos de proceso
7.	Entregable Ad-hoc	Administra un entregable ad-hoc
8.	Tarea por hacer	Administra una tarea por hacer
9.	Paso de proceso	Administra un solo paso de proceso
10.	Lista de tareas por hacer	Administra una lista de tareas por hacer
11.	Lista de tareas completadas	Administra una lista de tareas completadas
12.	Cuenta de usuario	Administra una sola cuenta de usuario
13.	Registro de esfuerzo	Administra un solo registro de esfuerzo

Tabla 22 - Objetos a nivel conceptual

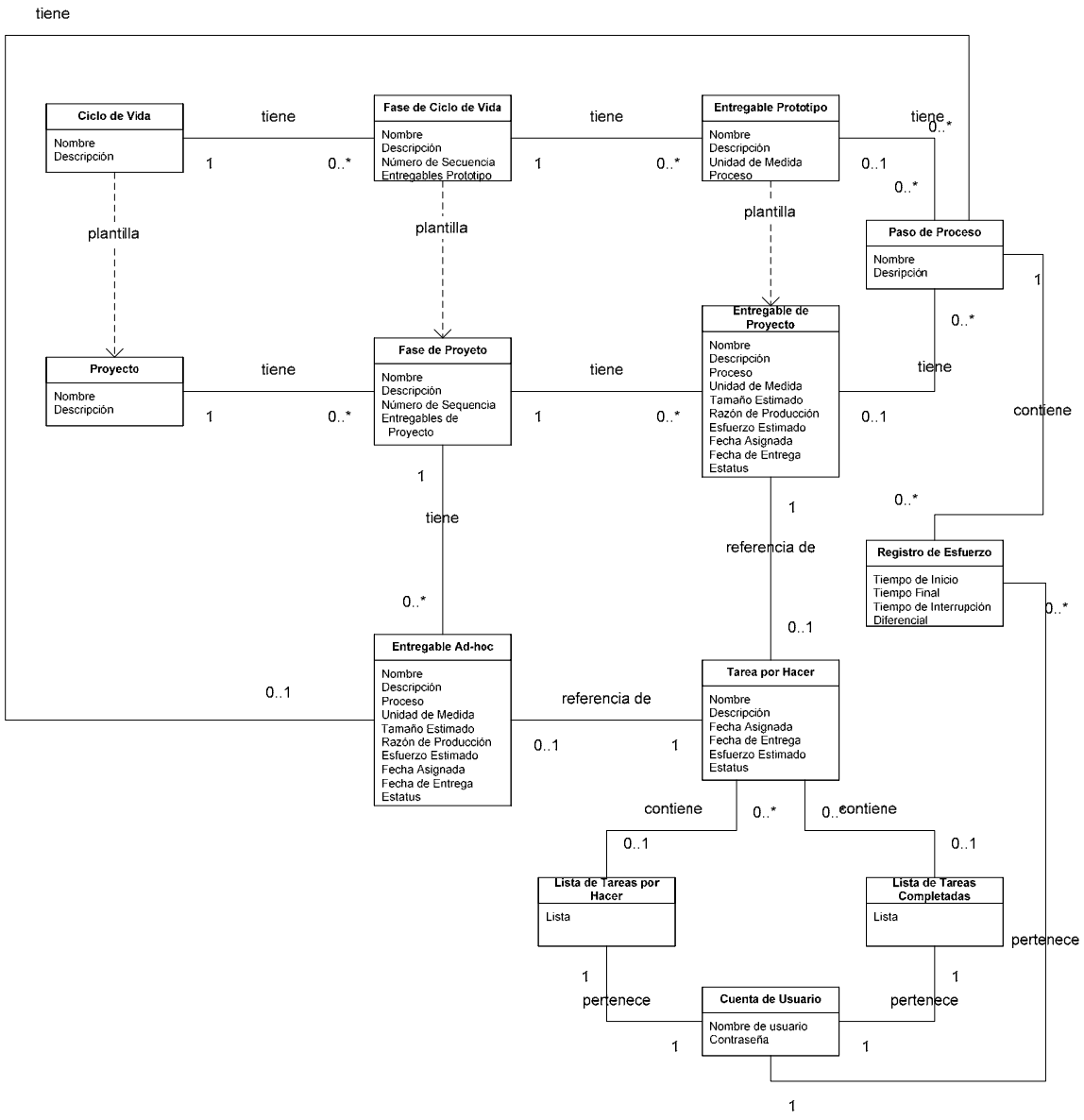

Diagrama 15 - Diagrama global de conceptos

Análisis de Estados

Para esta entrega del sistema, sólo existen 2 tipos de objetos que pueden tener diferentes estados:

- Ciclos de vida y proyectos
- Entregables

El resto de los objetos, sólo tienen estados muy simples del tipo de: creado, modificado y eliminado.

Los ciclos de vida y los proyectos tienen suficientes características similares para ser analizados en conjunto; específicamente para el análisis de estados, son exactamente iguales.

La posibilidad de modificar un ciclo de vida o un proyecto depende de si tiene referencias o no. Un ciclo de vida tiene referencias cuando algún proyecto es creado partiendo de el. Un proyecto es referenciado cuando algún desarrollador empieza a registrar esfuerzo a alguna de sus tareas.

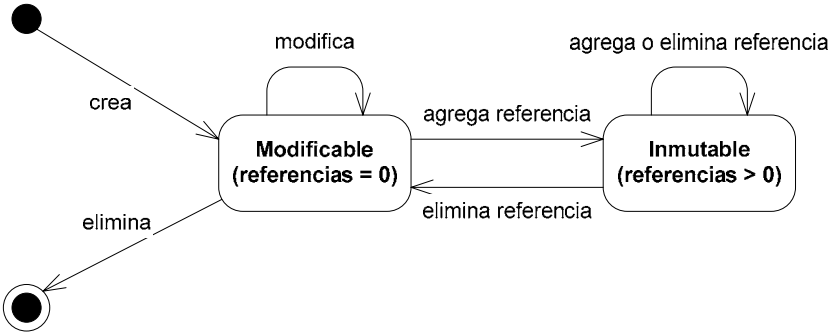

Diagrama 16 - Diagrama de estados de ciclo de vida/proyecto

Un entregable sólo puede ser eliminado cuando no hay esfuerzo registrado a el. Por otro lado, de un entregable prototipo, pueden ser derivados múltiples entregables de proyecto. Cabe notar que una vez que un entregable es asignado, éste comienza a ser considerado una tarea por hacer para el desarrollador a quien fue asignado.

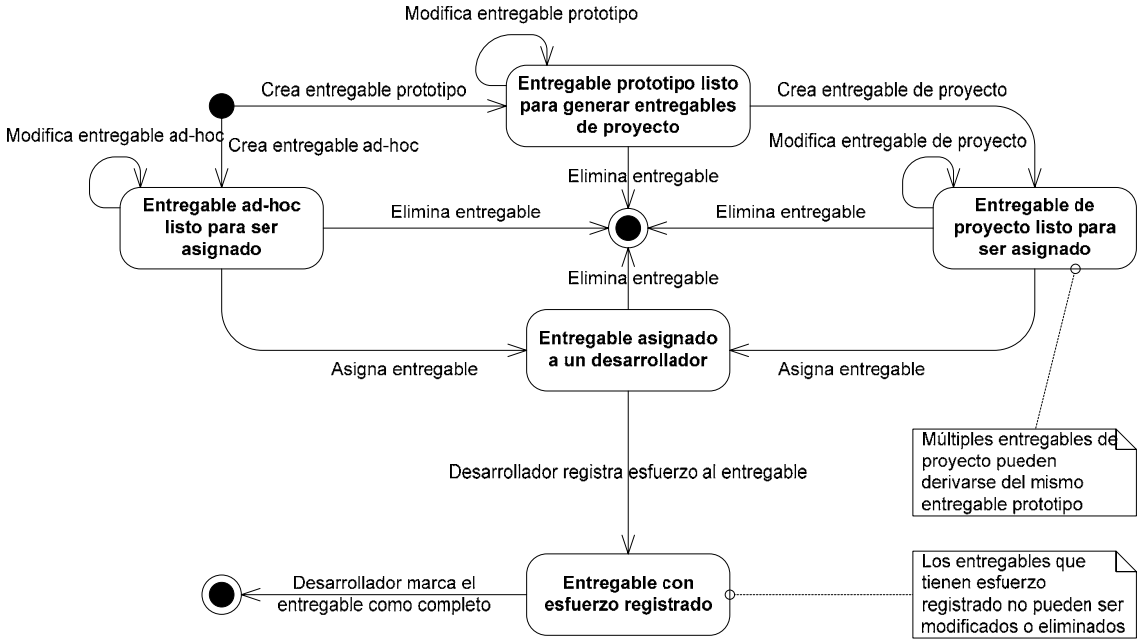

Diagrama 17 - Diagrama de estados de entregables

3.3 Requisitos No Funcionales

Requisitos de Rendimiento

El sistema va a consistir en un servidor central recibiendo peticiones de clientes múltiples.

El sistema debe de ser capaz de dar servicio a 50 usuarios concurrentes con un promedio de una petición por cada 10 segundos para cada uno. El sistema debe de estar preparado para recibir dichas peticiones con una distribución exponencial dependiendo del número de usuarios conectados.

El sistema debe de tener una respuesta media de 2 segundos por petición.

Requisitos de Seguridad y Confiabilidad

Existe solamente dos posibilidades de corrupción o pérdida de datos en este sistema, cuando la información es transmitida entre el cliente y el servidor y cando ésta es almacenada en el sistema de almacenamiento permanente del servidor.

La posibilidad de fallas en la transmisión de datos entre el cliente y el servidor depende directamente de la red que se utilice. Por otra parte, las posibilidades de falla en las transacciones entre el servidor y la base de datos pueden depender de la red, en el caso de que estén en diferentes computadoras o del sistema operativo.

Para asegurar la información y la integridad de ésta, se utilizará control de transacciones tanto entre el servidor y los clientes como con la base de datos.

Requisitos de Calidad de Software

Modificaciones

El sistema va a ser implementado mediante componentes. Cada componente debe de ser auto-contenido de manera que puedan ser modificados o reemplazadas conforme sea necesario.

La interfaz de usuario debe de ser independiente del resto del sistema.

El repositorio de datos va a estar constituido por un Manejador de Bases de Datos Relacional (RDBMS, por sus siglas en inglés) disponible comercialmente directamente del estante (COTS, por sus siglas en inglés). Éste debe de ser fácilmente reemplazable, es decir el sistema debe de estar diseñado de manera que no dependa de alguna función propia de algún RDBMS específico.

Todos los COTS deben de estar disponibles para varias plataformas de manera que el sistema final también funcione en múltiples plataformas.

Facilidad de Uso

La interfaz de usuario debe de ser fácil de usar por un usuario familiarizado con el ambiente operativo de MS Windows.

El usuario debe de confirmar el envío de información en todas las ocasiones en las que esto sea necesario. El sistema no va a permitir al usuario las operaciones de revertir o el re-uso de información previamente enviada.

Mantenimiento
El sistema debe de ser diseñado con una arquitectura que claramente defina cada uno de los componentes y las interfaces de éstos. También debe de ser diseñado utilizando prácticas estándares para permitir que sea fácil de mantener.

Escalabilidad
El sistema debe de ser capaz de satisfacer las necesidades de una organización pequeña que tenga 10 proyectos concurrentemente con 5 desarrolladores cada uno de ellos. El sistema debe de ser fácilmente ampliable para organizaciones de mayor tamaño.

Pruebas
El diseño del sistema debe de diferenciar claramente las unidades lógicas y de tener interfaces bien definidas entre éstas. Cada unidad lógica debe de ser capaz de reportar su estado interno al igual que tener salidas preestablecidas en caso de error en alguna operación.

Capítulo 4. Arquitectura

Objetivo

El documento de Arquitectura de Software (SAD, por sus siglas en inglés) contiene la descripción de la arquitectura seleccionada para el sistema.

Alcance

En este documento se describen los componentes del sistema desde el punto de vista de su arquitectura. Estos componentes pueden ser vistos como subsistemas y no necesariamente satisfacen enteramente uno o varios requisitos por si mismos.

Referencias

Los documentos referenciados en la preparación del presente, o aquellos en los que se basa son:

IEEE Software 12, 6. "The 4+1 View Model of Architecture", Kruchten P. 1995

Bass, Clements, Kazman. "Software Architecture in Practice" 2nd Ed.

4.1 El modelo de las 4+1 Vistas de Philippe Kruchten

El modelo más completo de descripción de la arquitectura de un sistema de software fue desarrollado por Philippe Kruchten de Racional Software Corp. y publicado en la revista IEEE Software de noviembre de 1995.

De acuerdo a Kruchten, "*La arquitectura de software trata con el diseño e implementación de la estructura de alto nivel del software. Es el resultado de ensamblar un cierto número de elementos arquitectónicos en formas bien conocidas que satisfagan los mayores requisitos de funcionalidad y rendimiento del sistema al igual que otros requisitos no-funcionales tales como, confiabilidad, escalabilidad, portabilidad y disponibilidad.*

…

… La arquitectura de software trata con la abstracción, descomposición y composición, con estilo y claridad. Para describir una arquitectura de software, usamos un modelo de múltiples vistas o perspectivas".

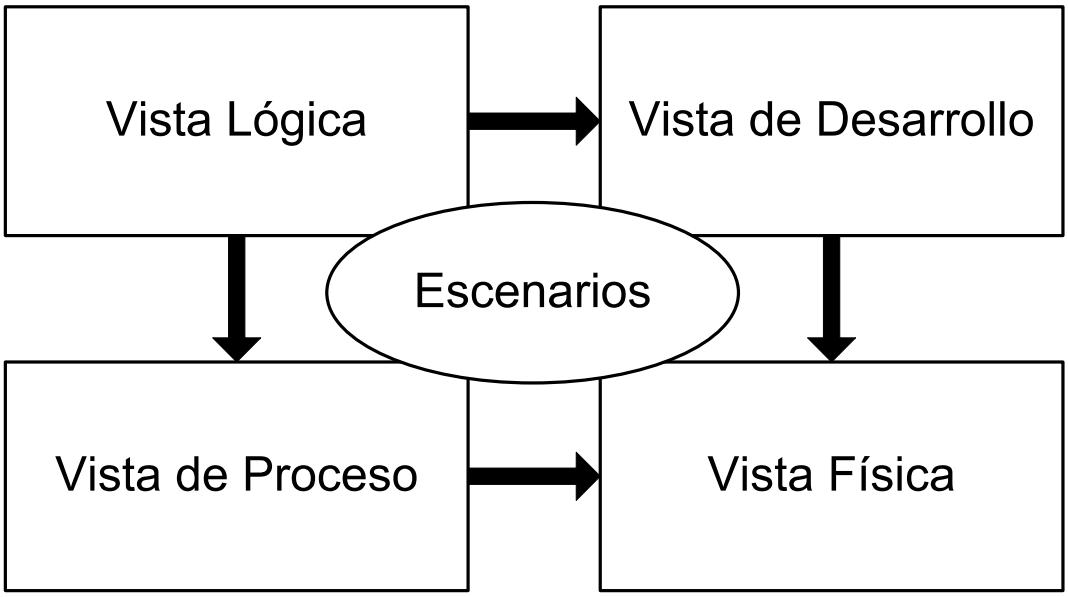

Diagrama 18 - El modelo de las 4+1 vistas

Kruchten define como sus 4 vistas a las siguientes:

- Vista Lógica
  Vista que describe los elementos lógicos del sistema.
- Vista de Desarrollo
  Vista que describe los elementos de software del sistema como van a ser implementados.
- Vista de Proceso
  Vista que describe como interactúan los elementos del sistema

- Vista Física
  Vista que describe como van a distribuirse los elementos del sistema en diferentes componentes de hardware.

La vista "+1" consiste en la descripción de uno o más escenarios específicos en los que se muestra como trabaja la arquitectura.

4.2 Características de la Arquitectura Seleccionada

La arquitectura escogida para este proyecto fue la "Basada en Eventos". Este tipo de arquitectura tiene como características principales las siguientes:

- La comunicación entre clientes y servidor es mediante mensajes
- Existe un componente que administra los mensajes llamado "Manejador de eventos"
- Cuando un cliente quiere hacer algún cambio a los datos, éste envía un mensaje de petición
- Cuando el servidor hace algún cambio a los datos, éste envía un mensaje de notificación a todos los clientes pertinentes. Éste caso es conocido como un evento

4.3 Metas Arquitectónicos y Restricciones

- Arquitectura basada en componentes
  El sistema debe de poder ser instalado en múltiples plataformas
- Escalabilidad
  El sistema debe de ser capaz de dar servicio a 10 proyectos concurrentemente con 5 desarrolladores cada uno
- Consistencia de datos
  El sistema utiliza un RDBMS para almacenamiento permanente de datos y debe de resolver los problemas resultantes de accesos concurrentes
- Mantenimiento
  El sistema debe de ser diseñado con una arquitectura que claramente defina cada uno de los componentes y las interfaces de éstos
- Modificaciones
  La RDBMS que va a usar el sistema como repositorio de datos, debe de ser fácilmente reemplazable. Por otro lado, la interfaz de usuario, debe de ser independiente del resto del sistema, de modo que se pueda cambiar su apariencia sin afectar ningún otro componente
- Manejador de eventos
  El manejador de eventos debe de controlar todas las peticiones de los usuarios y notificar a éstos de todo cambio ocurrido en los datos almacenados. Para lograr esto, el manejador de eventos va a tener una cola donde se procesaran las peticiones de los clientes siguiendo las primeras entradas, primeras salidas (PEPS o FIFO, por sus siglas en inglés)

4.4 Arquitectura Lógica

En la vista lógica de la arquitectura se definen los componentes del sistema desde el punto de vista de los requisitos funcionales. Para esto se definen las clases o paquetes que van a ser implementados y cómo se conectan éstos. Hay que tomar siempre en cuenta, que a estas alturas del diseño, aún no se definen las clases como van a quedar en la implementación. Debido a esta última razón, se utilizaron sólo paquetes en este diagrama.

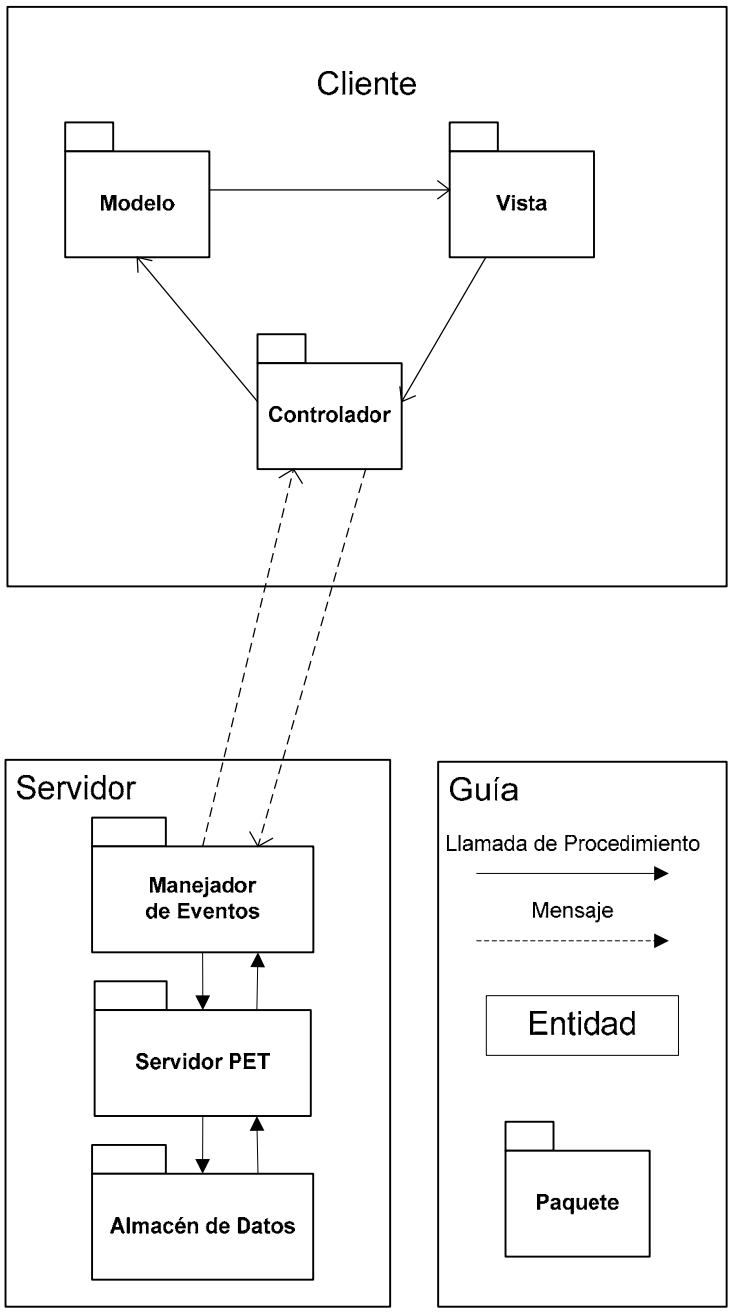

Diagrama 19 - Vista lógica

Como se puede apreciar en el Diagrama 19, los paquetes adscritos en el cliente, son los del modelo arquitectónico llamado "Modelo-Vista-Controlador" (MVC, por sus siglas en inglés). El hecho de que las clases pertenecientes al cliente estén separadas de esta forma asegura que éstas sean implementadas de forma independiente entre paquetes.

Por otro lado, el servidor está compuesto por un paquete que se encarga del Administrador de eventos y por ende de las comunicaciones, otro con las funciones propias del servidor y el último del almacenamiento de datos. Cabe recordar que el almacén de datos va a estar constituido por un COTS.

4.5 Arquitectura de Procesos

En la vista de proceso de la arquitectura se describen los tipos de comunicaciones que se dan entre los componentes.

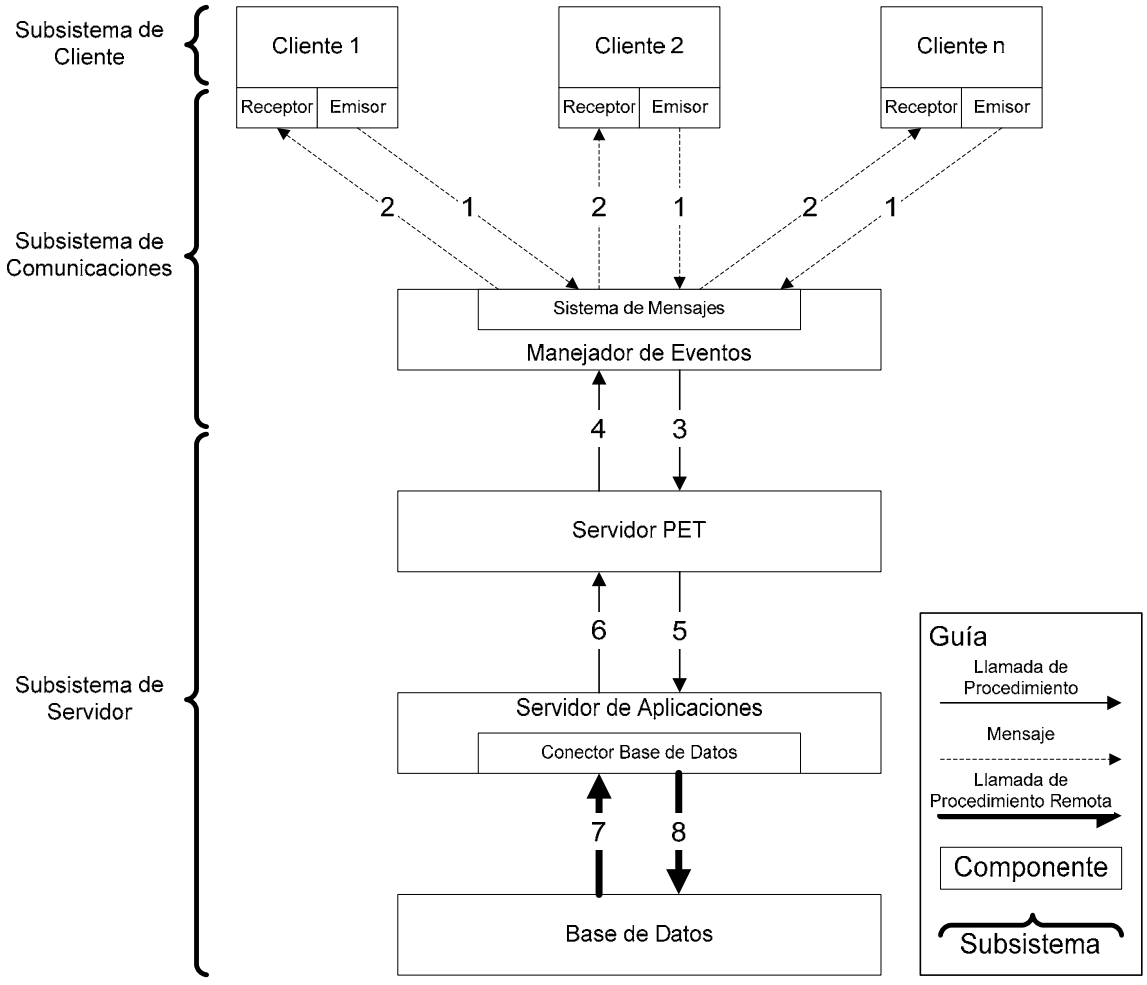

Diagrama 20 - Vista de proceso

En el Diagrama 20, cada conector representa un enlace entre componentes. Estos enlaces son de diferentes tipos y la comunicación establecida a través de ellos es de diversos tipos. Éstos son descritos para cada enlace a continuación:

Conector 1.

- Mensaje de Petición de Objeto de Datos
  El cliente manda un mensaje de petición de un objeto de datos
- Mensaje de Petición de Actualización de Datos
  El cliente manda una petición de actualización de datos en el servidor con un objeto de datos
- Mensaje de Petición de Autentificación de Usuario
  El cliente manda una petición de inicio de sesión (login) para registrarse como activo o de termino de sesión (logout) para eliminar su registro de cliente activo
- Mensaje de Respuesta de Prueba de Conexión
  Respuesta del cliente a un mensaje del manejador de eventos de verificación de conexión

Conector 2.

- Mensaje de Respuesta de Objeto de Datos
  El manejador de eventos regresa el mensaje de objeto de datos pedido por el cliente
- Mensaje de Respuesta de Actualización de Datos
  El manejador de eventos envía un mensaje a todos los clientes activos sobre la actualización en caso de haberse realizado ésta. En el caso contrario, envía un mensaje de falla al cliente responsable
- Mensaje de Respuesta de Autentificación de Usuario
  En el caso de que el cliente haya sido autentificado correctamente, el manejador de eventos envía un mensaje de confirmación. Por otro lado, si éste es desconocido, el mensaje es de rechazo
- Mensaje de Petición de Prueba de Conexión
  Periódicamente, el manejador de eventos envía un mensaje a todos los clientes registrados como activos para comprobar si éstos realmente lo están

Conector 3.

- Llamadas de Procedimiento

Conector 4.

- Regreso de Procedimiento

Conector 5.

- Llamadas de Procedimiento

Conector 6.

- Regreso de Procedimiento

Conector 7.

- Llamada de Procedimiento Remoto
  El objeto de tener llamadas de procedimiento remoto es el de dar mayor flexibilidad para no forzar a que el RDBMS esté en la misma computadora que el servidor de aplicaciones

Conector 8.

- Regreso de Procedimiento Remoto

El contenido de los mensajes es el siguiente:

Mensaje	**Contenido**
Mensaje de Petición de Objeto de Datos	ID Mensaje, Origen, Destino, ID Objeto
Mensaje de Respuesta de Objeto de Datos	ID Mensaje, Origen, Destino, Objeto, Bandera de Éxito
Mensaje de Petición de Actualización de Datos	ID Mensaje, Origen, Destino, Objeto
Mensaje de Respuesta de Actualización de Datos	ID Mensaje, Origen, ID Objeto, Bandera de Éxito
Mensaje de Petición de Autentificación de Usuario	ID Mensaje, Origen, Destino, Objeto
Mensaje de Respuesta de Autentificación de Usuario	ID Mensaje, Origen, Destino, Objeto, Bandera de Éxito
Mensaje de Petición de Prueba de Conexión	ID Mensaje, Origen, Destino
Mensaje de Respuesta de Prueba de Conexión	ID Mensaje, Origen, Destino

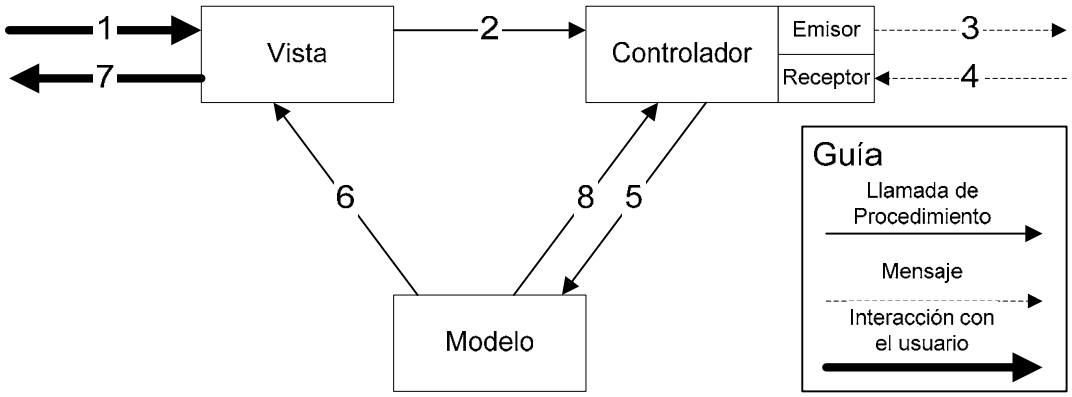

Diagrama 21 - Vista de proceso - detalle del cliente

Conector 1.

- Entradas de usuario

Conector 2.

- La instancia de la vista que implementa la interfaz de usuario utiliza al controlador para hacer cualquier operación

Conector 3.

- El controlador envía mensajes al manejador de eventos para solicitar información

Conector 4.

- La información es recibida del manejador de eventos por el controlador
- El controlador también recibe notificaciones de actualización en algún objeto

Conector 5.

- El controlador guarda la información recibida en el modelo
- El controlador verifica en el modelo si la notificación de actualización corresponde a algún objeto de los almacenados en el modelo

Conector 6.

- El modelo envía la nueva información a la vista
- El modelo notifica a la vista si hay cambios en la información desplegada

Conector 7.

- El usuario recibe la información mediante la interfaz gráfica de usuario

Conector 8.

- El modelo avisa al controlador si la notificación de actualización es sobre información guardada en el modelo o no

4.6 Arquitectura de Desarrollo

En la vista de desarrollo de la arquitectura se describen los diferentes módulos que componen al sistema desde el punto de vista del desarrollo de software. Se describen sus jerarquías y sus dependencias.

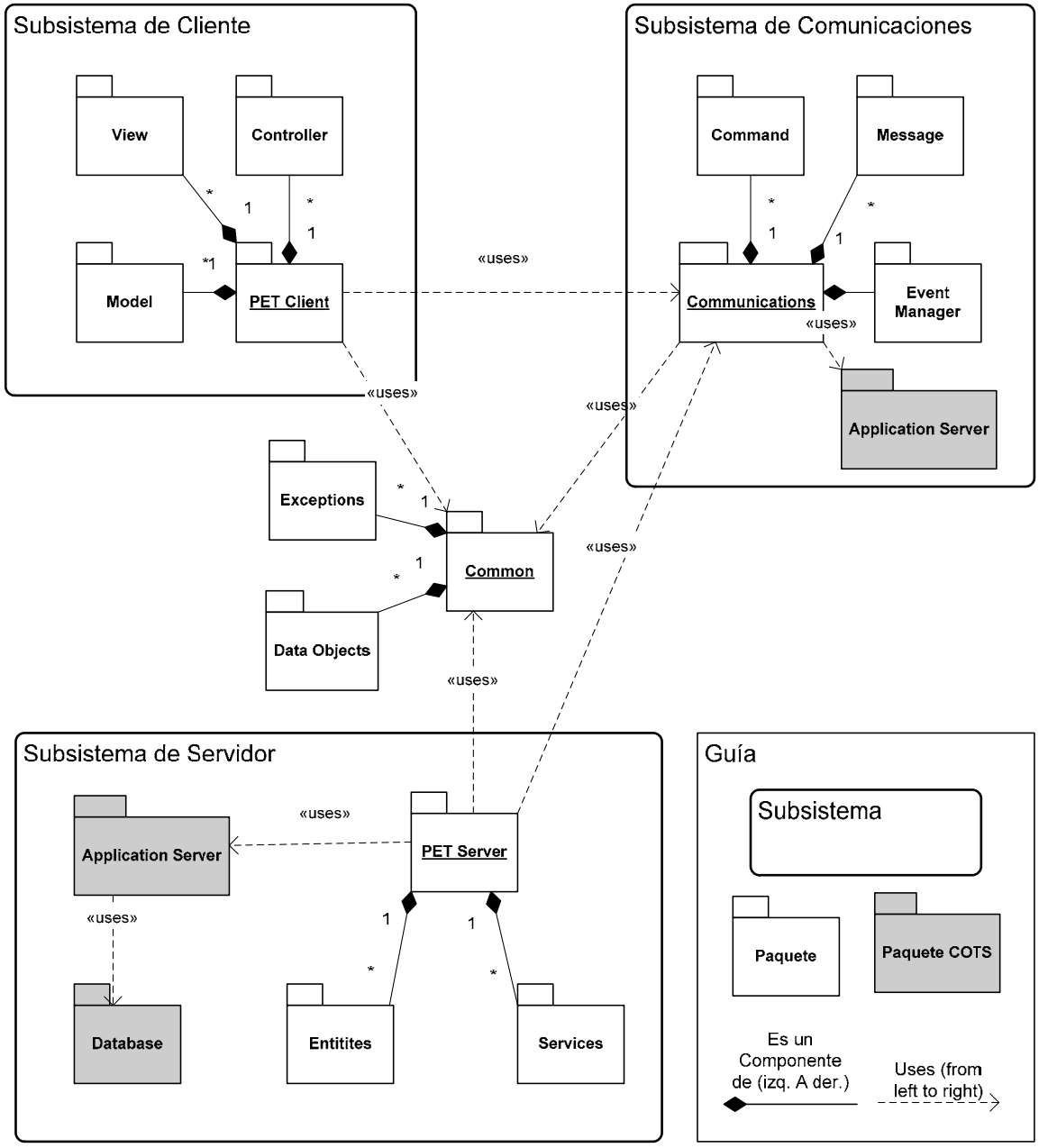

Diagrama 22 - Vista de desarrollo

En este diagrama, se muestran los paquetes que van a componer el sistema así, como las relaciones entre éstos. También se puede apreciar cómo se agrupan éstos en subsistemas con excepción de los paquetes catalogados de comunes. Estos últimos paquetes contienen funcionalidad que es compartida por más de un subsistema, razón por la cual no están agrupados.

Cabe destacar que aunque tanto el subsistema de servidor como el de comunicaciones utilizan un servidor de aplicaciones, no necesariamente deben de usar uno distinto o estar en computadoras separadas.

4.7 Arquitectura Física

En la vista física de la arquitectura, se describe la distribución de los componentes de software en los de hardware, es decir, se describe como se va a instalar el sistema.

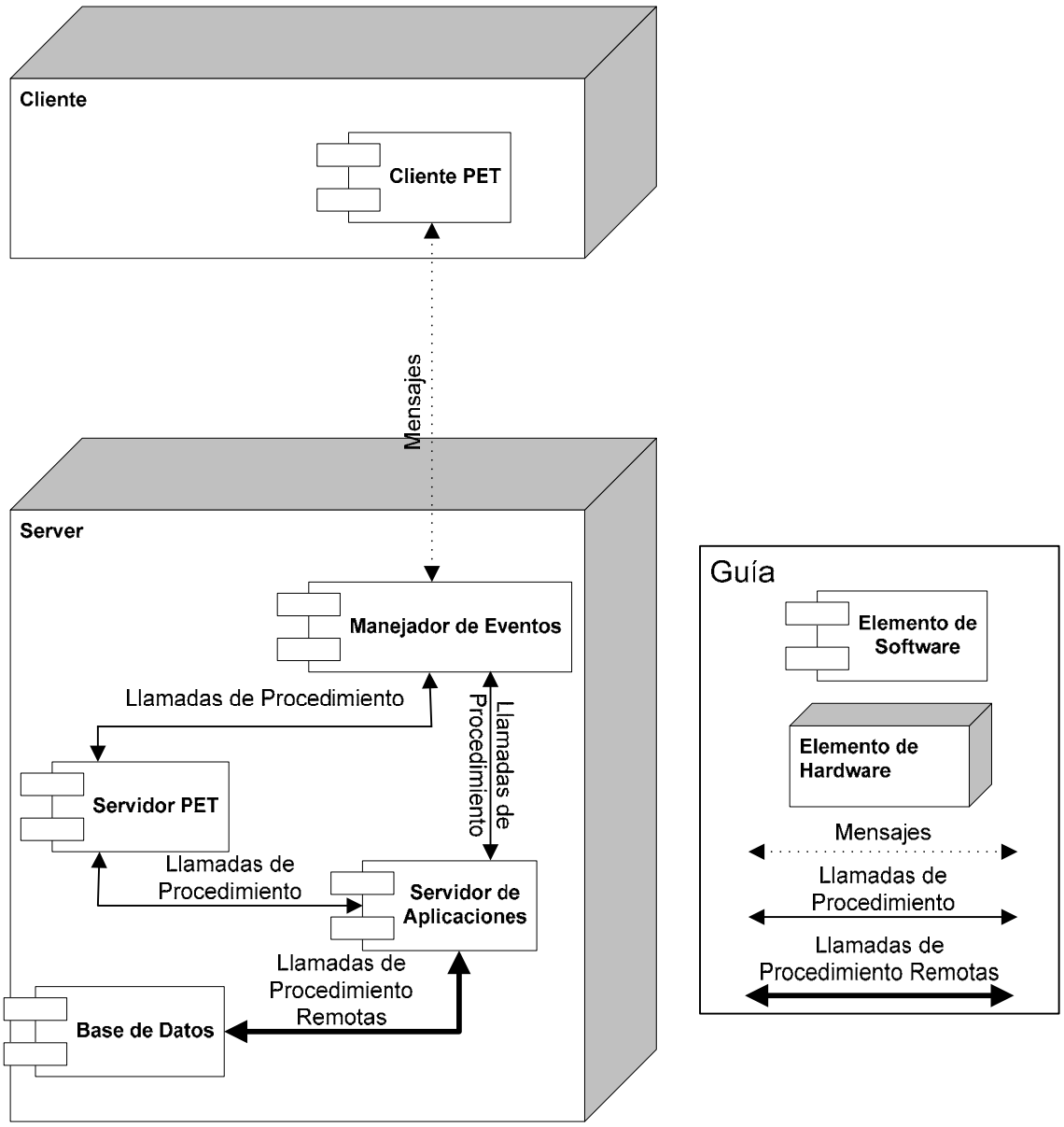

Diagrama 23 - Vista física

En este diagrama, se describe la forma en que se decidió instalar el sistema, pero es importante señalar que no es la única distribución posible de los elementos de software. Un ejemplo de distribución alterna sería el de utilizar un servidor de bases de datos, razón por la cual el sistema está diseñado con llamadas remotas entre el servidor de aplicaciones y la base de datos. Otra posibilidad sería la de separar el manejador de eventos del resto del servidor mediante otra computadora con un servidor de aplicaciones propio. Para esta última modificación sería necesario modificar las llamadas de procedimiento entre el manejador de eventos y el resto del sistema para que sean remotas.

Todas estas posibilidades pueden ser aplicadas para darle mayor flexibilidad y/o escalabilidad al sistema.

Para este proyecto se decidió que el servidor sea una PC de alto rendimiento con un servidor de aplicaciones JBoss y un manejador de bases de datos MySQL corriendo sobre cualquier sistema operativo que pueda correr estas aplicaciones y que tenga una maquina virtual de Java (JVM, por sus siglas en inglés). El cliente es cualquier computadora que tenga una JVM con un rendimiento aceptable para poder ejecutar aplicaciones desarrolladas usando los componentes gráficos de JFCSwing de Java.

4.8 Escenarios

Escenario 1: Creación y asignación de un entregable de proyecto

En este escenario, tenemos a un usuario que es líder de un equipo de desarrollo, referido de ahora en adelante como “líder”. Por otro lado tenemos a un desarrollador que trabaja en el equipo del líder, de ahora en adelante “programador”.

El líder quiere crear un entregable llamado “URD 1.0” para el proyecto “PET” que utiliza el entregable prototipo llamado “URD” de la fase de “Análisis” del ciclo de vida “Cascada”. Una vez creado el entregable, el líder lo asigna al programador para que este lo lleve a cabo. Por otro lado, el programador abre su lista de tareas por hacer y cuando el líder le asigna el entregable, éste aparece en ella.

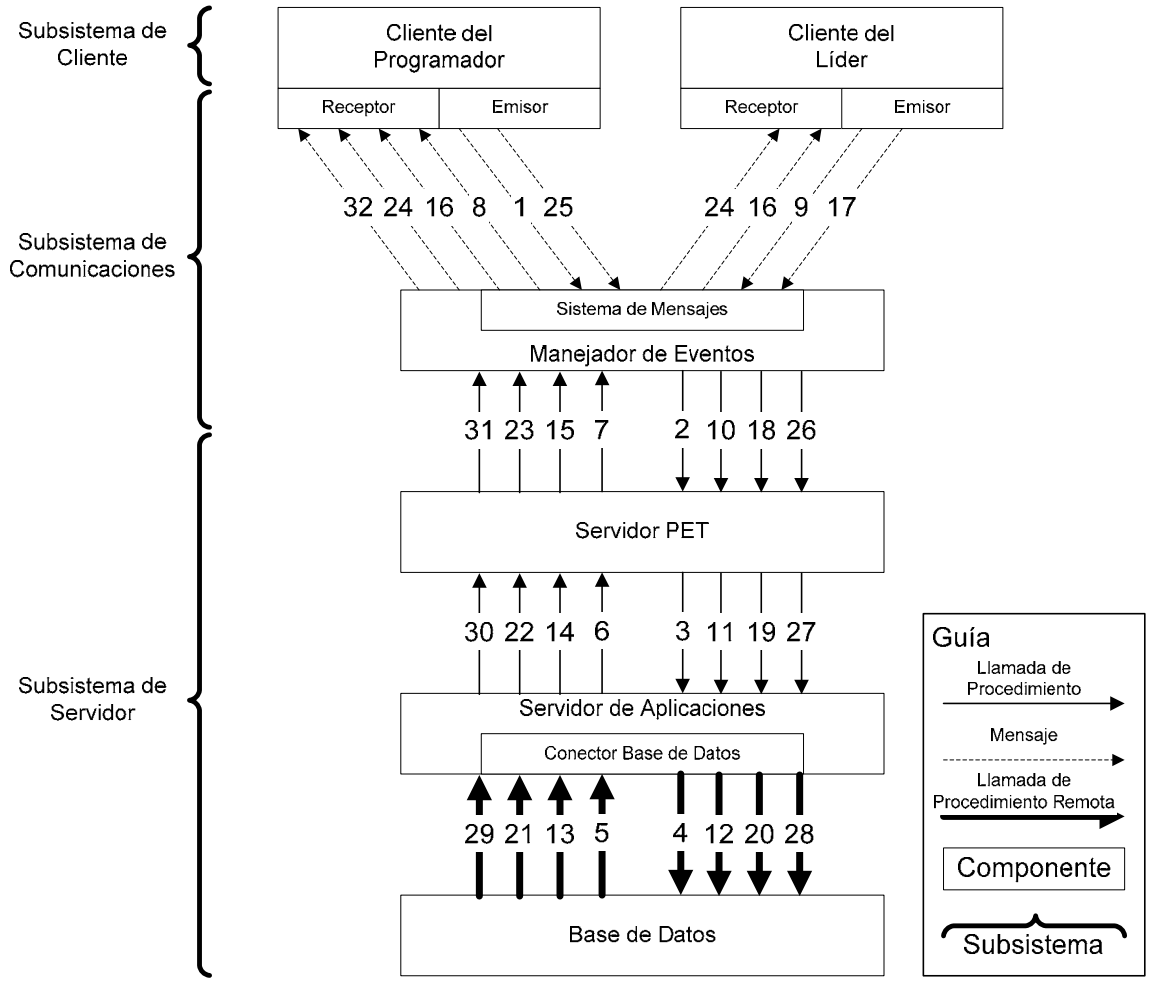

Diagrama 24 - Escenario 1: creación y asignación de un entregable de proyecto

Secuencia del Escenario:

El programador revisa su lista de tareas por hacer.

1. El programador abre su lista de tareas por hacer. Su cliente envía un mensaje de petición de objeto al manejador de eventos pidiendo la lista de tareas por hacer para el programador.
2. El manejador de eventos recibe el mensaje y pasa la petición al servidor PET.
3. El servidor PET, decodifica el mensaje y utiliza al servidor de aplicaciones para realizar la operación con la base de datos.
4. El servidor de aplicaciones mediante JDBC solicita la operación a la base de datos.
5. La transacción de la base de datos finaliza con éxito.
6. El servidor de aplicaciones regresa la información solicitada al servidor PET.
7. El servidor PET regresa la información al manejador de eventos con el cliente que originó la operación como destinatario.
8. El manejador de eventos envía de regreso la información al cliente del programador el cual la almacena y despliega.

El líder crea el entregable.

9. El líder llena los detalles del entregable. Su cliente envía un mensaje de petición de actualización de objeto, pidiendo la creación del objeto.
10. El manejador de eventos recibe el mensaje y pasa la petición al servidor PET.
11. El servidor PET, decodifica el mensaje y utiliza al servidor de aplicaciones para realizar la operación con la base de datos.
12. El servidor de aplicaciones mediante JDBC solicita los datos a la base de datos.
13. La transacción de la base de datos finaliza con éxito.
14. El servidor de aplicaciones regresa la información solicitada al servidor PET.
15. El servidor PET regresa la información al manejador de eventos con el cliente que originó la operación como destinatario.
16. El manejador de eventos envía una notificación de actualización a todos los clientes registrados. El cliente del líder al recibirla, almacena en el modelo el entregable.

El líder asigna el entregable al programador.

17. El líder asigna el entregable al programador. Su cliente envía un mensaje de petición de actualización de objeto, pidiendo la asignación del objeto.
18. El manejador de eventos recibe el mensaje y pasa la petición al servidor PET.
19. El servidor PET, decodifica el mensaje y utiliza al servidor de aplicaciones para realizar la operación con la base de datos.
20. El servidor de aplicaciones mediante JDBC solicita los datos a la base de datos.
21. La transacción de la base de datos finaliza con éxito.
22. El servidor de aplicaciones regresa la información solicitada al servidor PET.
23. El servidor PET regresa la información al manejador de eventos con el cliente que originó la operación como destinatario.
24. El manejador de eventos envía una notificación de actualización a todos los clientes registrados.
25. El cliente del programador compara la notificación con su modelo y se da cuenta de que la notificación es de uno de los objetos que le interesan, en este caso su lista de tareas por hacer. Envía un mensaje de petición de objeto al manejador de eventos pidiendo la lista de tareas por hacer para el programador.
26. El manejador de eventos recibe el mensaje y pasa la petición al servidor PET.
27. El servidor PET, decodifica el mensaje y utiliza al servidor de aplicaciones para realizar la operación con la base de datos.
28. El servidor de aplicaciones mediante JDBC solicita la operación a la base de datos.
29. La transacción de la base de datos finaliza con éxito.
30. El servidor de aplicaciones regresa la información solicitada al servidor PET.
31. El servidor PET regresa la información al manejador de eventos con el cliente que originó la operación como destinatario.
32. El manejador de eventos envía de regreso la información al cliente del programador el cual la almacena y despliega.

Escenario 2: Autentificación de usuario

En este escenario tenemos a dos usuarios y sólo uno de ellos tiene cuenta de usuario en el sistema. Ambos usuarios intentan registrarse y como es de esperarse, el sistema rechaza al que no tiene cuenta. Para efectos de este escenario, los usuarios van a ser referidos como “usuario” e “intruso”.

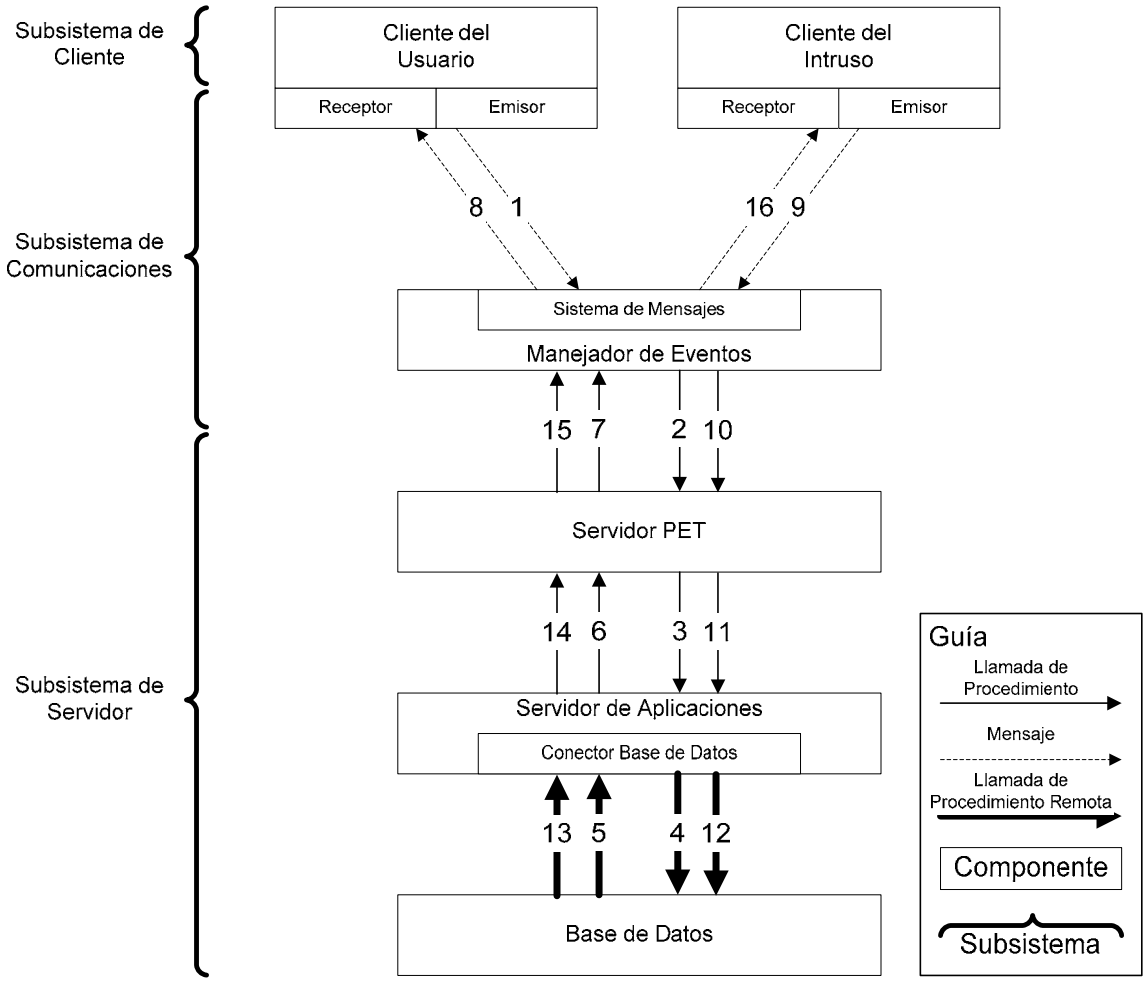

Diagrama 25 - Escenario 2 Autentificación de usuario

Secuencia del Escenario:

El usuario se registra.

1. El usuario se registra llenando los datos de su cuenta de usuario en la forma correspondiente. Su cliente envía un mensaje de petición de autentificación de usuario al manejador de eventos.
2. El manejador de eventos recibe el mensaje y pasa la petición al servidor PET.
3. El servidor PET, decodifica el mensaje y utiliza al servidor de aplicaciones para realizar la operación con la base de datos.

4. El servidor de aplicaciones mediante JDBC solicita la operación a la base de datos.
5. La transacción de la base de datos finaliza con éxito.
6. El servidor de aplicaciones regresa la información solicitada al servidor PET.
7. El servidor PET decide si el usuario debe de ser autentificado y regresa la respuesta al manejador de eventos con el cliente que originó la operación como destinatario.
8. El manejador de eventos registra al cliente como autentificado envía el mensaje de respuesta de autentificación de usuario como positiva al cliente del usuario.

El intruso intenta registrarse con datos falsos.

9. El intruso intenta registrarse llenando datos falsos en la forma correspondiente. Su cliente envía un mensaje de petición de autentificación de usuario al manejador de eventos.
10. El manejador de eventos recibe el mensaje y pasa la petición al servidor PET.
11. El servidor PET, decodifica el mensaje y utiliza al servidor de aplicaciones para realizar la operación con la base de datos.
12. El servidor de aplicaciones mediante JDBC solicita los datos a la base de datos.
13. La transacción de la base de datos finaliza con éxito.
14. El servidor de aplicaciones regresa la información solicitada al servidor PET.
15. El servidor PET decide si el usuario debe de ser autentificado y regresa la respuesta al manejador de eventos con el cliente que originó la operación como destinatario.
16. El manejador de eventos envía una notificación de actualización negativa del usuario al cliente del intruso.

Capítulo 5. Diseño

Objetivo

El documento de Diseño Detallado (DDD) contiene la descripción detallada de los componentes que van a ser traducidos en código. El objetivo es el de descomponer los requerimientos del sistema en entidades de diseño bien definidas que puedan ser traducidas en código de software en la fase de implementación. Es importante notar que todos los diagramas de este documento reflejan los componentes de software en su estado real, razón por la cual los nombres son en inglés. Es también digno de notar, que gran parte de estos diagramas fueron generados automáticamente por las aplicaciones de Ambiente Gráfico de Desarrollo, en este caso, Rational Rose de IBM y JBuilder de Borland.

Alcance

La IEEE define el contenido de este documento de la siguiente forma "*…Es una traducción de los requerimientos en una descripción de la estructura de software, componentes de software, interfaces y datos necesarios para la fase de la implementación*" En este documento también se describen las interacciones entre los componentes y los atributos de cada uno de ellos.

Los componentes en el caso de la fase de diseños son llamados "*Entidades de Diseño". "Las entidades de diseño son el resultado de la descomposición de los requerimientos de software del sistema. El objetivo es el de dividir el sistema en componentes separados que puedan ser considerados, implementados, cambiados y probados con un mínimo efecto en otras entidades.*" [IEEE std 1016-1998]

Referencias

Los documentos referenciados en la preparación del presente, o aquellos en los que se basa son:

IEEE. Recommended Practice for Software Design Descriptions [IEEE std 1016-1998]

Design Patterns: Elements of Reusable Object-Oriented Software; Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides; Addison' Wesley 1995

Wiley Computer Publishing. EJB Design Patterns: Advanced Patterns, Processes, and Idioms; Marinescu, Floyd

Wiley Computer Publishing. The Art of Computer Systems Performance Analysis; Jain, Raj

5.1 Descripción de los Subsistemas

Descripción General

El sistema está compuesto de tres grandes grupos componentes o subsistemas: servidor, cliente y comunicaciones. El servidor y el cliente se comunican entre sí utilizando invocación implícita en la forma de mensajes. El subsistema de comunicaciones es responsable de implementar el servicio de mensajes para clientes y el servidor, así como de servir de autoridad central para los registros, hace las funciones de manejador de eventos.

Subsistemas

El subsistema de servidor es donde todos los objetos de negocio y gran parte de la lógica de negocio residen. Los objetos de negocio están representados por los "Entity Beans" y la lógica de negocio está encapsulada en los "Session Beans". Debido a que se usó la arquitectura de "Enterprise Java Beans" (EJB, por sus siglas en inglés), el proceso de transacciones, la seguridad, persistencia y administración de recursos son solucionados por el contenedor EJB. El servidor nunca inicia ninguna comunicación con los clientes directamente sino mediante el manejador de eventos.

El subsistema de cliente según especificaciones, tiene una arquitectura MVC. Está construido bajo las siguientes premisas:

- La lógica de negocio es verificada en el servidor para eliminar la posibilidad de corrupción de la base de datos
- Las clases del modelo y el controlador son desarrolladas siguiendo el patrón "Singleton" para asegurar que sólo existe una instancia de ambos en cada cliente
- Los mismos objetos de datos que son usados en el servidor se usan en el cliente para mantener una copia de la información en la cual este tiene interés
- El código responsable de las comunicaciones esta en subsistema de comunicaciones
- La interfaz de usuario utiliza una presentación tipo árbol para permitir una fácil transición entre y desplegado de funciones relacionadas

El subsistema de comunicaciones provee de servicios y define interfaces para las comunicaciones entre el cliente y el servidor. El servicio central de este subsistema es el de manejador de eventos con el cual funciona como estación central de mensajes. El manejador de eventos filtra y procesa los mensajes antes de que sean redirigidos a sus destinatarios. También es responsable de hacer anuncios generales. El subsistema provee de objetos de comandos como lenguaje de comunicación a los clientes y se conecta directamente con el servidor. El medio por el cual son transportados estos comandos es mediante mensajes.

Los objetos de datos (DO, por sus siglas en inglés) son especificados por separado del resto de los subsistemas ya que son usados por todos ellos. Los DO definidos son de dos tipos: de dominio y a la medida. Los DOs de dominio reflejan atributos de los objetos de negocio y son usados como medio principal de envío y recepción de información por una

entidad individual del servidor. Los DOs a la medida contienen información acerca de uno o más objetos de negocio y son usados para propósitos de comunicación más específicos. Los DOs son creados individualmente en el lado del cliente y en el lado del servidor mediante una clase ayudante que funciona como fábrica. La fábrica de DOs contiene toda la lógica necesaria para crear y poblar estos DOs y es usada por las fachadas de sesión del servidor.

Interfaces

Los subsistemas de cliente y servidor utilizan como interfaces mensajes para comunicarse definidas por el subsistema de comunicaciones. El cliente también interactúa con los usuarios razón por la cual tiene que tener una interfaz de usuario. El servidor por su parte, interactúa con una base de datos como almacén permanente. Esta última interfaz es implementada usando al contenedor de EJB mediante su modalidad de "Container Manager Persistente" (CMP) y un "driver" JDBC.

5.2 Subsistema de Servidor

Diseño de Paquetes

El subsistema de servidor contiene dos paquetes: servicios y entidades. El paquete de servicios contiene toda la lógica de negocio mientras que el de entidades los objetos de negocio. En el servidor se usaron EJB para tomar ventaja de todos los servicios que provee el contenedor de EJB tales como control de transacciones, seguridad y persistencia de datos entre otros. Se utilizaron tres capas de EJB:

- Capa de Negocio: Objetos de negocio implementados como “Entity Beans”
- Capa de Servicios: Lógica de negocio implementada como “Session Beans”
- Capa de Mensajes: Receptor de eventos implementado como “Message Driven Bean”

De estas tres capas, la última pertenece al subsistema de comunicaciones por lo que será descrita en la sección correspondiente. La segunda capa, la de servicios, esta implementada siguiendo el patrón de fachada de sesión

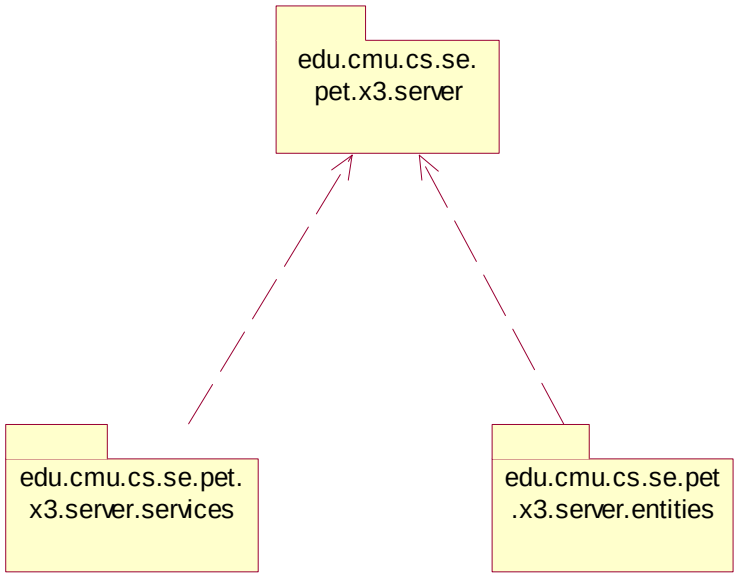

Diagrama 26 - Diagrama de paquetes del subsistema de servidor

El paquete de servicios contiene la lógica de los servicios que administran lo siguiente: entregables, ciclos de vida, desempeño profesional predecible, proyectos, cuentas de usuarios y el decodificador de comandos. Todos éstos son implementados mediante “Session Beans”. Este paquete también contiene la fábrica de DOs que es una clase normal de java.

El paquete de entidades contiene los objetos de negocio implementados usando “Entity Beans”, tales como ciclos de vida, fases, proyectos…

Una analogía útil para entender y describir la diferencia entre lógica y objetos de negocios y de cómo se relacionan éstos con los comandos es la siguiente. Se puede pensar en verbos y sustantivos tales como "crear ciclo de vida" con lo que crear es la acción y ciclo de vida el sustantivo y la frase como un comando. De esta forma se entiende que debe de haber un servicio en la lógica de negocio que sirva para crear ciclos de vida, y que tales sean un tipo de objetos de negocio.

Diseño de Clases

Fachada de Sesión

Se utilizó una fachada de sesión implementada con "Session Beans"para encapsular la lógica de negocio en esta capa, así como para aislar a los objetos reales de negocio del resto del sistema y par a servir como única interfaz a éstos. Todas las llamadas de método realizadas con los objetos de negocio son hechas por las fachadas de sesión con lo que se logran los objetivos descritos al igual que se minimiza la información que es necesario transmitir desde el cliente para realizar cualquier acción. Se utilizaron seis "Stateless Session Beans" que encapsulan toda la lógica de negocio: "LifeCycleManagerEJB", "ProjectManagerEJB", "DeliverableManagerEJB", "PPPManagerEJB", "UAManagerEJB" y el "CommandDecoderEJB". Cada uno de los primeros cinco solo interactúa con un cierto grupo de objetos de negocio. El sexto es un poco diferente ya que es utilizado como interfaz con el subsistema de comunicaciones.

Clases

A continuación se detallan los componentes del servidor. Primero se da una vista general de los objetos de negocio, y después detalles de algunos grupos de éstos que están relacionados entre si. Finalmente se sigue un proceso similar con los componentes que encapsulan la lógica de negocio. Cabe señalar que todos los componentes que son EJBs tienen estas siglas en su nombre y que en realidad están formados por una clase y dos o cuatro interfaces.

Otra consideración que se tomó en este diseño, fue la de crear un componente raíz del que toda la jerarquía de éstos dependa. Esto se hizo para simplificar la búsqueda inicial y hacer la estructura más semejante a un árbol. Van a existir solamente cuatro instancias de este componente y van a servir para contener referencias a los conjuntos de objetos que dependen directamente de el. Estas instancias son las siguientes:

Nombre	Hijos
Root	LifeCycleSet, ProjectSet, y UserAccountSet
LifeCycleSet	LifeCycle 1, LifeCycle 2, …
ProjectSet	Project 1, Project 2, …
UserAccountSet	UserAccount 1, UserAccount 2, …

Una de las más importantes ventajas de este esquema es que en cualquier momento se pueden agregar nuevos elementos a esta estructura sin necesidad de realizar muchos cambios.

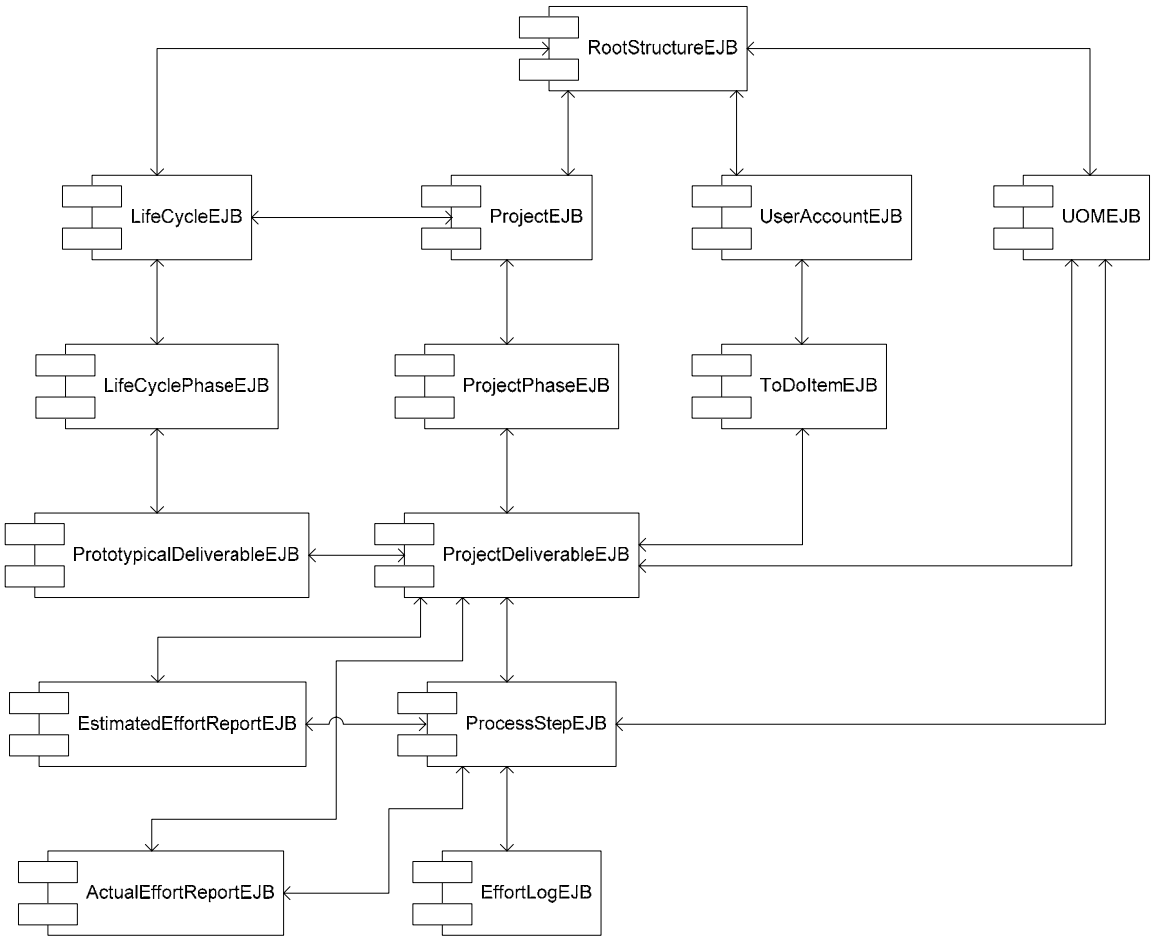

Diagrama 27 - Capa de negocio: objetos de negocio

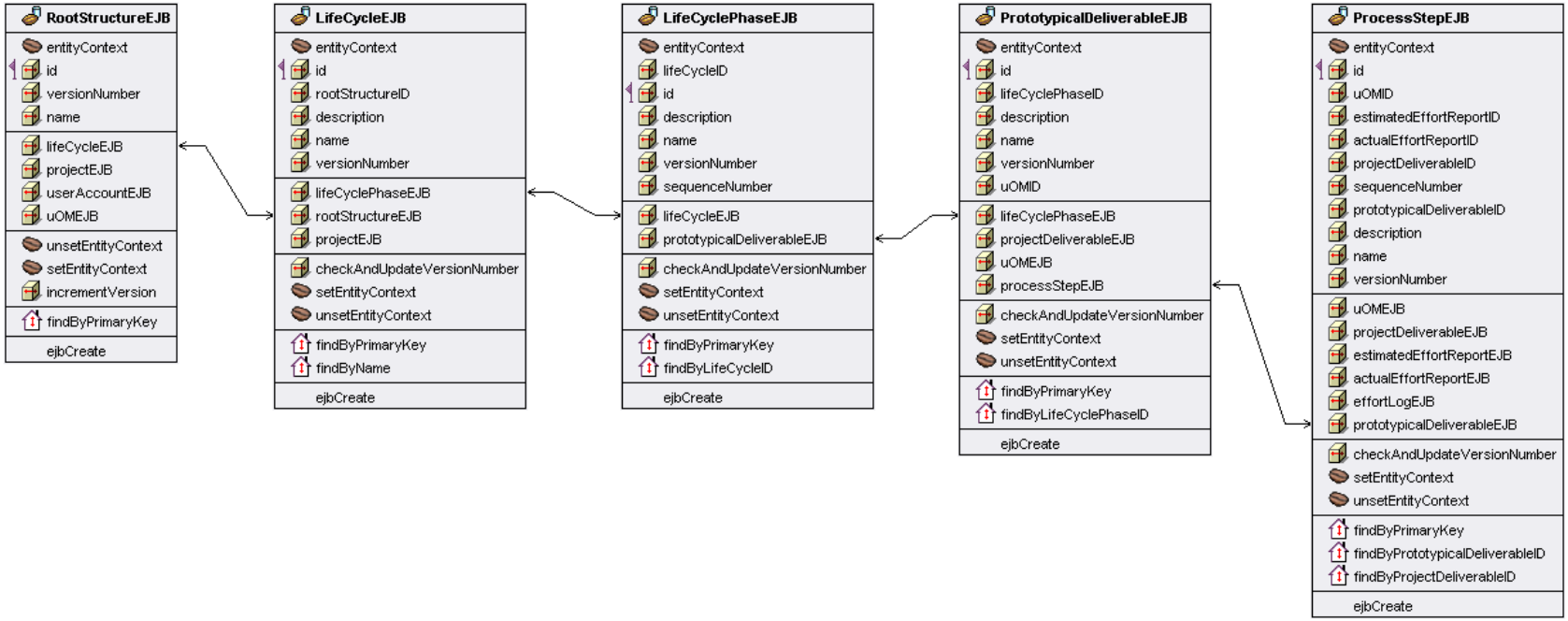

Diagrama 28 - Capa de negocio: objetos de ciclo de vida

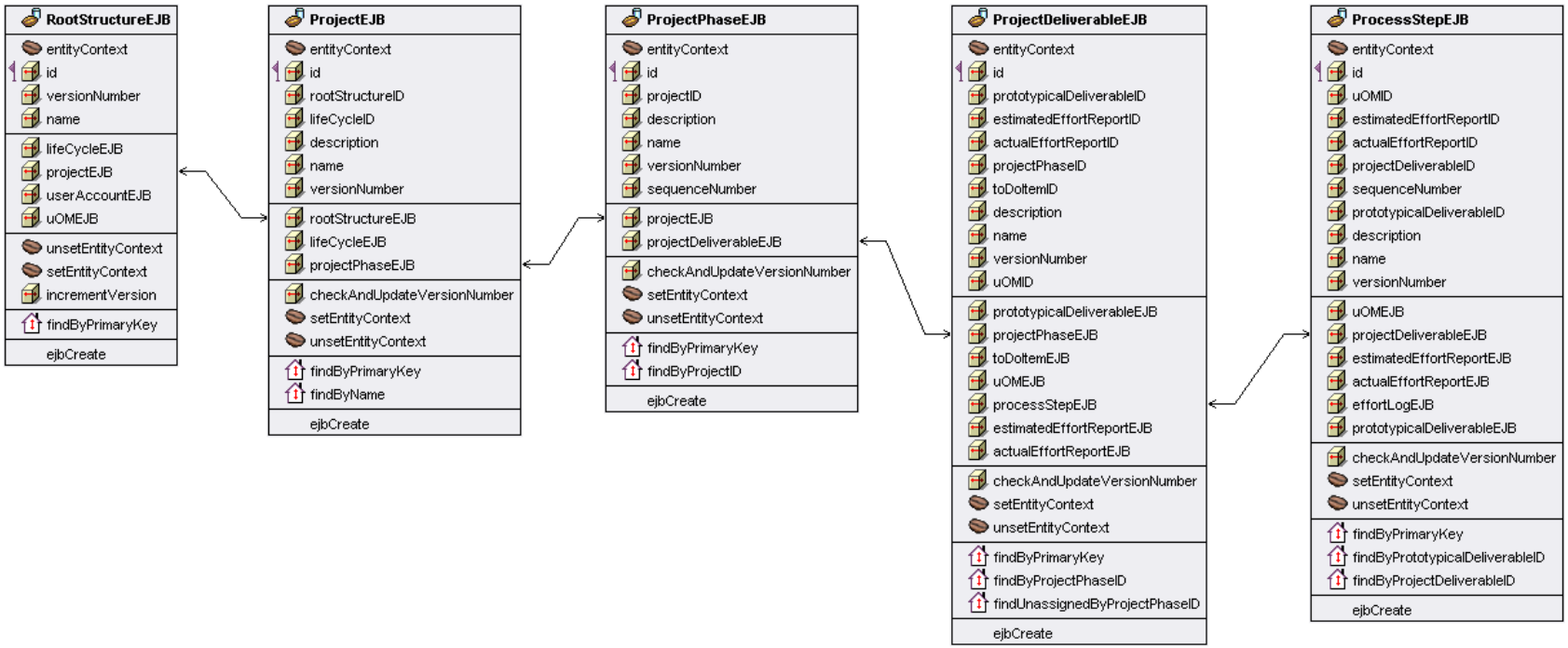

Diagrama 29 - Capa de negocio: objetos de proyecto

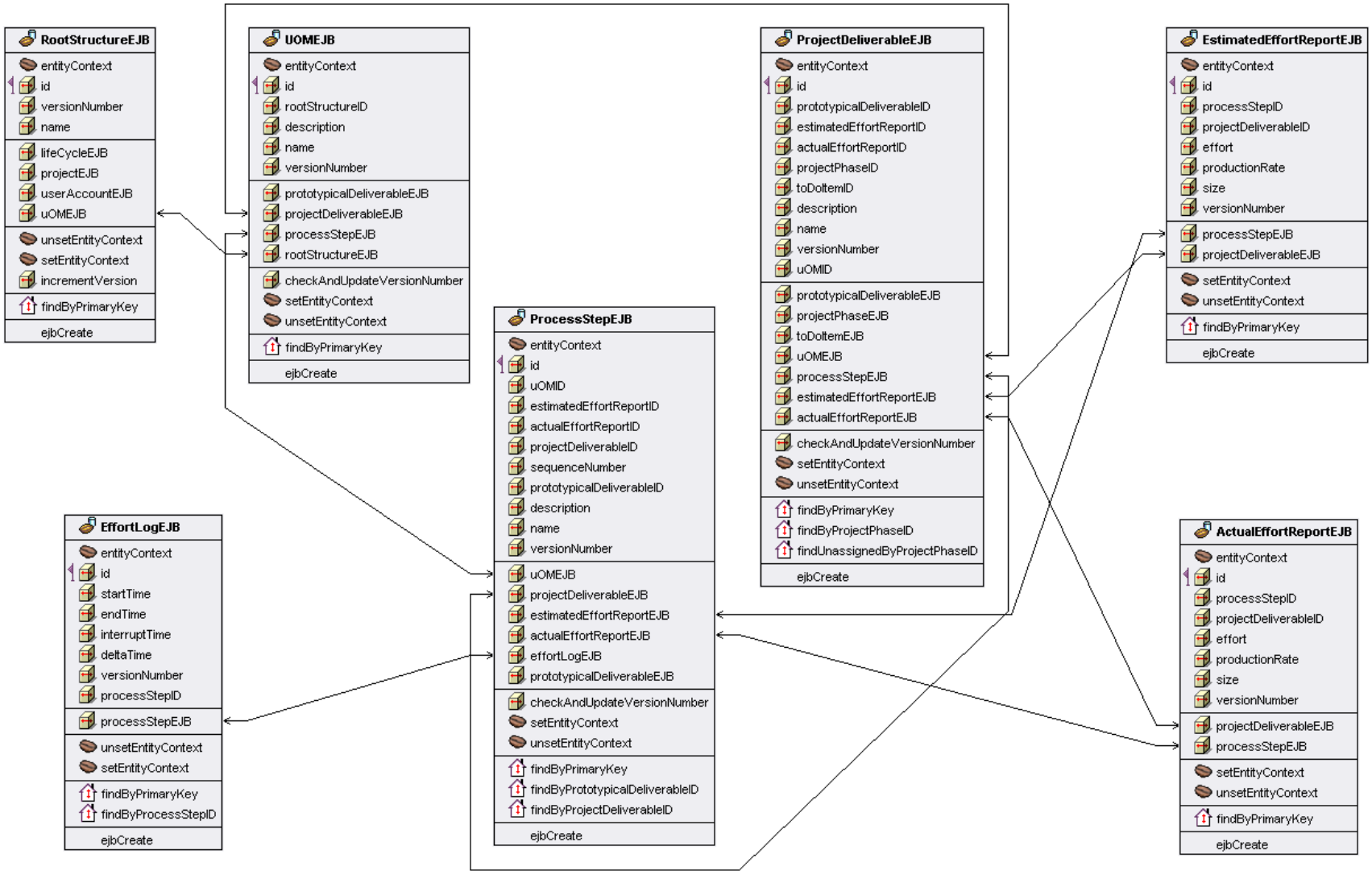

Diagrama 30 - Capa de negocio: objetos de esfuerzo

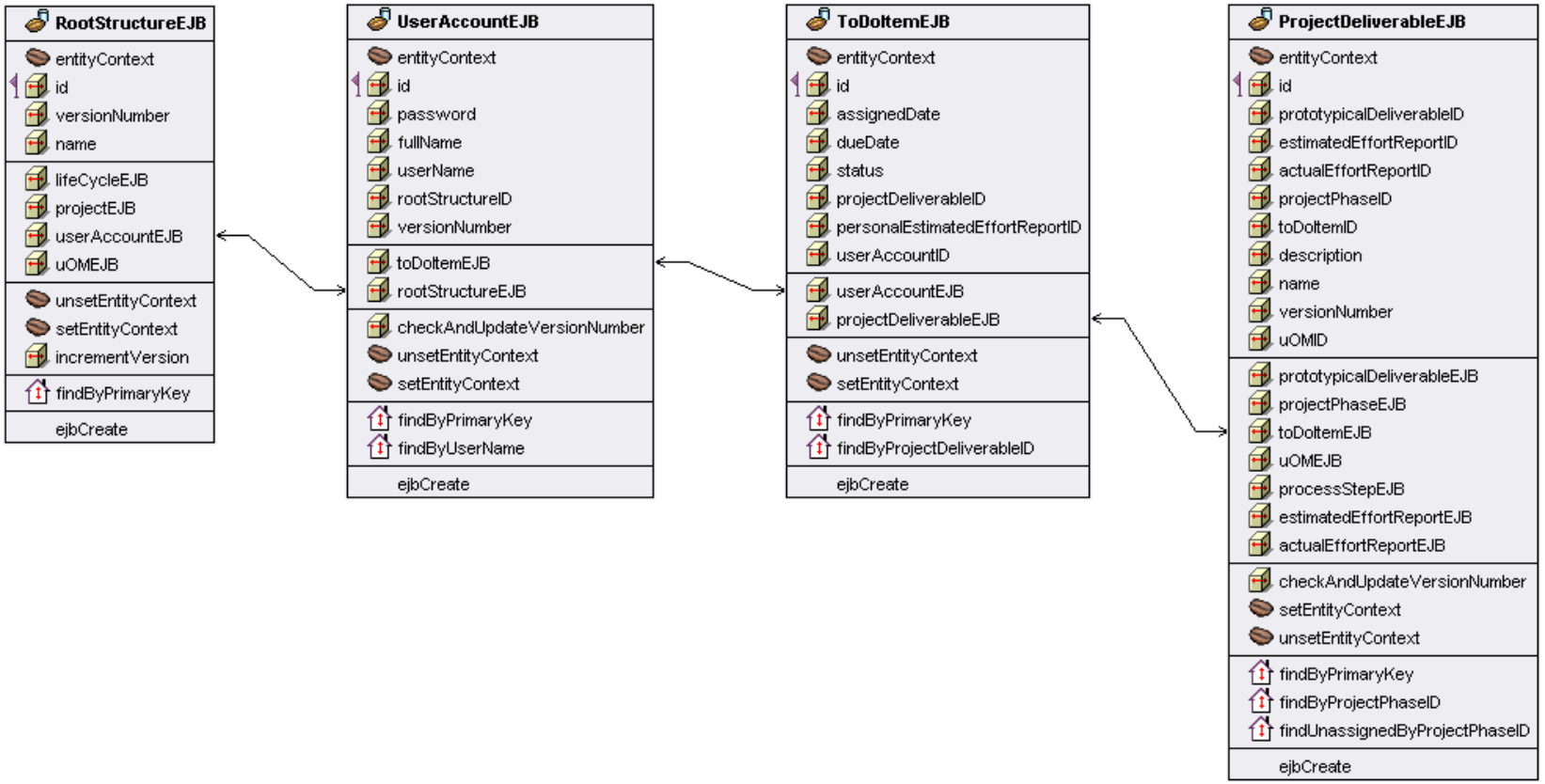

Diagrama 31 - Capa de negocio: objetos de cuenta de usuario

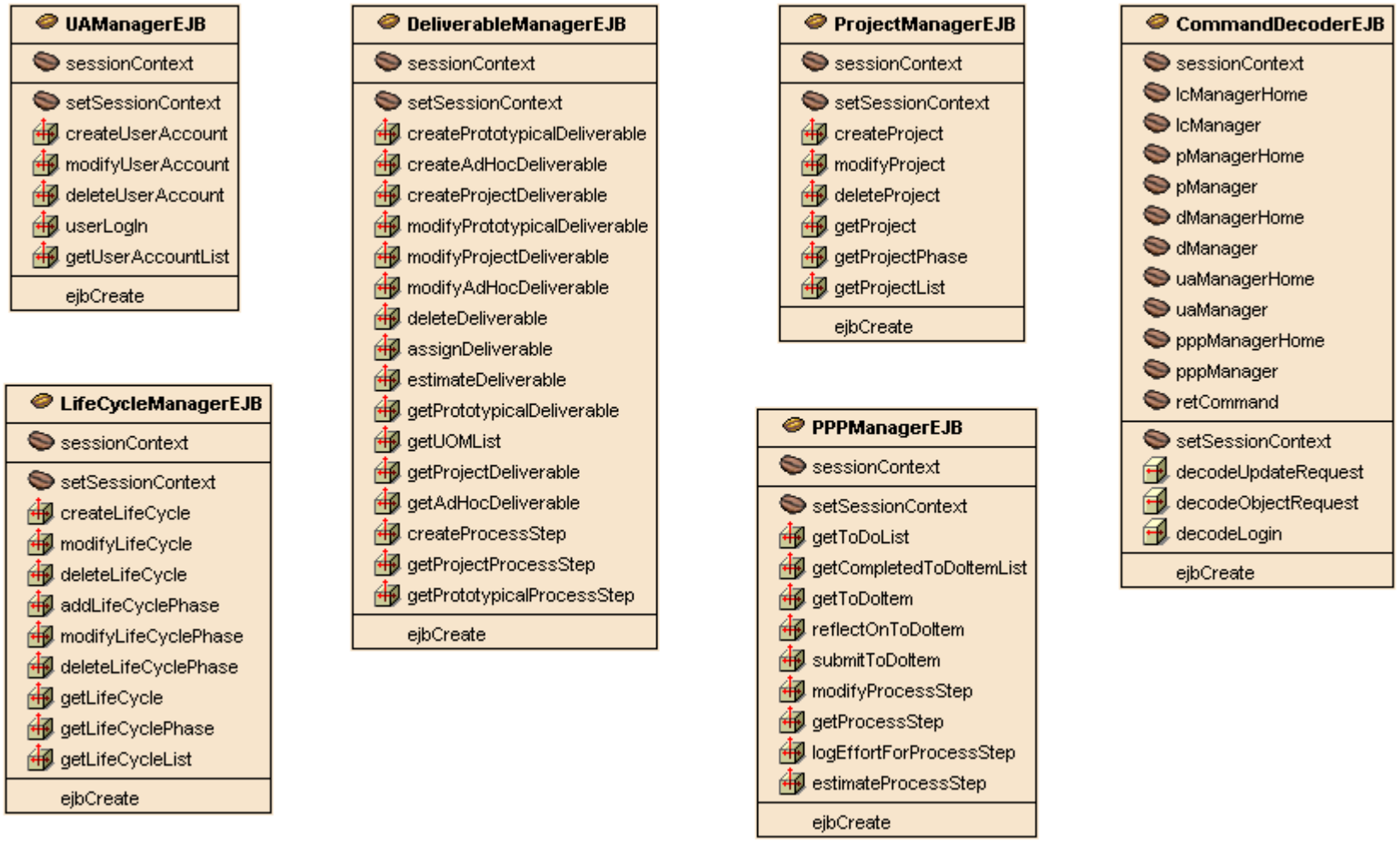

Diagrama 32 - Capa de servicios: fachadas de sesión

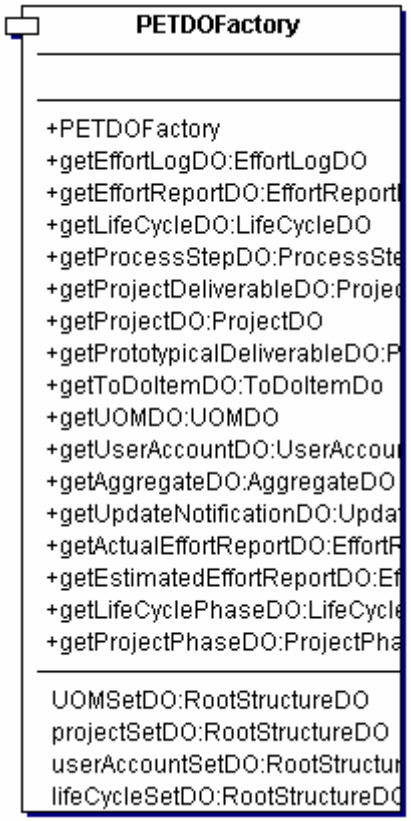

Diagrama 33 - Capa de servicios: fábrica de objetos de datos

Concurrencia

Todos los objetos de negocio tienen un número de versión. Los comandos transmitidos en toda operación contienen en el objeto de datos que llevan también un número de versión. Con estos números, se puede verificar que en cualquier petición de actualización de datos, ésta no sea solicitada sobre datos obsoletos. En caso de que los números de versión de ambos objetos sean los mismos, el cambio va a ser procesado y la versión incrementada.

Existe un escenario en el cual aun usando los números de versión se pueden tener problemas de concurrencia. Considérese el caso de que dos clientes quieran hacer una actualización de datos casi simultáneamente de manera que cuando los datos de la primera están siendo procesados llega la petición de la segunda. En este caso se da que el número de versión aun no es incrementado por lo que la segunda petición también puede ser procesada. La solución a este caso de problemas de concurrencia está en el uso de control de transacciones por parte del contenedor, de manera que en cuando se termine de procesar la primera actualización, a la segunda se le aplique una operación de "rollback".

Interfaces

En este sistema, el servidor interactúa únicamente con el subsistema de comunicaciones. Todas las peticiones van a ser recibidas por el manejador de eventos y canalizadas hacia el servidor mediante el decodificador de comandos en la capa de servicios. La vía inversa de comunicación se da mediante los regresos de los métodos de este componente.

Respecto a las interfaces de los componentes de este subsistema, es importante mencionar que como lo señalan los diagramas, los "Entity Beans" que componen los objetos de negocio sólo tienen interfaces locales, es decir para comunicación dentro de una sola JVM. Por otra parte los "Session Beans" que forman la capa de servicios tienen interfaces tanto locales como remotas, estas últimas para permitir que el subsistema de comunicaciones resida en una diferente JVM y posiblemente en una diferente computadora.

5.3 Subsistema de Cliente

Diseño de Paquetes

El subsistema de cliente contiene tres paquetes: Modelo, Vista y Controlador. El paquete de modelo contiene una copia temporal de los objetos que están siendo utilizados por esa instancia del cliente. El paquete de Vista contiene las clases que implementan toda la interfaz de usuario. Finalmente, paquete de Controlador contiene la clase que sirve de repositorio central de la lógica de funcionamiento de todo el subsistema.

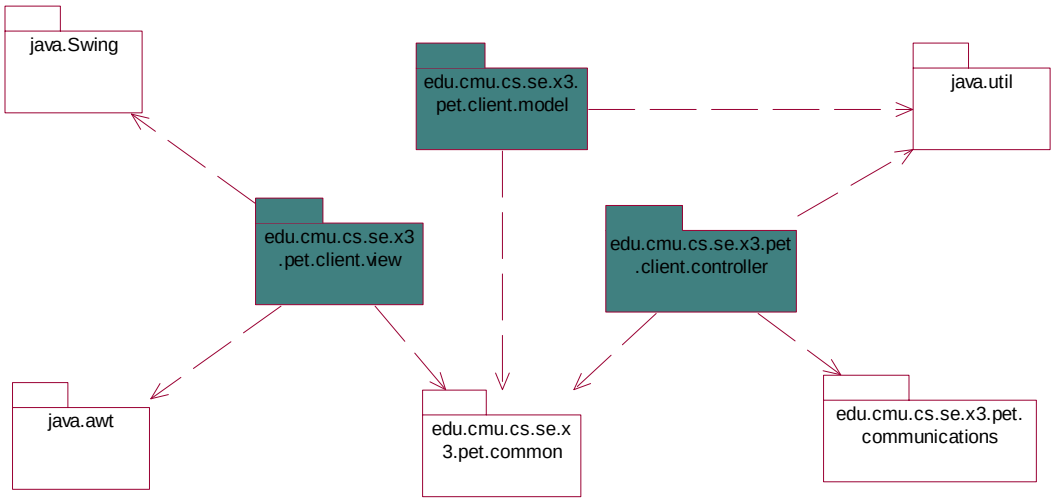

Diagrama 34 - Diagrama de paquetes del subsistema de cliente

En el diagrama de paquetes del Diagrama 34, los paquetes de color oscuro son los desarrollados como parte de este subsistema, el resto son ya sea de otros subsistemas o utilerías incluidas como parte del kit de desarrollo de Java.

Diseño de Clases

Este subsistema está compuesto por clases de Java comunes, es decir, no utiliza ningún tipo de componente como los EJBs del servidor. Los paquetes de controlador y modelo contiene solamente una clase pero aún así se considero necesario mantenerlas en paquetes separados ya que no sólo tienen funcionalidades muy distintas sino que también así se hace énfasis en la separación de los tres elementos según la arquitectura de Modelo-Vista-Controlador. Por su parte, la vista esta compuesta de múltiples clases que implementan las interfaces de cada caso de uso según se han descrito a lo largo de todo este proyecto.

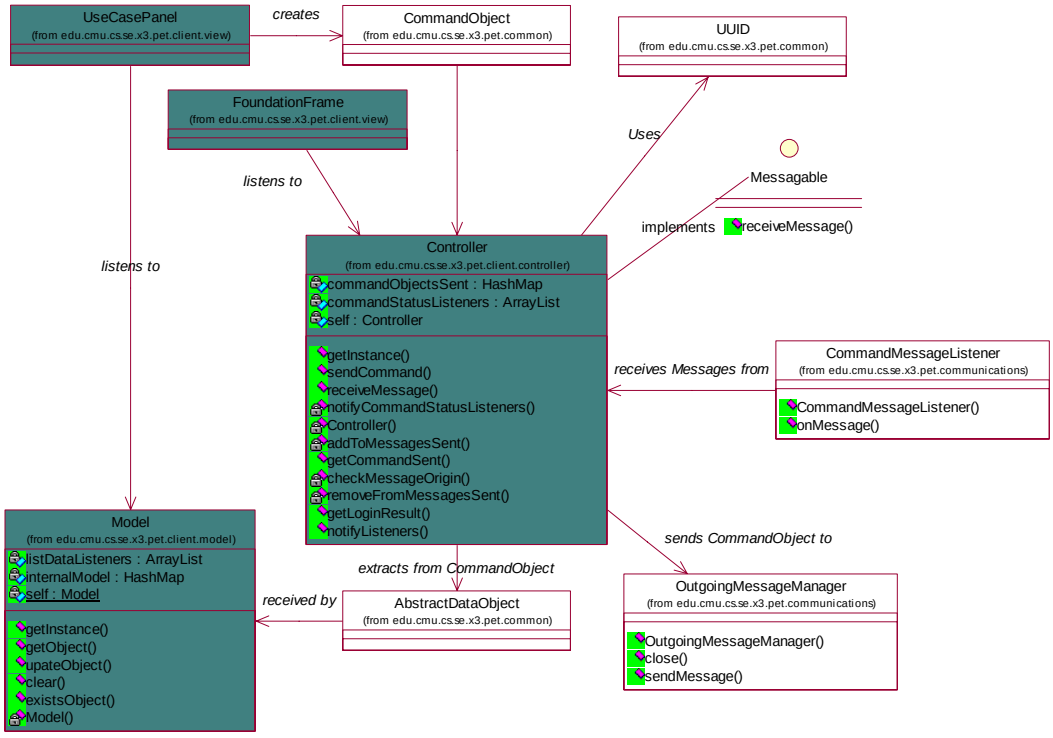

Diagrama 35 - Diagrama de clases del subsistema de cliente

En el diagrama de paquetes del Diagrama 35, los paquetes de color oscuro son los desarrollados como parte de este subsistema, el resto son ya sea de otros subsistemas o utilerías incluidas como parte del kit de desarrollo de Java. También en dicho diagrama, la clase llamada “UseCasePanel” realmente se refiere a múltiples clases que son detalladas más adelante.

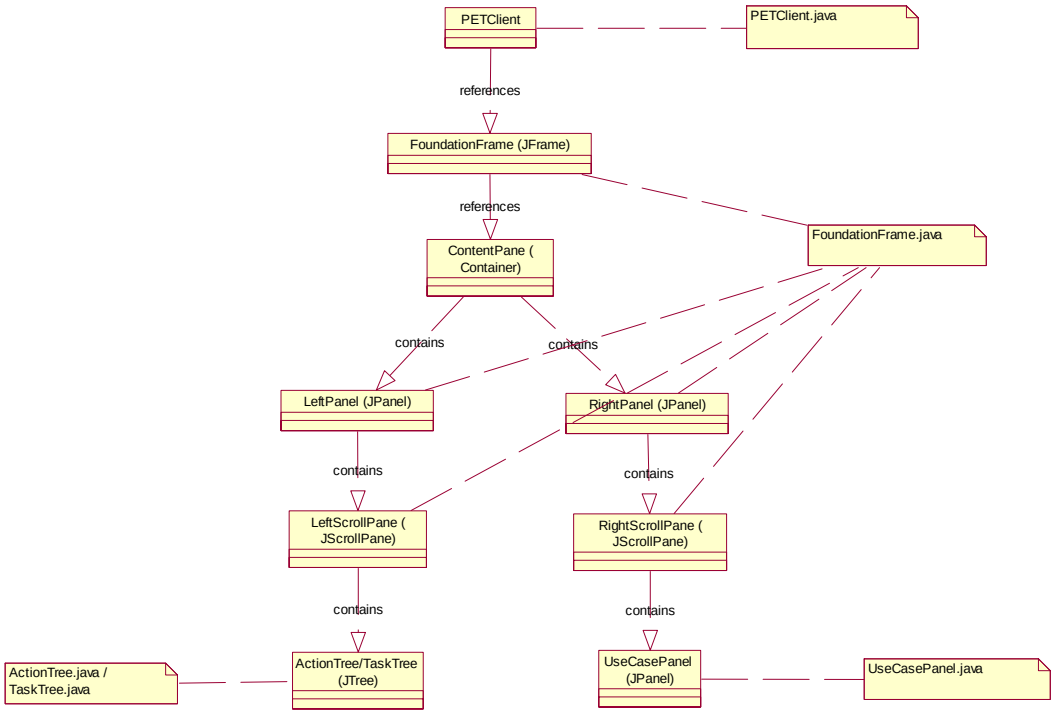

Diagrama 36 - Diagrama de clases de la interfaz de usuario

La interfaz de usuario está compuesta de cinco tipos diferentes de clases:

Clase	Descripción
PETClient.java	Sirve como clase inicial del cliente. Carga la "FoundationFrame" y la activa.
FoundationFrame.java	Todos los "frames" son contenido dentro de este. Esta implementada como "Singleton" y nunca es eliminada cuando el cliente termina su ejecución.
ActionTree.java	Poblada con las acciones permitidas y organizada por tipo de objeto.
TaskTree.java	Poblada con las listas de tareas por hacer y de tareas completadas.
UseCasePanel.java[7]	Dependiendo de la selección en "ActionTree" o "TaskTree", despliega la información relevante o las formas para captura de información.

Tabla 23 – Tipos de clases de la interfaz de usuario

Modelo

El modelo sirve como repositorio temporal de los objetos que son relevantes a la vista que está siendo desplegada por el cliente en todo momento. Una ves que éstos ya no son

[7] UseCasePanel se refiere a diversos JPanel específicos a cada caso de uso. Los paneles están especificados en el diagrama de flujo de la interfaz de usuario

requeridos, es decir se ha cambiado la vista, los objetos guardados en el modelo pueden ser eliminados y reemplazados por los que a partir de ese momento son relevantes.

El modelo está compuesto por una clase que sigue el patrón "Singleton" y todos sus métodos de acceso están declarados como sincronizados para asegurar que sólo un hilo de procesamiento tenga acceso al modelo. Los datos internos son mantenidos por medio de un "HashMap" referenciados por su identificador único universal (UUID, por sus siglas en inglés).

Controlador

El controlador funciona como principal conducto entre el subsistema y el manejador de eventos. Implementa la interfaz "Messageable" para poder recibir y enviar mensajes. Es también responsable de mantener un registro de los mensajes originados desde el subsistema para poder notificar al usuario en caso de falla o error en alguno de éstos.

El controlador, al igual que el modelo es una clase "Singleton" que mantiene los datos internamente por medio de un "HashMap" referenciados por medio del identificador de mensaje.

Vista

La vista esta compuesta por una serie de clases que implementan la interfaz de usuario (UI, por sus siglas en inglés). En respuesta a las acciones del usuario, ésta crea objetos de comando para ser pasados al controlador que a su vez envía al manejador de eventos. La vista registra componentes de "Swing" como "listeners" en el modelo para detectar y reaccionar a cualquier cambio en los datos contenidos en éste. También registra "listeners" en el controlador para recibir resultados de los comandos enviados. Más adelante se detallan las pantallas como fueron diseñadas.

Concurrencia

Debido a que la comunicación con el servidor (mediante el manejador de eventos) es asíncrona, pueden ocurrir casos de problemas de accesos concurrentes de información. Una ves que se envía algún comando al servidor, el usuario puede seguir utilizando el sistema o incluso enviar otro comando sin necesidad de esperar por una respuesta, esto funciona mediante el uso de hilos múltiples de procesamiento. Para evitar que esto cause problemas, todos los métodos de acceso del modelo y del controlador son declarados como sincronizados. De esta manera y debido a que ambos tienen solamente una instancia, se puede asegurar que solamente un hilo tiene acceso a la información contenida en estos componentes.

Interfaces

Mensajes

El cliente usa el paquete "Messaging" del subsistema de comunicaciones para interactuar con el resto del sistema. Más información al respecto en la sección correspondiente a dicho subsistema (Mensajes).

Interfaz de Usuario

Especificaciones Generales

Característica	**Valor**
Kit de herramientas gráficas	Java Swing & Java AWT
Tamaño por inicial	1024x768
Posibilidad de cambiar el tamaño	Si
Estilo	Windows (excepto los "JTree" que no tienen íconos)
Letra	MS Sans Serif
Posición inicial	Centrada

Tabla 24 - Especificaciones generales de la interfaz de usuario

Descripción General

Todas las pantallas catalogadas como de caso de uso con excepción de la de "login" tienen dos paneles, izquierdo y derecho. El izquierdo contiene un árbol de selección, mientras que el derecho contiene ya sea la información pertinente a la opción seleccionada o la forma de captura de información.

Sin importar el caso de uso que este desplegado, en todo momento el nombre del usuario, así como la modalidad en la que esta usando el sistema son desplegados. La modalidad puede ser cambiada de desarrollador a administrador y viceversa por medio de un componente tipo "drop down combo box".

El árbol del panel izquierdo, dependiendo de si el usuario esta usando los privilegios de desarrollador o administrador va a ser poblado con los objetos pertinentes. Si el usuario esta en modo de desarrollador, éste va a ser poblado con su lista de tareas por hacer, mientras que como administrador con las acciones válidas.

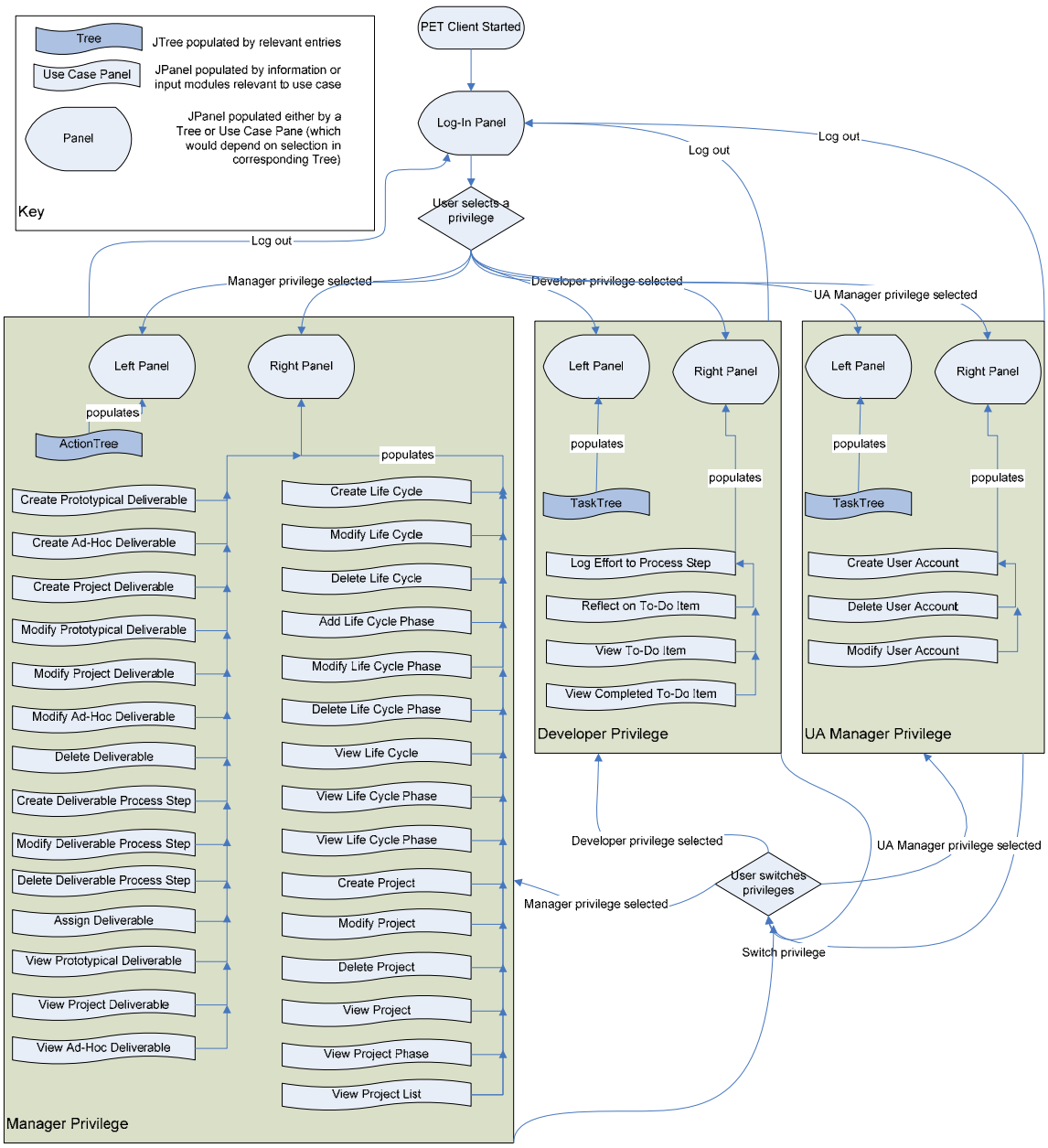

Diagrama 37 - Diagrama de flujo de la interfaz de usuario

Respuestas de Comandos

Cualquier comando enviado al servidor resulta en una respuesta, ya sea de éxito o de fracaso, pero siempre son respondidos. Si el comando es exitoso, la barra de estatus notifica al usuario, mientras que en el caso de fracaso, una ventana de diálogo tipo “Pop-up” es desplegada informando al usuario del problema y del curso de acción. Después de enviar cualquier comando, el usuario puede continuar su uso del sistema mientras que el cliente espera por la respuesta.

Interfaces para “Log in” y “Log out”

Al iniciar el cliente se despliega un panel de “Login” que le pregunta al usuario por su nombre de usuario y contraseña. En el caso de que el usuario sea reconocido por el servidor, este panel es sustituido por el panel inicial dependiendo de la modalidad en la que el usuario seleccione trabajar. Un botón de “Logout” esta disponible en la esquina superior derecha, que al ser presionado, ejecuta dicha operación y despliega el panel de “Login”.

Ventanas de Diálogo tipo “Pop-up”

Debido a la necesidad de reducir el amontonamiento de la información en la interfaz de usuario, y para hacer más eficiente el flujo de datos con el usuario, este tipo de ventanas son desplegadas solamente cuando se dan alguna o varias de las condiciones siguientes:

- El usuario intenta de cambiar de caso de uso de tarea por hacer sin enviar la información en la que esta trabajando.
- El usuario intenta enviar números inválidos (negativos o letras) en campos de captura numérica.
- El cliente recibe un resultado de fracaso del servidor en respuesta a un comando enviado.
- El cliente descubre que ha sido desconectado del manejador de eventos.

5.4 Subsistema de Comunicaciones

Diseño de Paquetes

El subsistema de comunicaciones contiene tres paquetes. El paquete de comandos contiene todas las clases que representan acciones que los clientes pueden solicitar al servidor. El paquete de mensajes contiene las clases que son utilizadas para enviar y recibir mensaje por parte de los clientes. Finalmente, el paquete de Administrador de Eventos, contiene al administrador de eventos y sus clases ayudantes que es quien se encarga de enviar, recibir y procesar los mensajes. Este último esta implementado como un tipo de EJB llamado "Message Driven Bean" (MDB, por sus siglas en inglés) y que implementa las funciones de receptor y transmisor de mensajes.

Al igual que en el caso del servidor, la decisión de usar EJB fue debido a la gran cantidad de ventajas que existen al usar el servidor de aplicaciones y el contenedor de EJB que utilizan éstos. En este caso, todo el sistema de mensajes esta implementado en el servidor de aplicaciones, con lo que solamente es necesario habilitarlo y configurarlo para poder hacer uso programático de el.

Como fue mencionado en la sección del servidor, el manejador de eventos funciona como una capa aislante entre el servidor y el cliente e implementa una fachada de mensajes.

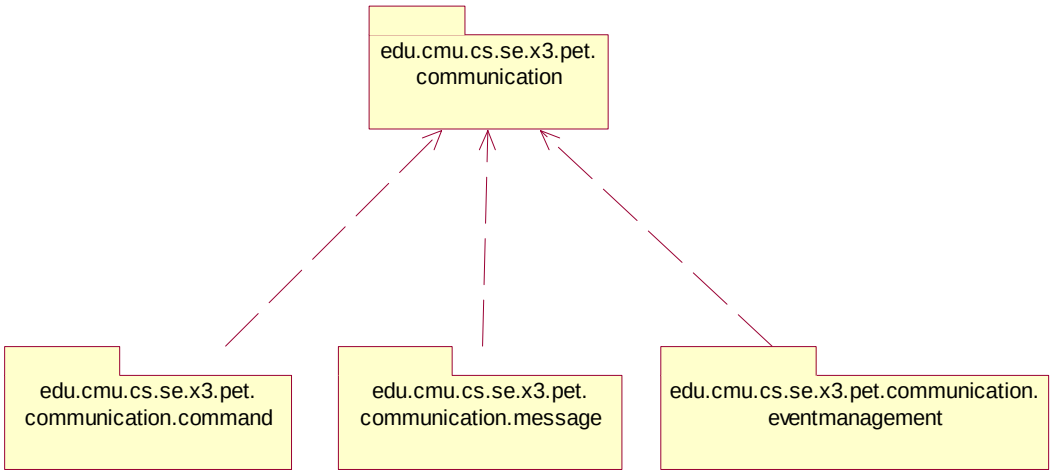

Diagrama 38 - Diagrama de paquetes del subsistema de comunicaciones

Diseño de Clases

Comandos

Las clases del paquete de comandos son usadas para especificar peticiones del cliente y respuestas del servidor. Encapsulan objetos de datos y son encapsuladas a su vez por objetos de mensaje mientras son enviados por la red. Algunas de las clases de estos comandos no contienen ningún atributo o método, ya que sus peticiones o respuestas son triviales, y el nombre de la clase es suficiente para especificar el motivo del comando.

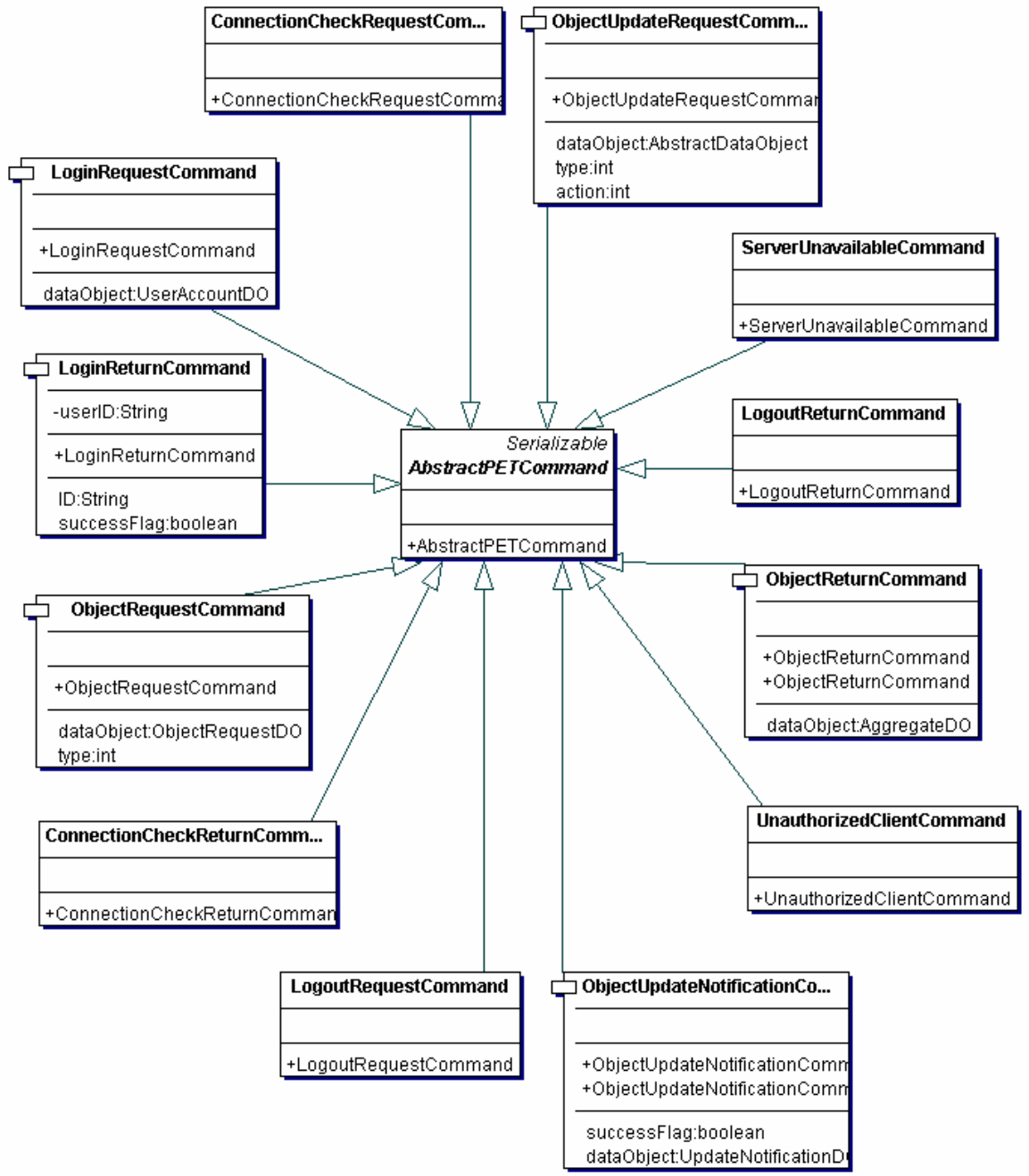

Diagrama 39 - Diagrama de clases del paquete de comandos

Mensajes

El paquete de mensajes contiene solamente dos clases: “OutgoingMessageHandler” e “IncomingMessageManager”. La primera es usada por el cliente para establecer la conexión necesaria para enviar mensajes a cualquier destino. La segunda es usada también por el cliente para establecer el enlace para recibir mensajes de cualquier destino.

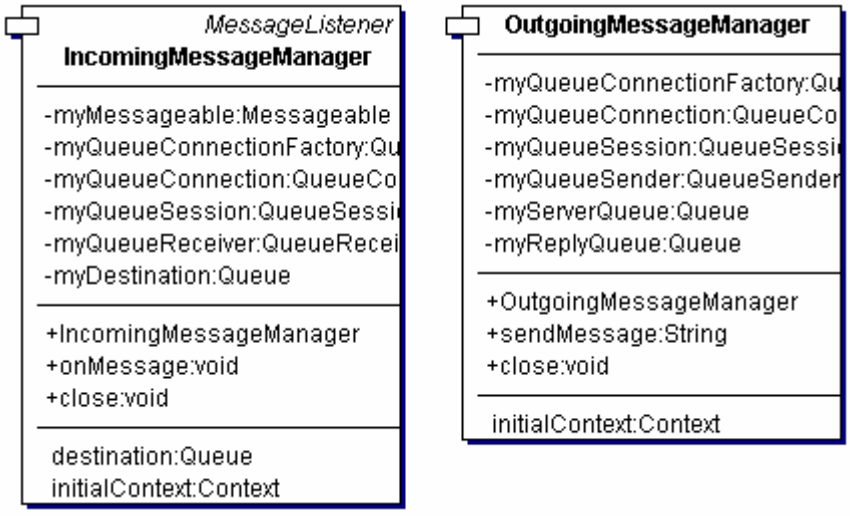

Diagrama 40 - Diagrama de clases del paquete de mensajes

Manejador de Eventos

El manejador de eventos contiene cuatro clases: "ConnectionChecker", "ConectionManager", "RegistrationManager" y "EventReceiverMBDBean". Las primeras tres clases implementan una tarea recurrente que cada cierto tiempo realiza una verificación de los clientes conectados en un hilo de procesamiento independiente. Periódicamente envían mensajes de verificación de conexión a los clientes y si éstos no responden en un periodo específico, los eliminan de la lista de clientes conectados.

La última clase, la de "EventReceiverMDBBean" implementa el manejador de eventos propiamente, es decir es la que se encarga del envío y recepción de mensajes. En el método de "onMessage" contiene toda la lógica de procesamiento de los mensajes recibidos. Este método es ejecutado cada vez que se recibe algún mensaje. Por otro lado, el método de "sendReplyMessage" contiene la lógica de envío de respuestas. Este método se encarga tanto de respuestas individuales como de mensajes generales o "broadcasts".

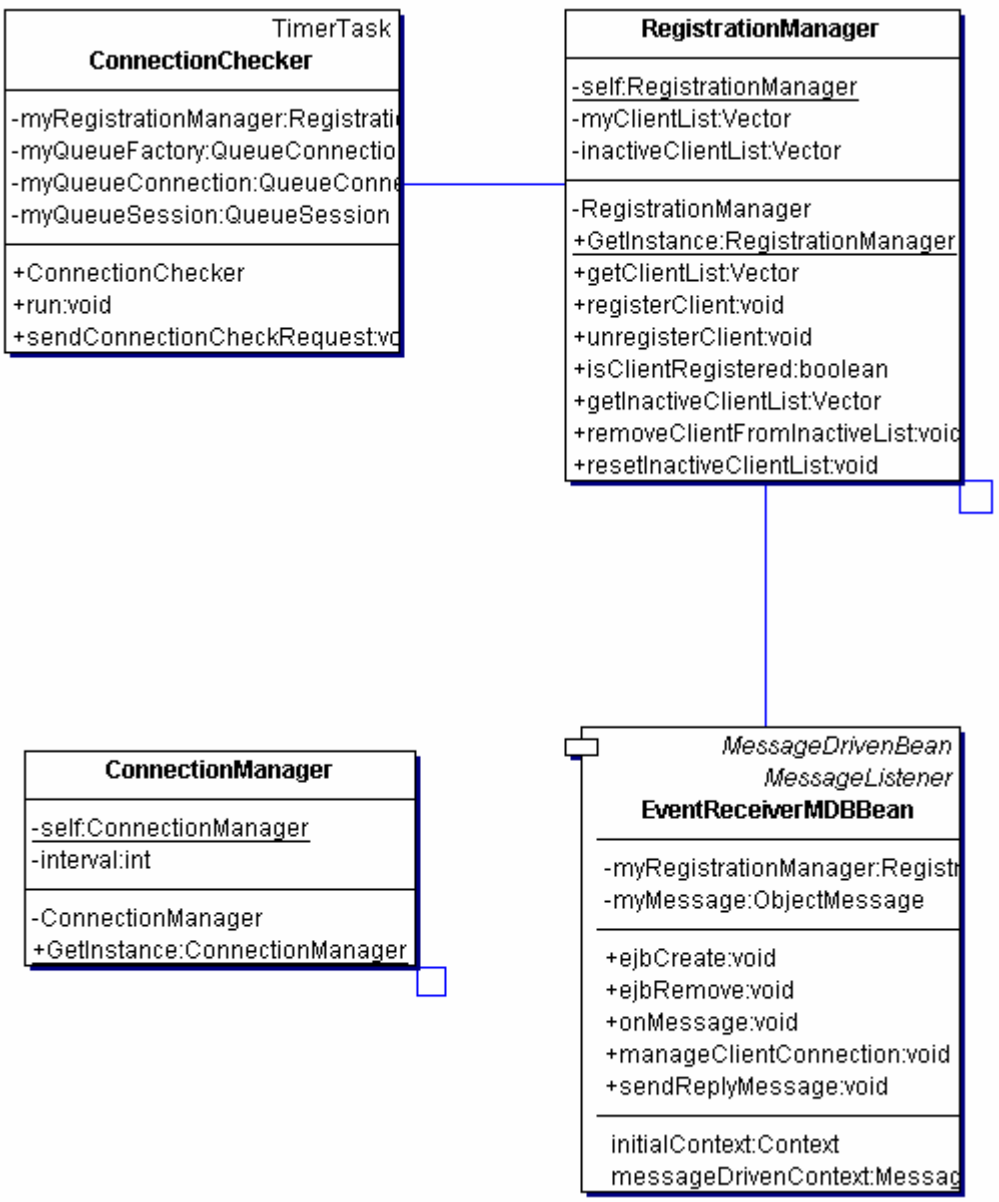

Diagrama 41 - Diagrama de clases del paquete de manejador de eventos

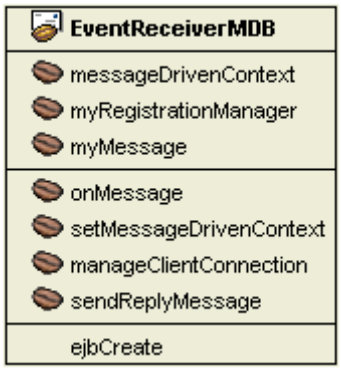

Diagrama 42 - Capa de mensajes: fachada de mensajes

Concurrencia

El único posible problema de concurrencia en el subsistema de comunicación es el que puede surgir al verificar los clientes que están conectados. El problema consiste en el caso de que un cliente este siendo agregado a la lista de clientes registrados al mismo tiempo que se dispara una verificación de clientes conectados, es decir, se da el acceso concurrente por parte del "EventReceiverMDB" como del "ConnectionChecker" de los listados del "RegistrationManager". Para evitar que existan problemas, todos los métodos de acceso del "RegistrationManager" se definieron como sincronizados.

Interfaces

El subsistema de comunicaciones es la interfaz entre el servidor y los clientes, es decir todo el subsistema es la interfaz. Siendo más precisos, la interfaz entre este subsistema y los clientes son los métodos de "onMessage" y "sendReplyMessage" del "EventReceiverMDB". Por su parte, la interfaz con el servidor es una seria de llamadas de procedimiento dentro de los mismos métodos.

5.5 Utilerías en Común

En esta sección se detallan las clases y que no son propiamente de ninguno de los subsistemas y que son utilizadas por más de uno de ellos.

Objetos de Datos

Los objetos de datos (DO, por sus siglas en inglés), son clases simples que tienen una serie de atributos y exclusivamente métodos de acceso a éstos, es decir, solamente pueden guardar información sin darle procesamiento alguno. La razón de ser de estos objetos es la de que es mucho más eficiente para las comunicaciones, especialmente en redes, cuando lo que se necesita es mandar un objeto con varios datos, contrario a tener que pedir y enviar muchas veces datos simples.

Existen dos tipos de objetos de datos y ambos son usados en este sistema:

- DOs de datos de dominio: Sus atributos son una copia fiel de los de algún objeto de negocio
- DOs a la medida: Sus atributos son compuestos de varios objetos de negocio y responden a necesidades muy específicas

Todos los DOs usados en este sistema implementan una interfaz trivial llamada "DataObject" y que fue desarrollada para permitir que cualquier comando pueda contener a cualquier DO que la implemente.

Debido a que en este sistema casi todos los objetos de datos son creados y poblados en el servidor, se creo en éste una clase que sigue el patrón de fábrica y que genera los DOs.

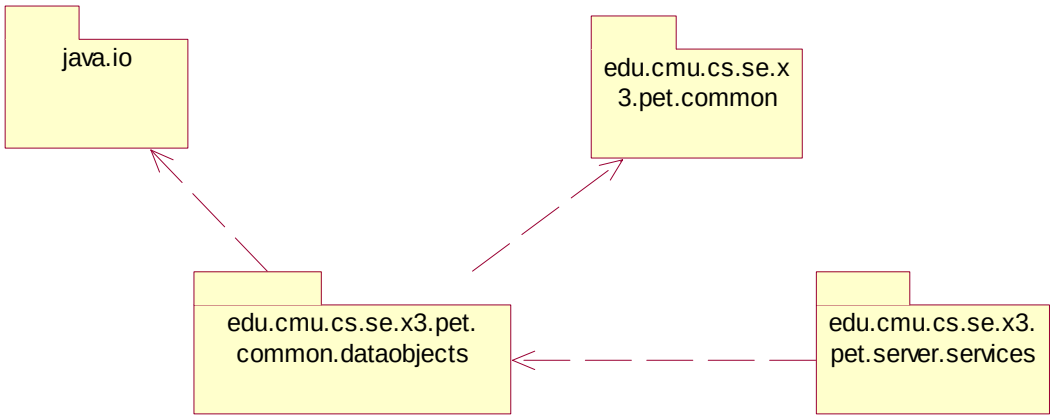

Diagrama 43 - Diagrama de paquetes de los objetos de datos

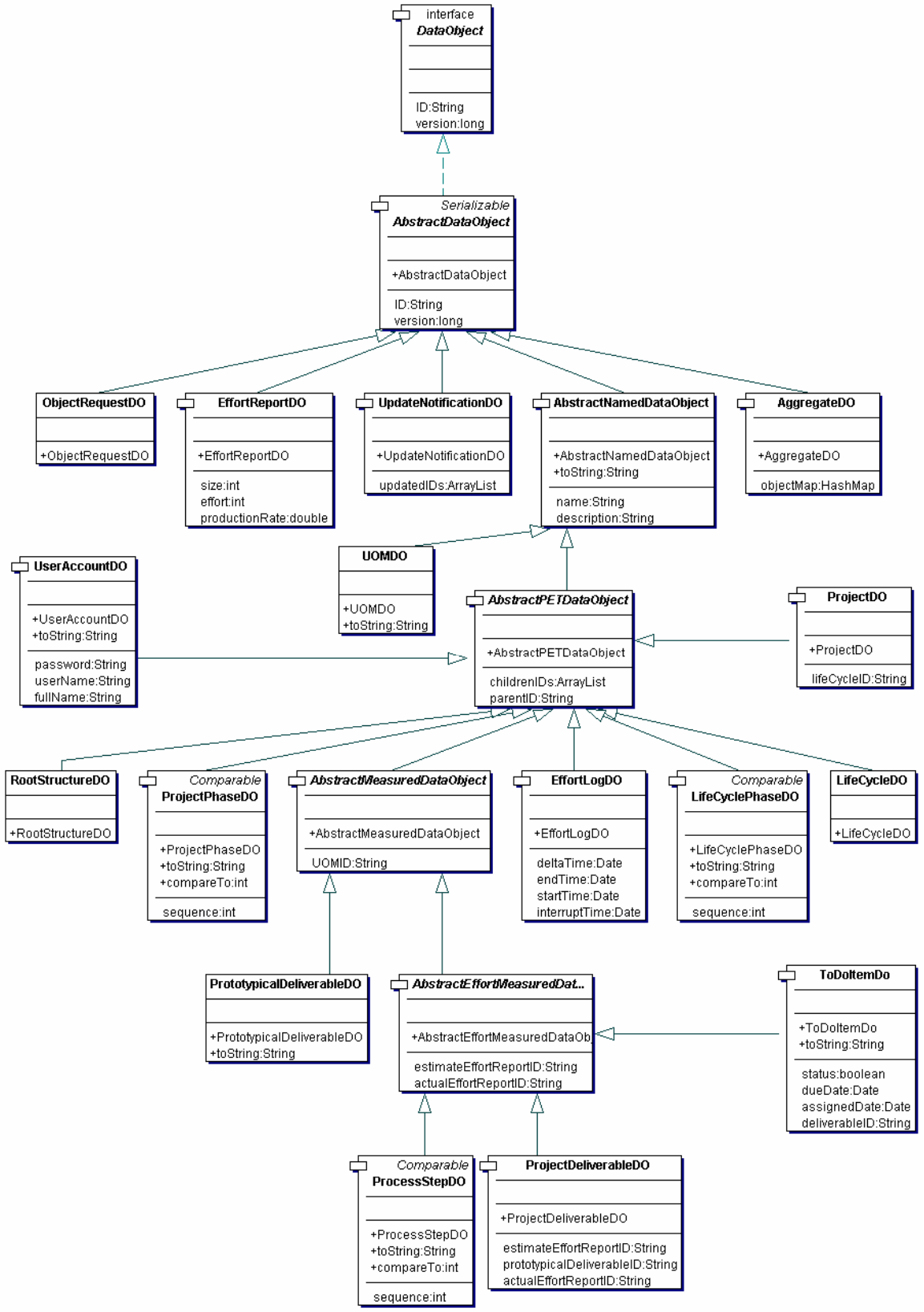

Diagrama 44 - Diagrama de clases de los objetos de datos

Excepciones

Como en cualquier sistema, se pueden generar excepciones y muchas de éstas son ya sea predecidas o esperadas. Estas excepciones puedes ser fácilmente pasadas en el servidor de una capa a otra, ya que se utilizan llamadas de procedimiento regulares, y propagadas hasta el manejador de eventos. Más allá de este último, éstas no pueden ser propagadas por lo que son transformadas en mensajes de falla como ha sido detallado en las secciones correspondientes.

Las excepciones consisten de clases muy sencillas que normalmente sólo contienen un mensaje identificador del problema y que extienden a la clase "Exception" de Java. Esta última clase contiene toda la lógica y mensajes detallados, por lo que el mensaje añadido solo sirve para se un poco más especifico en excepciones esperadas.

En este caso sólo se usaron tres excepciones: "NoChildrenException", "IncorrectVersionException" y "ConstantNotFoundException". La primera es usada para el caso de cuando se piden los hijos de un objeto de negocio que aún no tiene ninguno, éste es el caso por ejemplo de un ciclo de vida que no tiene ninguna fase.

La segunda excepción, la de "IncorrectVersionException" es utilizada cuando algún cliente pretende hacer una actualización de datos pero éstos ya no son válidos debido a que algún otro cliente ya los modificó.

La tercera excepción es la de "ConstantNotFoundException", ésta es usada cuando se le pide al método buscador de constantes de la clase dedicada a esto que le entregue alguna que no exista.

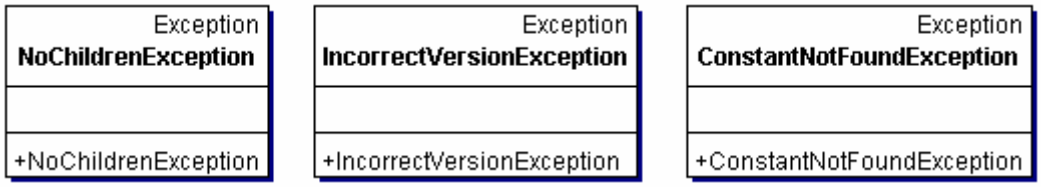

Diagrama 45 - Diagrama de clases de las excepciones

Constantes

Para este sistema se definieron una serie de constantes que van a ser compartidas por todos los subsistemas y que fueron englobadas en una sola clase. Para simplificar algunos procesos, se incluyó en esta clase un método que compara la clase de cualquier objeto y regresa la constante utilizada para reverenciarlo si es que esta existe.

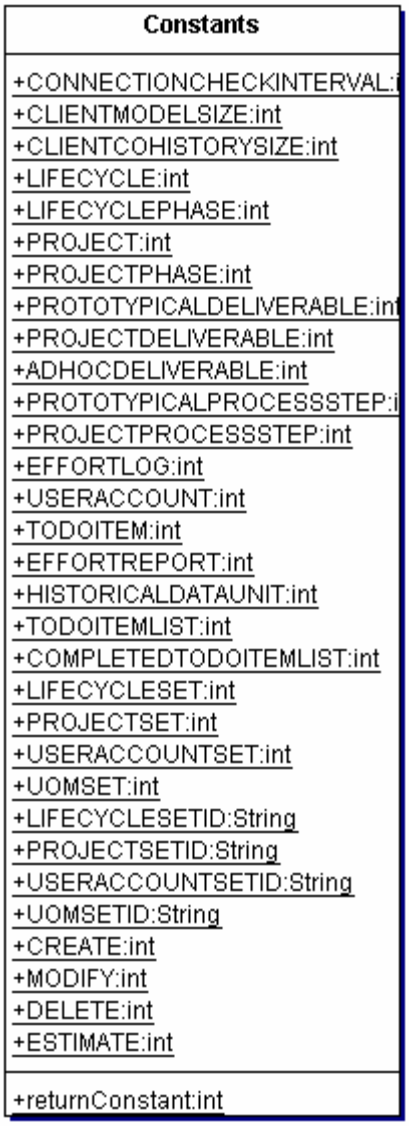

Diagrama 46 - Diagrama de clases de las constantes

Identificador Único Universal

A lo largo de todo el sistema existen muchas clases donde se requiere generar un identificador, y por muchas razones es muy conveniente utilizar un identificador universalmente único. Estos identificadores son utilizados tanto en los "Entity Beans" del servidor como en los mensajes generados desde los clientes. Los objetos identificados por estas claves, muchas veces son almacenados indistintamente de su clase original, por lo que un identificador que no se repita sin importar de que clase sea el objeto o en que computadora haya sido generado es necesario.

Existen dos formas de generar estos identificadores únicos, generarlos centralmente, lo que consumiría muchos recursos a la vez que generaría mucho tráfico innecesario o generarlos independientemente desde múltiples lugares según es descrito en el artículo al

respecto de UUIDs[8]. Se decidió implementar dicho solución usando una clase simple de Java de tipo "Singleton" para evitar la extraña pero posible ocurrencia de que dos hilos generen dos instancias que a su vez generen el mismo identificador.

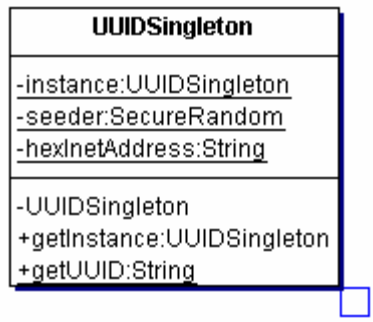

Diagrama 47 - Diagrama de clases del "Singleton" generador de UUIDs

[8] "UUIDs and GUID," Leach, Paul; Salz, Rich; Network Working Group: Internet Draft

Conclusiones

La administración de cualquier proyecto de software, contempla el control de cuatro variables fundamentales: costo, tiempo, calidad y extensión.

Estas cuatro variables, reflejan el tipo de control que se puede ejercer sobre un proyecto desde el punto de vista de los administradores, clientes o quien sea que este facultado para la administración de éste. Las acciones de control que se puede ejercer son del tipo de fijar límites, recortarlos, extenderlos o dejar libre dicha variable.

Estas cuatro variables se pueden describir de la siguiente forma:

Costo: Recursos materiales y humanos disponibles para el desarrollo. Esta variable tiene una repercusión directa en la cantidad de desarrolladores, equipos, herramientas de software… con los que ha de contar el proyecto.

Tiempo: Fecha límite de entrega.

Calidad: La calidad en proyectos de software, se refiere normalmente a dos principales conceptos, la cantidad de errores que el código contiene y a la cobertura de los requerimientos no funcionales, es decir, el "cómo" debe de funcionar el sistema en cuanto a desempeño, facilidad de uso y de modificación, escalabilidad…

Extensión: En un proyecto de software, la extensión se refiere directamente a que proporción de los requerimientos funcionales se completaron, es decir a los requisitos del "que" debe de hacer el sistema, tales como los problemas que debe de solucionar, hasta donde se ha de llegar en estas soluciones…

Como Kent Beck refiere en su libro "Extreme Programming Explained", la parte administrativa en un proyecto de software no debe de ejercer control sobre más de tres de estas variables. Si por el contrario, se le pidiera a un equipo de desarrolladores que lleven a cabo un proyecto en un tiempo reducido con un presupuesto limitado que solucione un conjunto amplio de requerimientos, el resultado muy probablemente sería de mala calidad.

Dado que este proyecto fue desarrollado con un objetivo académico, estas variables tuvieron ciertas características particulares. En específico, el costo no se refirió al monetario o directo como lo sería en un proyecto comercial, sino al indirecto, es decir, la cantidad de desarrolladores y equipos de cómputo disponibles, acceso a asesores externos, herramientas de software… Adicionalmente, el tiempo aún cuando tuvo cierta flexibilidad, siempre estuvo circunscrito al calendario académico. Esto permitió que durante el proceso de desarrollo se pudieran modificar los parámetros correspondientes a la calidad y a la extensión.

Tomando todo lo anteriormente expuesto, es justo aclarar que este proyecto tuvo un inicio muy ambicioso en lo correspondiente a requerimientos funcionales y no funcionales. El problema consistió en que dadas las limitaciones de tiempo y costo, el

resultado final contiene un subconjunto reducido de dichos requerimientos. Dado que este problema fue previsto desde el inicio, se escogió un ciclo de vida incremental para el desarrollo del proyecto. La estructura de fases del ciclo de vida, forzó la definición de prioridades en los requerimientos.

La división de los requerimientos por fases, permitió que en la primera se desarrollara el conjunto mínimo de éstos que permiten un funcionamiento coherente del sistema. Con todo esto, se logró que dicho subconjunto de los requerimientos iniciales y que finalmente fue implementado, tuviera cuando menos la funcionalidad mínima para considerarse completo.

Por último, los proyectos de software son por naturaleza difíciles de planear, y lo que es peor, normalmente están llenos de imprevistos. Los proyectos suelen terminar en un éxito o un fracaso, dependiendo solamente de la capacidad y experiencia de los desarrolladores. Es en proyectos como éstos, donde los procesos de administración y planeación de proyectos deben de tener un mayor desarrollo, ya que esto permitiría que los resultados sean más predecibles. El presente cumplió exitosamente con la doble función de servir como ejercicio del uso de los procesos definidos por el "Software Engineering Institute" para este propósito y de desarrollar un primer prototipo de herramienta para facilitar estos mismos procesos. Finalmente, las técnicas y procesos de optimización y estandarización de la Ingeniería de Software como disciplina, deberían de formar parte integral de los planes de estudio de todas las disciplinas relacionadas con el desarrollo de software a nivel profesional.

Glosario de Términos

Término	Descripción
API	Interfaz de Programación de Aplicaciones ("Application Program Interface"). Conjunto de funciones, rutinas o clases que sirven como herramientas para el desarrollo de programas.
AWT	Caja de Herramientas Abstractas para Ventanas ("Abstract Window Toolkit"). Es la API (ver entrada en este glosario) estándar para generar interfaces de usuario en Java. Forma parte de las JFC (ver entrada en este glosario).
CMM	Modelo de Madures de Capacidades ("Capability Maturity Model"). Este modelo desarrollado por el SEI (ver entrada en este glosario) permite evaluar el proceso de desarrollo de software de las organizaciones en una escala del 1 al 5.
CMP	Persistencia Manejada por el Contenedor ("Container Managed Persistence"). Se refiere a un tipo específico de EJB (ver entrada en este glosario) que utiliza al contenedor para almacenar de forma no volátil su estado. El contenedor requiere de tener configurado uno o más medios de almacenamiento permanente de información, éstos varían desde archivos de texto hasta bases de datos relacionales.
COTS	Componentes Disponibles Comercialmente Listos para Usarse ("Commercial of the Shelf"). Componentes desarrollados comercialmente que pueden ser usados para propósitos específicos en contraste con componentes desarrollados para una aplicación específica.
DDD	Documento Detallado de Diseño ("Detailed Design Document"). La IEEE (ver entrada en este glosario) describe este documento como una traducción de los requisitos en una descripción de la estructura de software, los componentes, interfaces y datos necesarios para la fase de implementación.
DO	Objeto de Dato ("Data Object"). Cuando es necesario enviar muchos elementos de información entre un componente de software y otro para una sola operación, normalmente no es conveniente enviarlo en muchas partes. Para solucionar este problema se usa el patrón de diseño llamado DO o DTO ("Data Transfer Object") que consiste en usar un objeto con todos los atributos necesarios para la operación y métodos de acceso a éstos.
EJB	Grano de Café de Empresarial ("Enterprise Java Bean"). Objetos de servidor definidos dentro de J2EE (ver entradas respectivas) que son auto contenidos y entre otras cosas implementan control de transacciones, persistencia, control de accesos concurrentes, servicios de directorio,

seguridad,... Existen EJBs de Entidad, que representan objetos de negocio, de Sesión con y sin estado, que implementan servicios y de Mensaje, éstos últimos responden a eventos de JMS (ver entrada en este glosario).

FIFO Primeras Entradas, Primeras Salidas ("First In, First Out"). Modelo de colas que en el contexto de las estructuras de datos en las ciencias de la computación, se refiere a las listas en las que las entradas y salidas de la información se dan de esta manera.

IDE Ambiente de Desarrollo Integrado ("Integrated Development Environment"). Se conoce por este acrónimo a las aplicaciones de desarrollo de software que ayudan al programador a generar el código de manera más eficiente. Normalmente incluyen un editor de texto, un interprete y/o compilador y algunas veces un "debugger", es decir una aplicación para la búsqueda y resolución de errores en la programación.

IEEE Instituto de Ingenieros Eléctricos y Electrónicos ("Institute of Electrical and Electrónics Engineers"). Organización internacional sin fines de lucro que tiene como objetivo fundamental el desarrollo de las tecnologías relacionadas con el uso de la electricidad. Una de las funciones más importantes del Instituto es la generación de estándares.

ISO Organización Internacional para la Estandarización (el nombre está derivado del griego *isos* - igual). Organización compuesta por los representantes de organizaciones nacionales de estándares de gran cantidad de países. Esta organización es la principal autoridad en el desarrollo de estándares para uso comercial e industrial y éstos comprenden una gran diversidad de temas, desde los tecnológicos hasta la construcción.

J2EE Java 2 Edición Empresarial ("Java 2 Enterprise Edition"). Plataforma para el desarrollo y ejecución de aplicaciones distribuidas y de múltiples capas basadas mayoritariamente en componentes modulares ejecutados en un servidor de aplicaciones.

J2SE Java 2 Edición Estándar ("Java 2 Standard Edition". Colección de APIs (ver entrada en este glosario) que definen la plataforma de programación del lenguaje Java.

JDBC Conectividad de Bases de Datos de Java ("Java Data Base Connectivity"). API (ver entrada en este glosario) que definen como una aplicación puede tener acceso a los servicios de un sistema manejador de bases de datos.

JFC Clases de Fundamentales de Java ("Java Foundation Classes"). Marco de desarrollo de aplicaciones gráficas portátiles (ejecutables en diferentes plataformas) de Java.

JFCSwing Conocido también como Swing. Juego de herramientas de desarrollo de aplicaciones gráficas del lenguaje Java. Utiliza la API de Java llamada AWT (ver entrada en este glosario) pero añade la definición de un modelo de desarrollo de tipo MVC (ver entrada en este glosario).

JMS Servicio de Mensajes de Java ("Java Message Service"). API de Java que

define la forma de enviar mensajes entre dos o más aplicaciones utilizando la conectividad proporcionada por las redes TCP/IP (ver entrada en este glosario).

Jtree Componente gráfico que forma parte de Swing. Este componente despliega en la pantalla una lista jerarquizada de información expandible y contraible.

JVM Máquina Virtual de Java ("Java Virtual Machine"). Aplicación que sirve como capa de abstracción entre la plataforma real de un sistema de cómputo y el entorno de programación y ejecución del lenguaje Java. El proceso de desarrollo de un programa en Java consiste en la escritura del código fuente por parte del programador, una compilación intermedia a lo que es conocido como "BiteCode" (versión del lenguaje de máquina de la JVM) y su ejecución en la JVM.

MDB Tipo de EJB (ver entrada en este glosario) que recibe y responde a mensajes generados usando JMS (ver entrada en este glosario).

MVC Modelo-Vista-Controlador ("Model-View-Controller"). Modelo arquitectónico de software que separa el modelo de datos, la interfaz de usuario y la lógica de control en tres diferentes componentes de manera que los cambios en el diseño de la interfaz de usuario tengan un impacto mínimo en el modelo de datos. En el lenguaje de Sun Microsystems, esta arquitectura es conocida como "modelo 2"

PEPS ver FIFO

PET Herramienta de Implementación de Proceso (“Process Enactment Tool”). Herramienta desarrollada como objetivo principal de este proyecto

PPP Desempeño Profesional Predecible (“Predictable Professional Performance”). Conjunto de técnicas y procesos desarrollados por el Dr. Lynn Carter en la Universidad Carnegie Mellon que expande las técnicas del PSP para todas las actividades relativas al software de desarrollo individual. Añade también notaciones resumidas para los documentos de diseño.

PSP Proceso Personal de Software (“Personal Software Process”). Método desarrollado por Watts S. Humprey del SEI

RDBMS Sistema Manejador de Base de Datos Relacional (“Relational Data Base Management System”)

SAD Documento de Arquitectura de Software ("Software Architecture Document"). Documento que describe el diseño e implementación de la estructura de alto nivel de un proyecto de software.

SEI Instituto de Ingeniería de Software (“Software Engineering Institute”). Centro de investigación y desarrollo de la Universidad Carnegie Mellon cuyo objetivo es el de desarrollar la Ingeniería de Software como ciencia

SRS Documento de Requisitos de Sistema ("Software Requirements Specification"). Este documento forma parte del URD (ver entrada en este

glosario) para analizar los casos de uso desde diferentes perspectivas para así eliminar inconsistencias, ambigüedades y omisiones. Este documento es utilizado para definir los requisitos del sistema desde un punto de vista del desarrollo de software, es decir como conceptos, interfaces, módulos...

TCP/IP Conjunto de protocolos de comunicación que definen como se comunican los diferentes equipos conectados en red. Estos protocolos forman una pila que va desde los que definen como se desarrollan las comunicaciones físicamente, es decir a nivel de bits hasta niveles más abstractos que reflejan como se entienden las comunicaciones por parte de las aplicaciones que utilizan la red. Todos los niveles superiores dependen de los inferiores para desarrollar su función.

TSP Proceso de Equipo de Software ("Team Software Process"). Método que utiliza los principios del PSP adaptados al funcionamiento de un equipo de desarrollo de software y que redistribuye las responsabilidades de administración del proyecto entre los desarrolladores

UI Interfaz de Usuario ("User Interface"). Conjunto de medios por medio de los cuales los usuarios de un sistema interactúan con éste. Éstos permiten la comunicación del usuario al sistema para controlarlo, y en la dirección contraria para desplegar información.

URD Documento de Requisitos de Usuario ("User Requirements Document"). Este documento es el principal medio donde se definen los objetivos de un desarrollo de software en términos de uso de los interesados en el desarrollo. En este documento, se definen las funcionalidades esperadas del sistema, así como el ambiente de desarrollo y operativo en que el sistema va a trabajar.

UUID Identificador Único Universal ("Universally Unique Identifier"). Concepto definido por Paul Leach y Rich Salz como parte de su trabajo en el Network Working Group que definió mucho de la ARPANET ahora Internet. El UUID consiste en un identificador único en el tiempo y el especio dentro de un dominio.

XP Programación extrema ("Extreme Programming"). Conjunto de principios, valores y prácticas que permiten un desarrollo de software muy ágil. XP fue definido por Kent Beck en su libro "eXtreme Programing eXplained".

Apéndices

Apéndice A. Análisis de Rendimiento

Debido a que no se han hecho pruebas de tiempos de respuesta del sistema ante diferentes escenarios de carga, no se tienen datos para definir el rendimiento. Lo que se puede hacer a estas alturas es dejar estos valores como variables para que se puedan definir las metas de rendimiento.

Existen dos elementos que son cruciales para el rendimiento de este sistema, el servidor y el manejador de eventos. Ambos subsistema implementan colas tipo primeras entradas-primeras salidas (FIFO, por sus siglas en inglés) para dar servicio a las peticiones.

De acuerdo a los requisitos no-funcionales del sistema, tal como fueron especificados en el SRS, éste debe de ser capaz de darle servicio a un máximo de 50 usuarios con una frecuencia de respuesta media de una cada 10 segundos para cada uno. Esto último se traduce en 6 peticiones de servicio cada minuto. Si consideramos 50 usuarios, tenemos 300 peticiones por minuto ó 5 por segundo. El manejador de eventos y el servidor tienen que por lo menos dar servicio a esta razón para mantenerse delante de la carga máxima esperada. Recordemos también que el tiempo de respuesta debe de ser menor a 2 segundos.

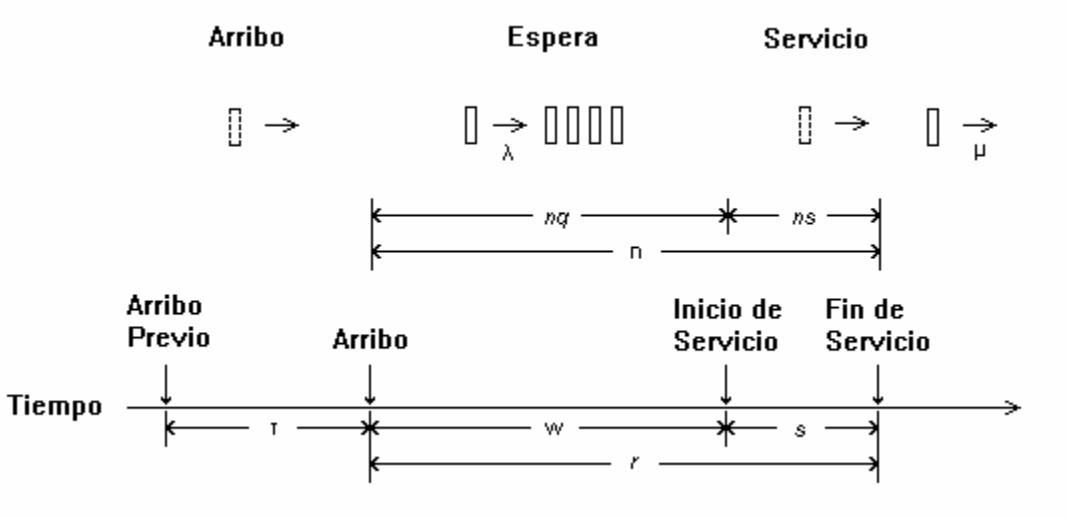

Diagrama 48 - Variables aleatorias usadas en el análisis de la cola

Para ambos subsistemas, tenemos que $\lambda = 0.2s$. Como ambos tienen solo un procesador dando respuesta a las peticiones, tenemos que $\lambda = \mu$. El resto de las variables no las podemos especificar a estas alturas.

El subsistema del servidor se comporta como una cola que tiene un intervalo de arribo exponencial con un tiempo de servicio exponencial con un solo procesador, o lo que en la notación de Kendall[9] es una cola tipo M/M/1.

Por otra parte, el manejador de eventos tiene un intervalo de arribo exponencial con un tiempo de servicio determinístico o constante con un solo procesador, en la misma notación esto es una cola tipo M/D/1.

En este documento se transcribe el análisis de una cola tipo M/D/1 según se describe en la referencia[10] para poder ser usado cuando se cuente con los datos de respuesta.

1. Parámetros:
$\lambda = 0.2$
$E[s]$ es constante

2. Intensidad de Tráfico:
$\rho = \lambda E[s]$

3. Condición de Estabilidad:
El sistema es estable si la intensidad de tráfico de menor que 1.

4. Probabilidad de N trabajos en el sistema:

$$p_n = \begin{cases} 1-\rho, n = 0 \\ (1-\rho)(e^{\rho}-1), n = 1 \\ (1-\rho)\sum_{j=0}^{n} \dfrac{(-1)^{n-j}(j\rho)^{n-j-1}(j\rho+n-j)e^{jp}}{(n-j)!}, n \geq 2 \end{cases}$$

5. Media de trabajos en el sistema:
$E[n] = \rho + \rho^2/[2(1-\rho)]$

6. Varianza de trabajos en el sistema:
$Var[n] = E[n] + \rho^3/[3(1-\rho)] + \rho^4/[4(1-\rho)^2]$

7. Función de distribución acumulada para el tiempo de respuesta:

$$F(r) = p_n \frac{(r - nE[s])}{E[s]} + \sum_{j=0}^{n-1} p_j \qquad r \geq E[s] \qquad n = \left\lfloor \frac{r}{E[s]} \right\rfloor$$

8. Tiempo de respuesta media:

---

[9] "The Art of Computer Systems Performance Analysis"; Jain, Raj; Wiley; página 509
[10] "The Art of Computer Systems Performance Analysis"; Jain, Raj; Wiley; página 542. El M/M/1 puede ser encontrado en la página 542.

$$E[r] = E[s] + \rho E[s]/[2(1-\rho)]$$

9. Varianza del tiempo de respuesta:

$$Var[r] = \rho(E[s])^2/[3(1-\rho)] + \rho^2(E[s])^2/[4(1-\rho)^2]$$

10. Media del número de trabajos en la cola:

$$E[n_q] = \rho^2/[2(1-\rho)]$$

11. Varianza del número de trabajos en la cola:

$$Var[n_q] = \rho^2 + \frac{\rho^2}{2(1-\rho)} + \frac{\rho^3}{3(1-\rho)} + \frac{\rho^4}{4(1-\rho)^2}$$

12. Tiempo medio de espera:

$$E[w] = \rho E[s]/[2(1-\rho)]$$

13. Varianza del tiempo de espera:

$$Var[w] = Var[r]$$

14. Probabilidad de darle servicio a n trabajos en un periodo ocupado:

$$P(n) = \frac{(np)^{n-1}}{n!}e^{-np}$$

15. Función de distribución acumulada para un periodo ocupado:

$$F(b) = \sum_{j=1}^{n} \frac{(j\rho)^{j-1}}{j!}e^{-j\rho}, \qquad n = \left\lfloor \frac{b}{E[s]} \right\rfloor$$

Apéndice B. Código Fuente: Ejemplos

B1 Subsistema de Servidor: Ciclos de Vida

Este ejemplo de código comprende un EJB en su especificación 2.0 de tipo de entidad con persistencia manejada por controlador (CMP). Éstos componentes están compuestos por una clase que implementa el contenido del "Bean" y uno o dos juegos de interfaces: (Remota y de Casa) y/o (Local y Local de Casa). Éstos se diferencian en que el primer juego implementa llamadas de procedimiento remotas, es decir que el procedimiento llamado puede estar en una diferente computadora que el llamante, mientras que la segunda permite el uso de llamadas locales de procedimiento. Por otro lado, las interfaces llamadas "de Casa" sirven para la administración de las instancias del "Bean", mientras que las remotas contienen los métodos de acceso al contenido de éstos.

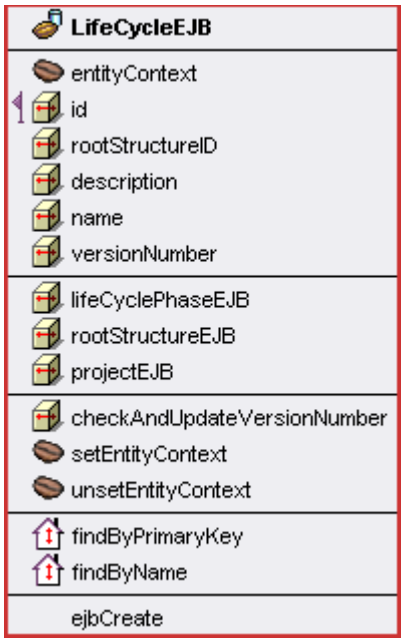

Diagrama 49 - Diagrama conceptual del EJB de LifeCycle

```
package edu.cmu.cs.se.x3.pet.server.entities;

import javax.ejb.EJBLocalHome;
import javax.ejb.CreateException;
import javax.ejb.FinderException;

public interface LifeCycleEJBHome
    extends EJBLocalHome {

  public LifeCycleEJB create() throws CreateException;

  public LifeCycleEJB findByPrimaryKey(String id) throws FinderException;

  public LifeCycleEJB findByName(String name) throws FinderException;
}
```

Interfaz Local de Casa del EJB: LifeCycleEJBHome.java

```
package edu.cmu.cs.se.x3.pet.server.entities;

import javax.ejb.EJBLocalObject;
import edu.cmu.cs.se.x3.pet.common.dataobjects.AbstractDataObject;
import edu.cmu.cs.se.x3.pet.common.exceptions.IncorrectVersionException;
import java.util.Collection;

public interface LifeCycleEJB
    extends EJBLocalObject {

  public void checkAndUpdateVersionNumber(AbstractDataObject db) throws
      IncorrectVersionException;

  public String getId();

  public void setLifeCyclePhaseEJB(Collection lifeCyclePhaseEJB);

  public Collection getLifeCyclePhaseEJB();

  public void setRootStructureEJB(RootStructureEJB rootStructureEJB);

  public RootStructureEJB getRootStructureEJB();

  public String getRootStructureID();

  public void setProjectEJB(Collection projectEJB);

  public Collection getProjectEJB();

  public void setDescription(String description);

  public String getDescription();

  public void setName(String name);

  public String getName();

  public void setVersionNumber(long versionNumber);

  public long getVersionNumber();

}
```

Interfaz Local del EJB: LifeCycleEJB.java

```
package edu.cmu.cs.se.x3.pet.server.entities;

import javax.ejb.EntityContext;
import javax.ejb.CreateException;
import javax.ejb.RemoveException;
import java.util.Collection;
import edu.cmu.cs.se.x3.pet.common.*;
import java.util.*;

public abstract class LifeCycleEJBBean
    extends AbstractBusinessObject {
  EntityContext entityContext;

  public void ejbRemove() throws RemoveException {
   }

  public abstract void setId(String id);

  public abstract void setLifeCyclePhaseEJB(Collection lifeCyclePhaseEJB);

  public abstract void setRootStructureEJB(RootStructureEJB rootStructureEJB);

  public abstract void setRootStructureID(String rootStructureID);

  public abstract void setProjectEJB(Collection projectEJB);

  public abstract String getId();

  public abstract Collection getLifeCyclePhaseEJB();

  public abstract RootStructureEJB getRootStructureEJB();

  public abstract String getRootStructureID();

  public abstract Collection getProjectEJB();

  public String ejbCreate() throws CreateException {
    setRootStructureID(Constants.LIFECYCLESETID);
    setId(UUIDSingleton.getUUID());
    setVersionNumber(0);
    return getId();
  }
  public void ejbPostCreate() throws CreateException {
  }
}
```

Cuerpo del EJB: LifeCycleEJBBean.java

B2 Subsistema de Comunicaciones: Manejador de Conexiones

El manejador de conexiones es un buen ejemplo del uso del patrón de diseño “Singleton”, es decir de una clase que solo puede tener una instancias, sin importar cuantas veces se intente instanciar. Por otro lado, es interesante notar que el manejador de conexiones implementa un “Timer”, es decir una clase que repite una acción cada cierto tiempo.

```
package edu.cmu.cs.se.x3.pet.communications.eventmanagement;

import java.util.Timer;

public class ConnectionManager {
  private static ConnectionManager self;
  private                    int                    interval                    =
edu.cmu.cs.se.x3.pet.common.Constants.CONNECTIONCHECKINTERVAL;

  private ConnectionManager() {
    Timer t1 = new Timer(true);

    //the connection check process starts after delay of interval*1000 milliseconds
    //and runs every interval*1000 milliseconds
    t1.schedule(new ConnectionChecker(),interval*1000 ,interval*1000);
    System.out.println("Connection Manager has been initialized");
    System.out.println("Connection  check  will  be  run  every  "  +  interval  +  "
seconds");
  }

  public static ConnectionManager GetInstance(){
    if(self == null){
      self = new ConnectionManager();
    }
    return self;
  }
}
```

ConnectionManager.java

B3 Subsistema de Comunicaciones: Comando de Petición de Objeto

Esta clase es interesante por que es una implementación del patrón de diseño de uso de comandos para la comunicación entre capas. En este caso se crean una serie de clases llamadas de comando que contienen toda la información necesaria para ejecutar una acción. Estos comandos al ser recibidos se puede definir de que clase son por introspección y con este mismo método extraer la información almacenada para poder ejecutar dicha acción.

```
package edu.cmu.cs.se.x3.pet.communications.command;

import edu.cmu.cs.se.x3.pet.common.dataobjects.*;

public class ObjectRequestCommand extends AbstractPETCommand{
  private int type;
  private ObjectRequestDO dataObject;

  public ObjectRequestCommand(int aType, ObjectRequestDO aDataObject) {
    type = aType;
    dataObject = aDataObject;
  }
  public int getType(){
    return type;
  }
  public void setType(int aType){
    type = aType;
  }
  public ObjectRequestDO getDataObject(){
    return dataObject;
  }
  public void setDataObject(ObjectRequestDO aDataObject){
    dataObject = aDataObject;
  }
}
```

ObjectRequestCommand.java

Apéndice C. Capturas de Pantalla: Casos de Uso Ejemplo

C1 Crear Proyecto

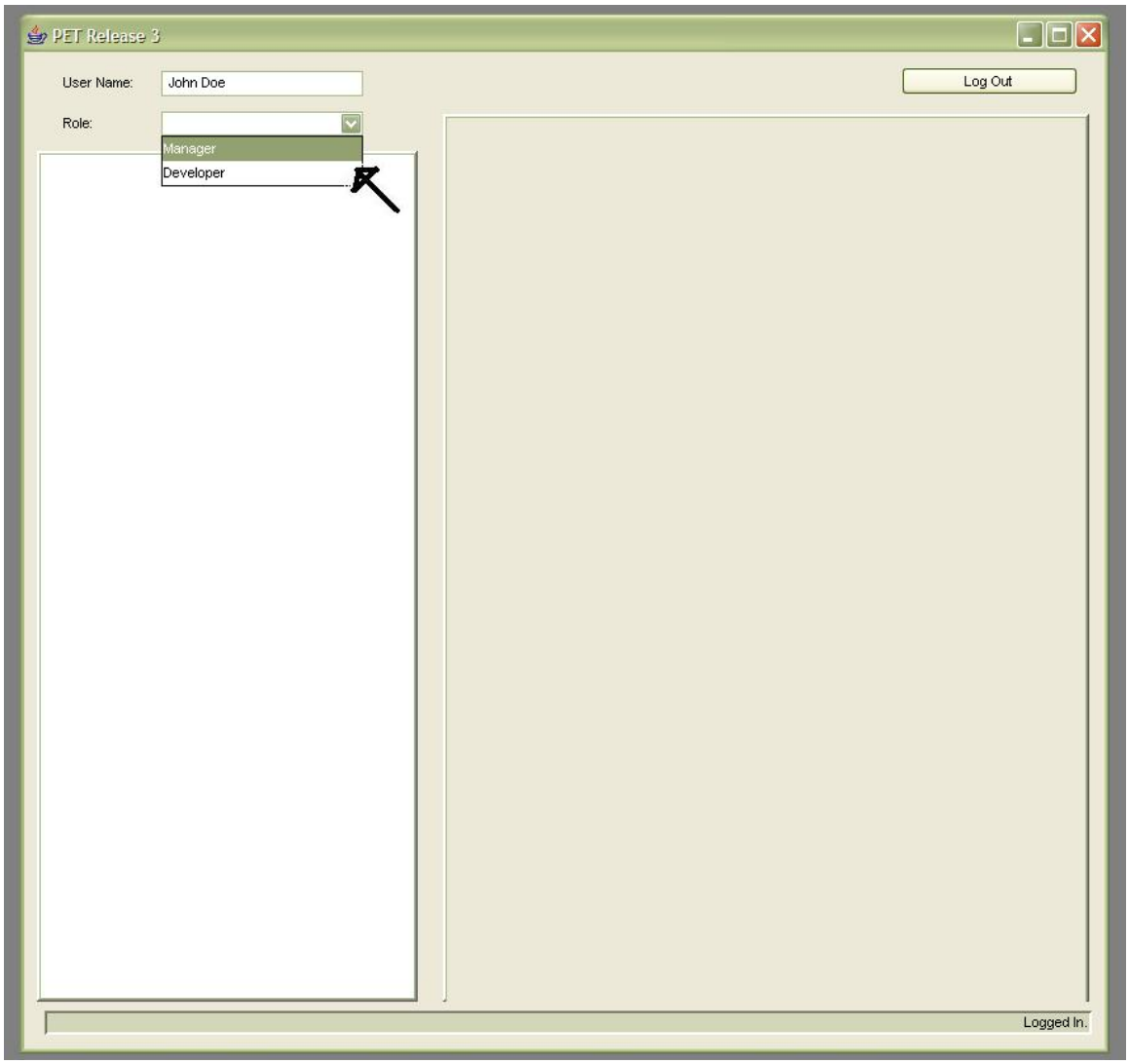

Diagrama 50 - Propuesta de UI: John Doe se registra y selecciona privilegios de administrador

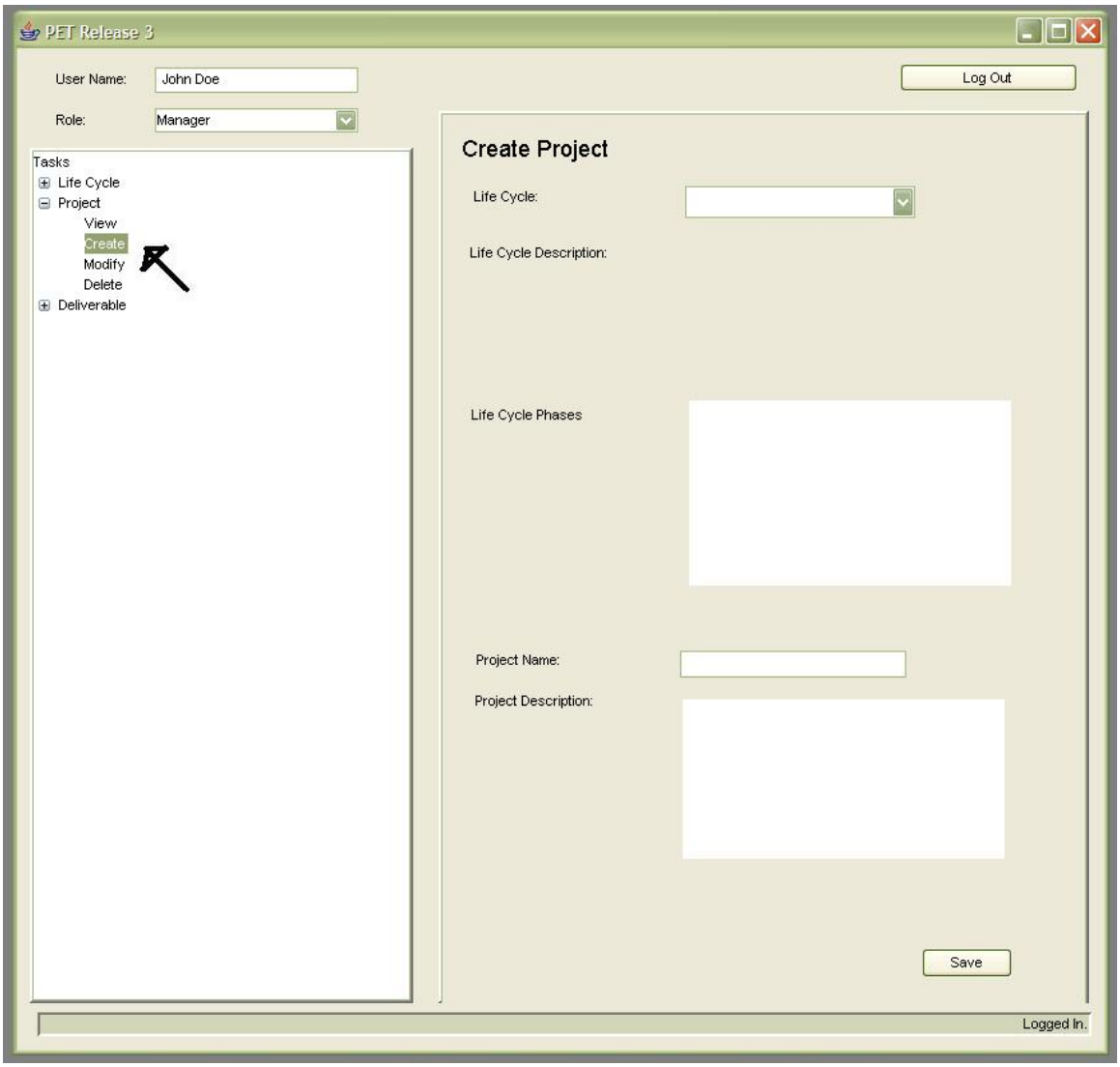

Diagrama 51 - Propuesta de UI: John Doe decide crear un proyecto

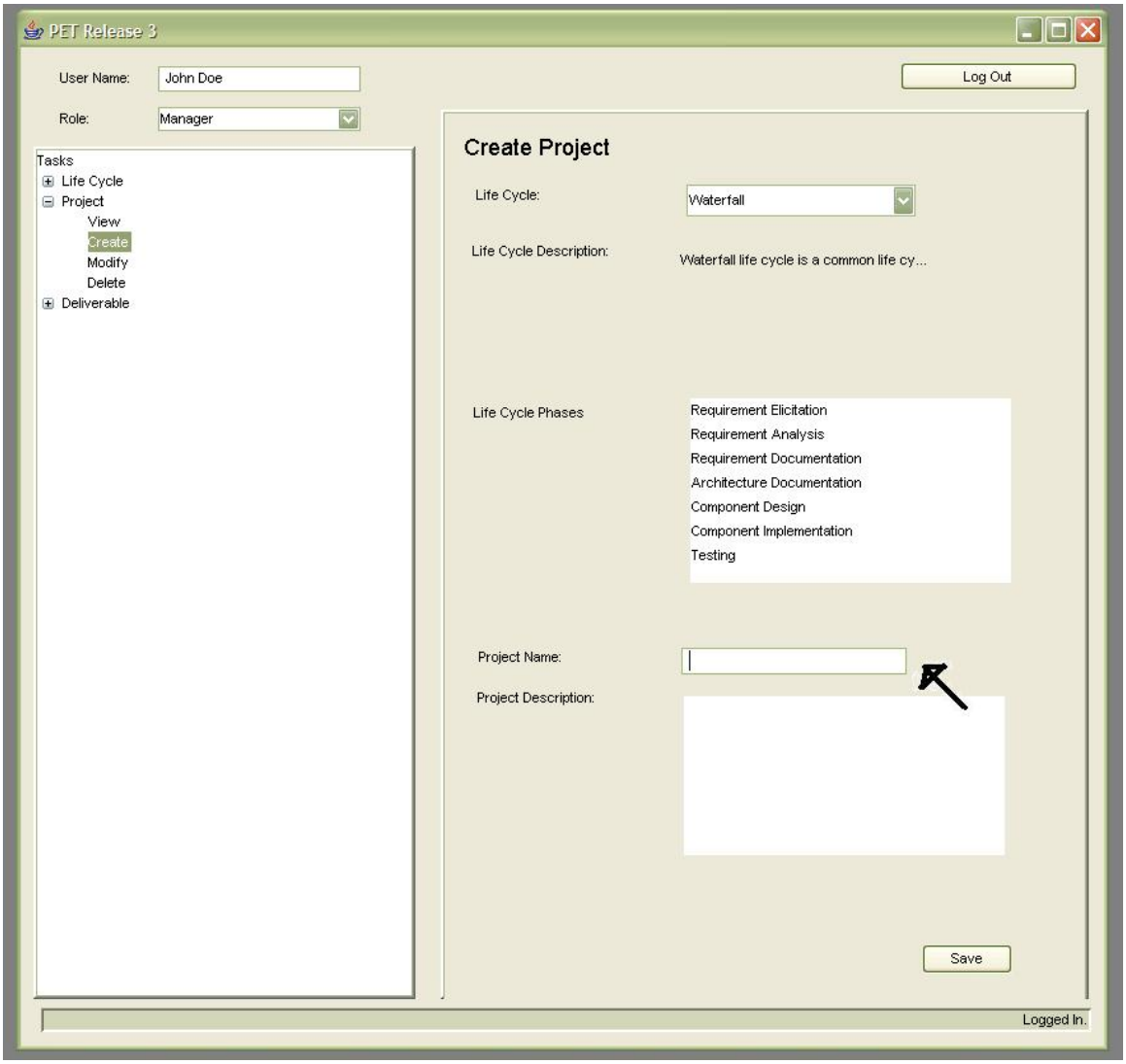

Diagrama 52 - Propuesta de UI: John Doe selecciona un ciclo de vida. Los detalles del proyecto son llenados según el ciclo de vida seleccionado

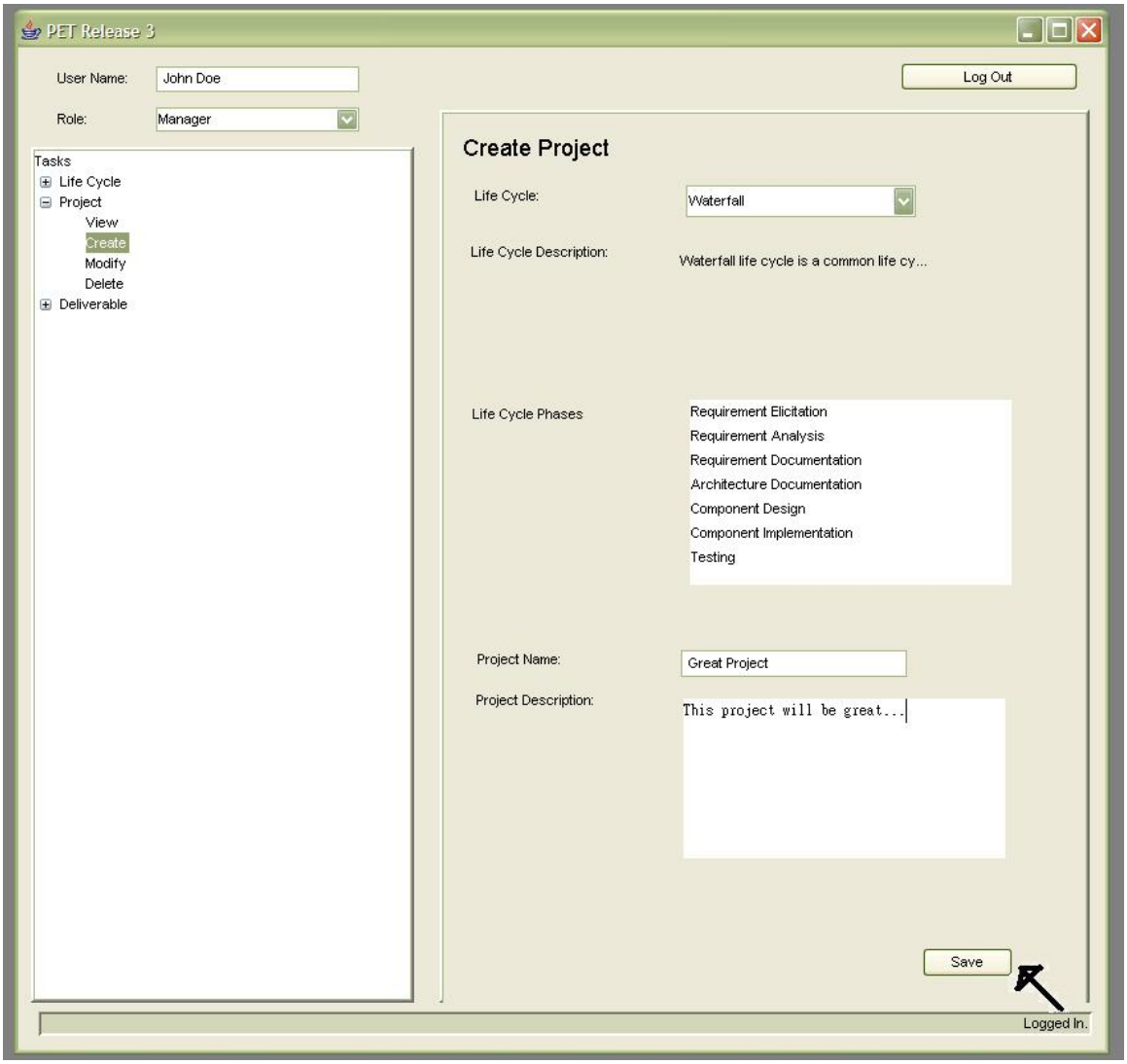

Diagrama 53 - Propuesta de UI: John Doe llena los detalles del proyecto y los envía

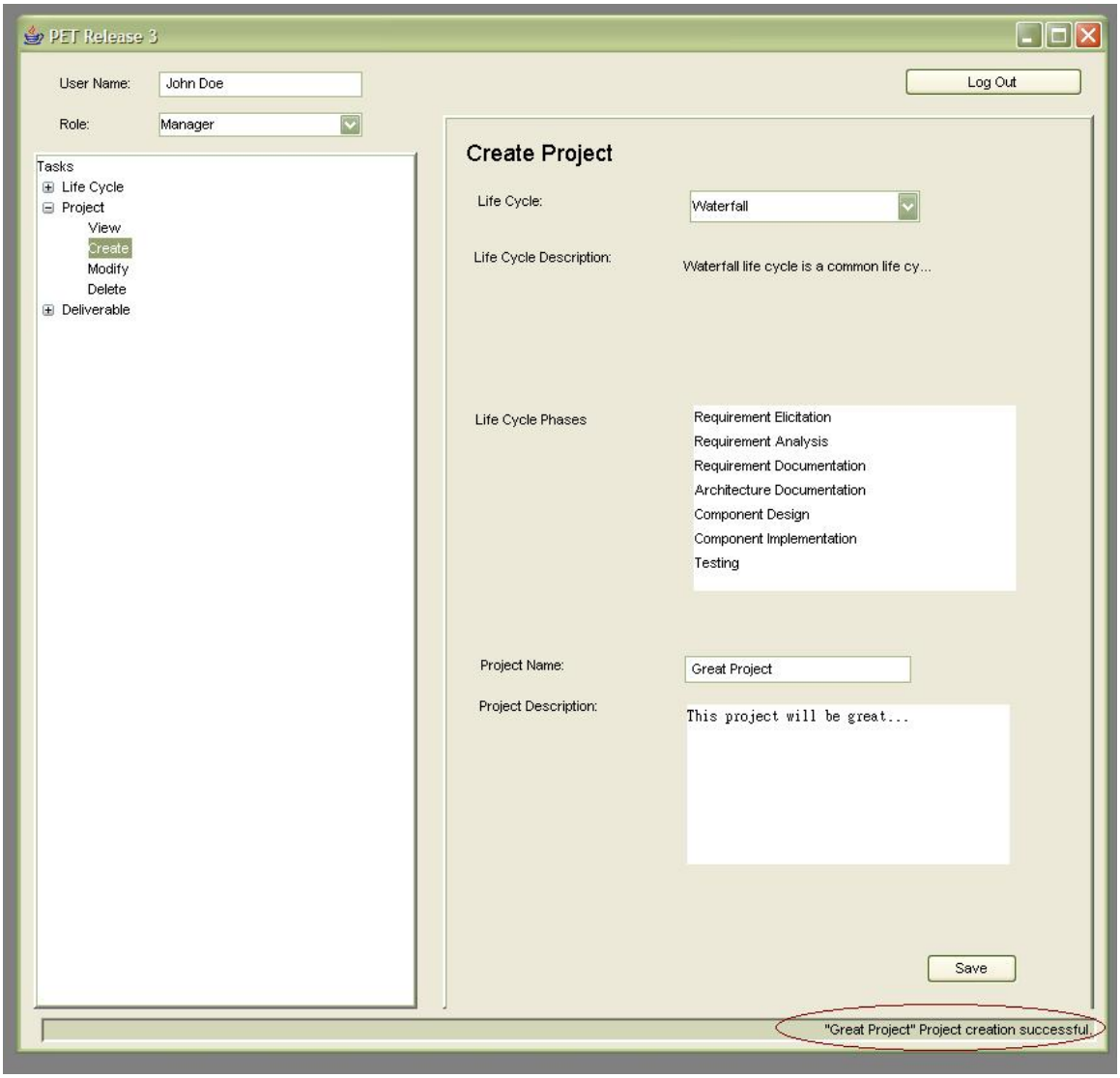

Diagrama 54 - Propuesta de UI: El cliente despliega el mensaje de éxito en la operación al recibir dicha respuesta del servidor

C2 Registro de Esfuerzo

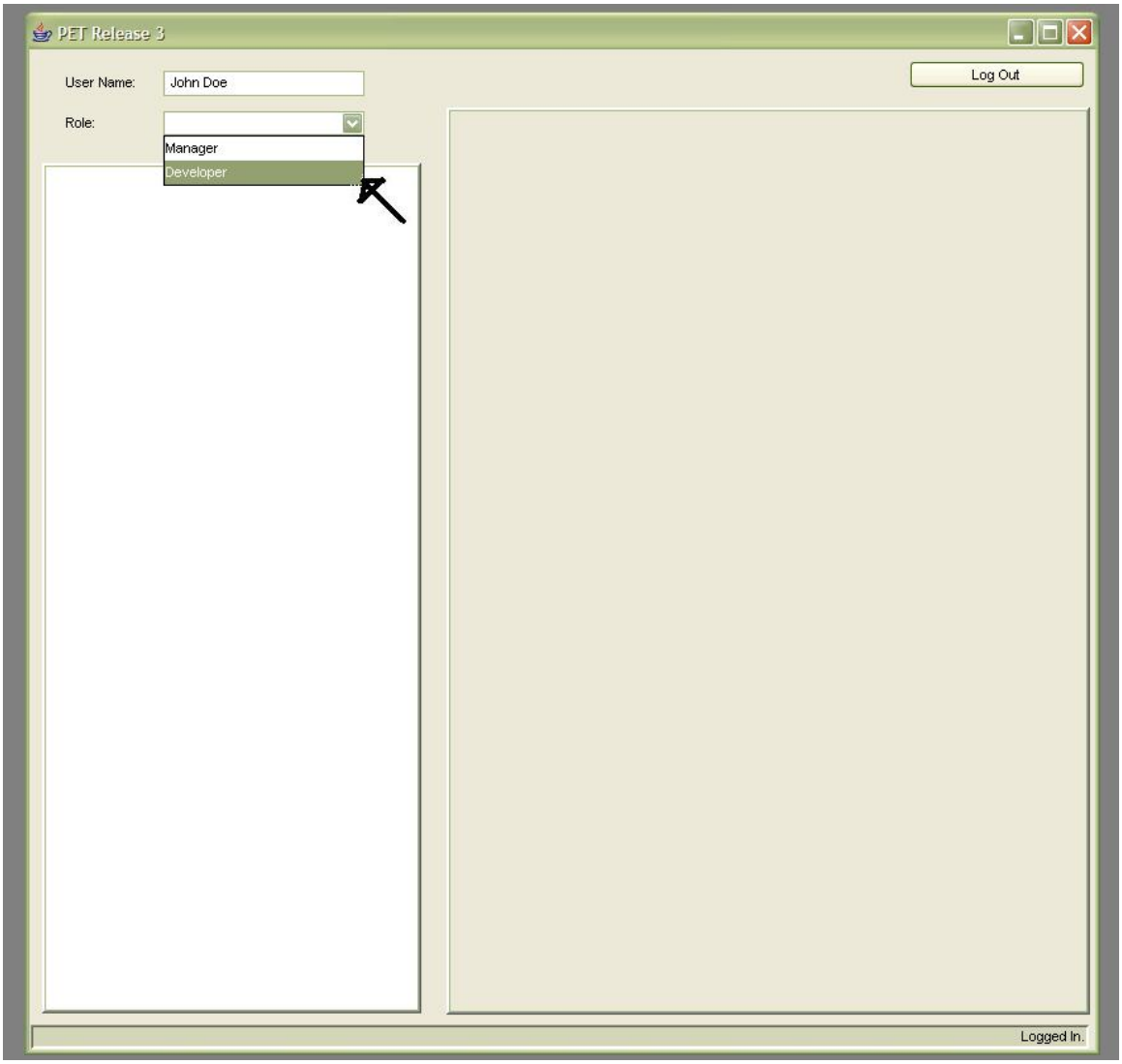

Diagrama 55 - Propuesta de UI: John Doe se registra y selecciona privilegios de desarrollador

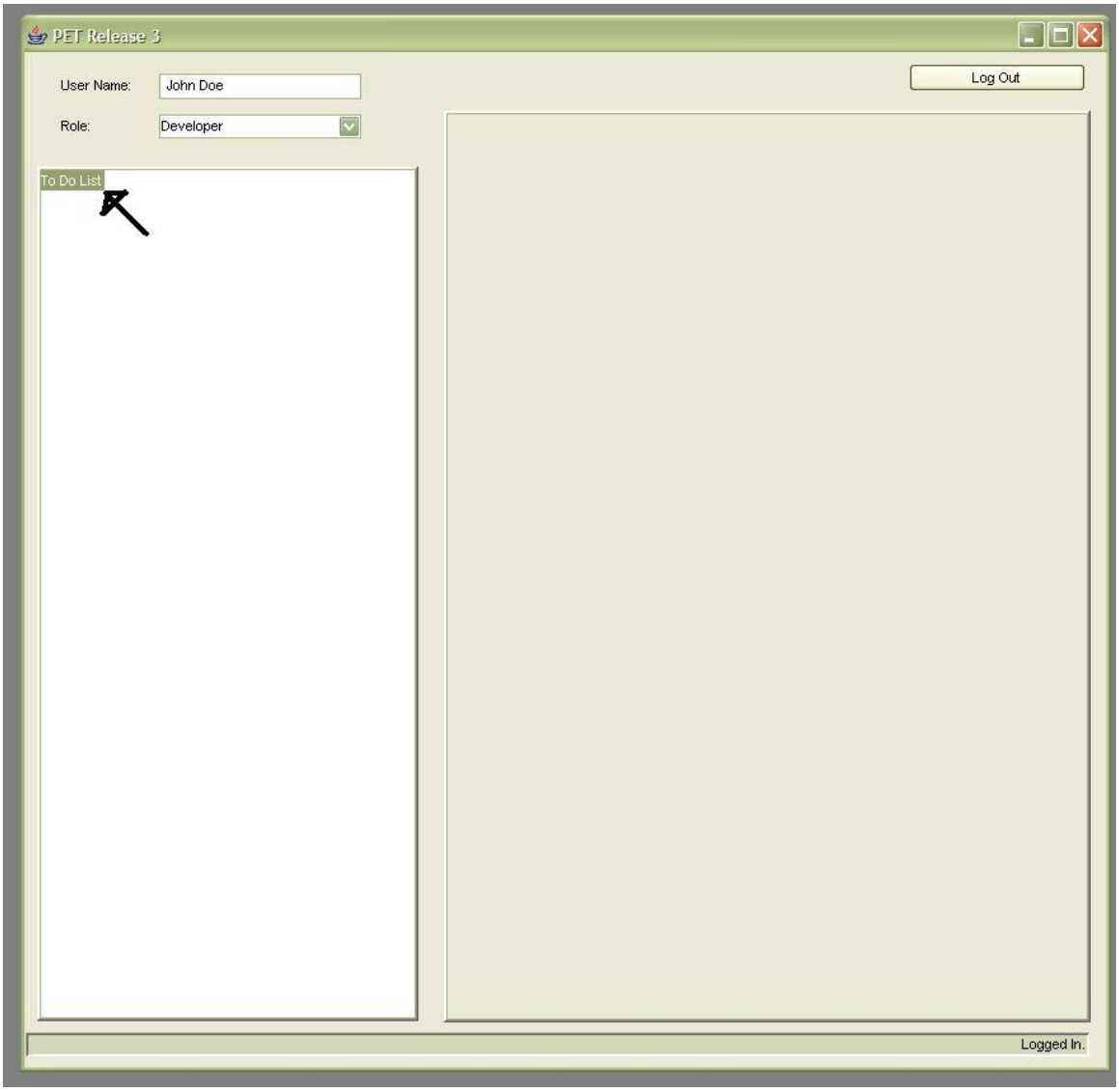

Diagrama 56 - Propuesta de UI: John Doe es presentado con una lista de tareas por hacer

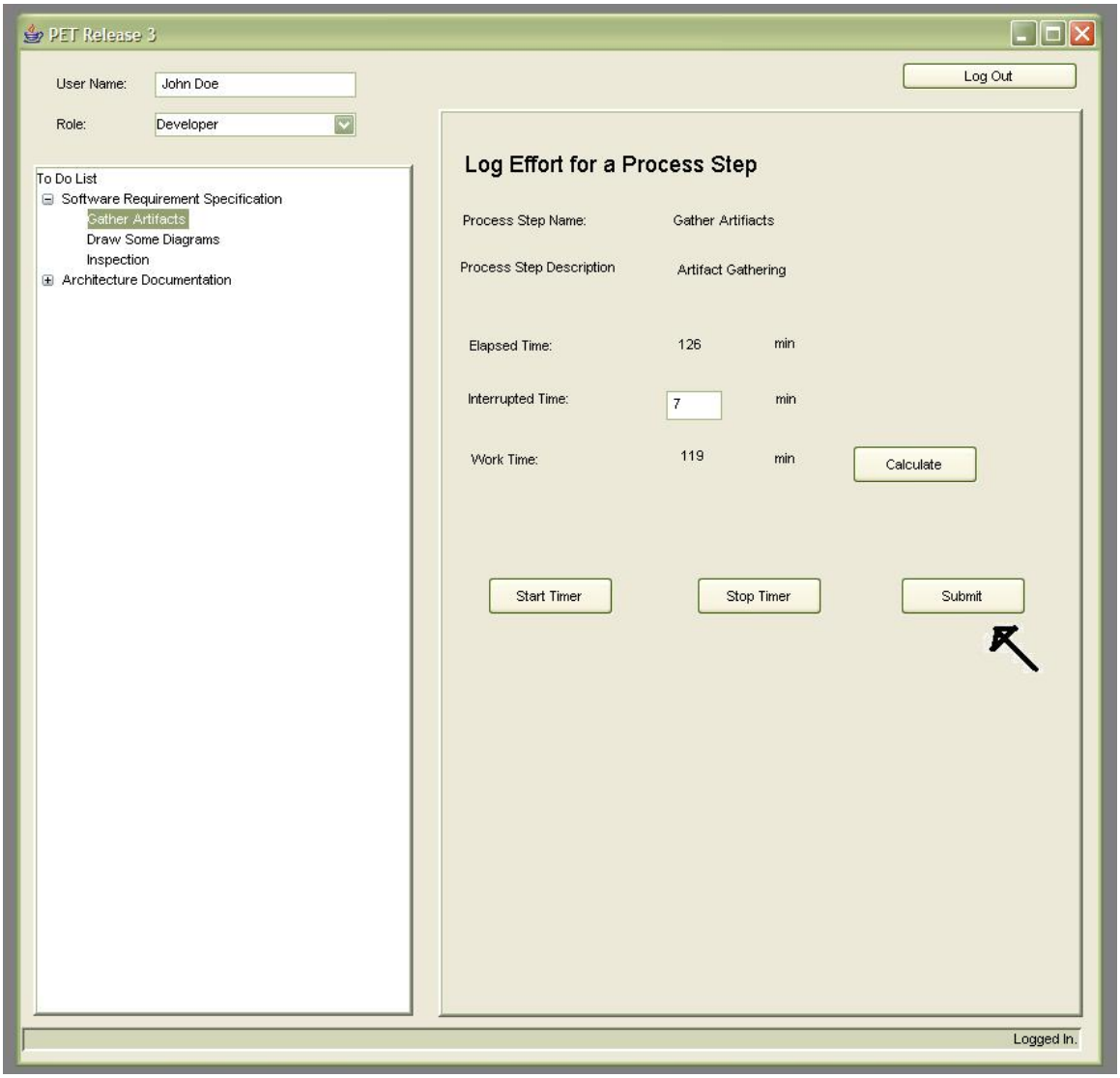

Diagrama 57 - Propuesta de UI: La lista de tareas por hacer se expande al ser seleccionado, mostrando las tareas y sus pasos de proceso respectivos. John Doe selecciona el paso en el que está trabajando y registra su esfuerzo

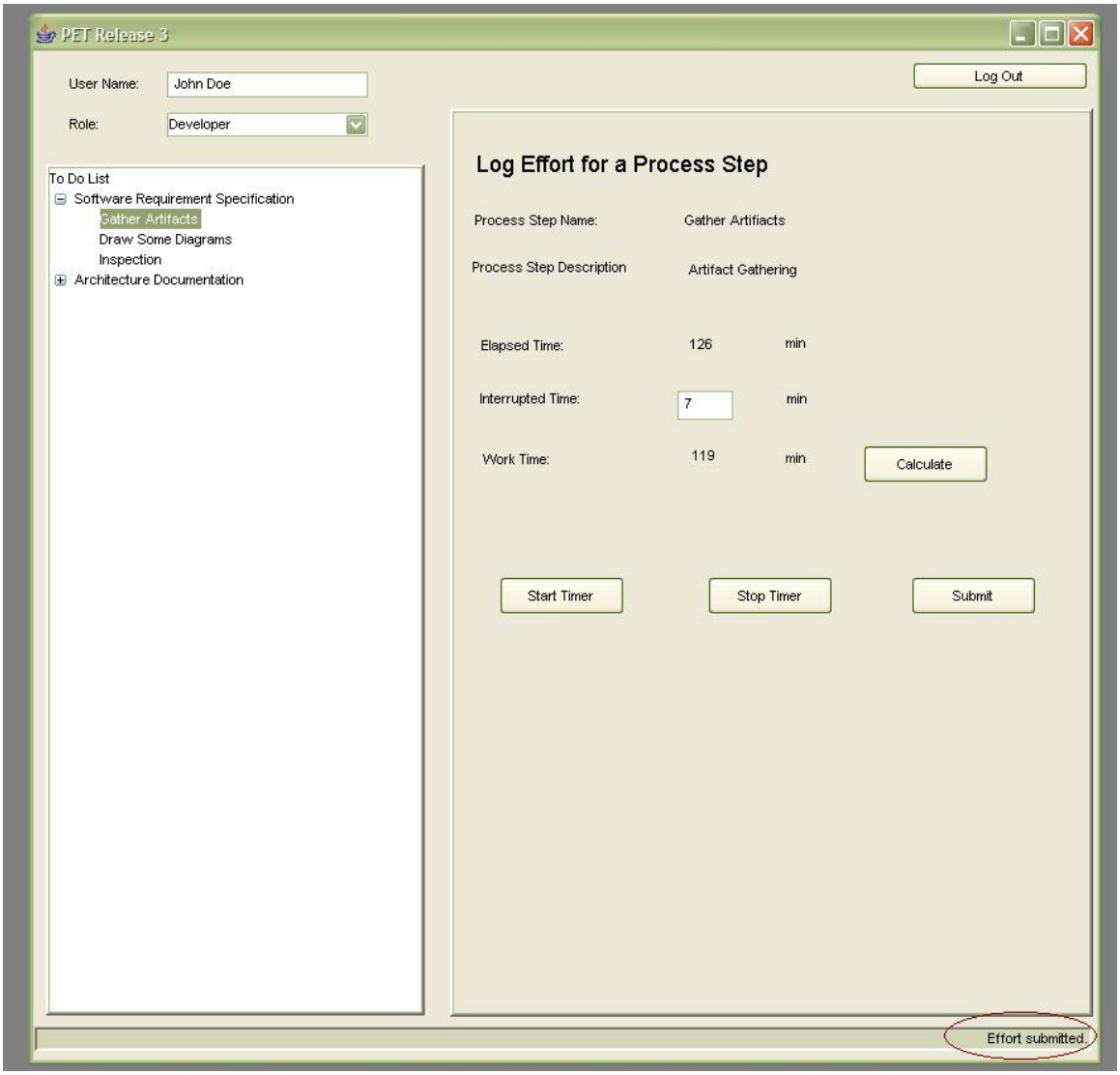

Diagrama 58 - Propuesta de UI: El cliente despliega El cliente despliega el mensaje de éxito en la operación al recibir dicha respuesta del servidor

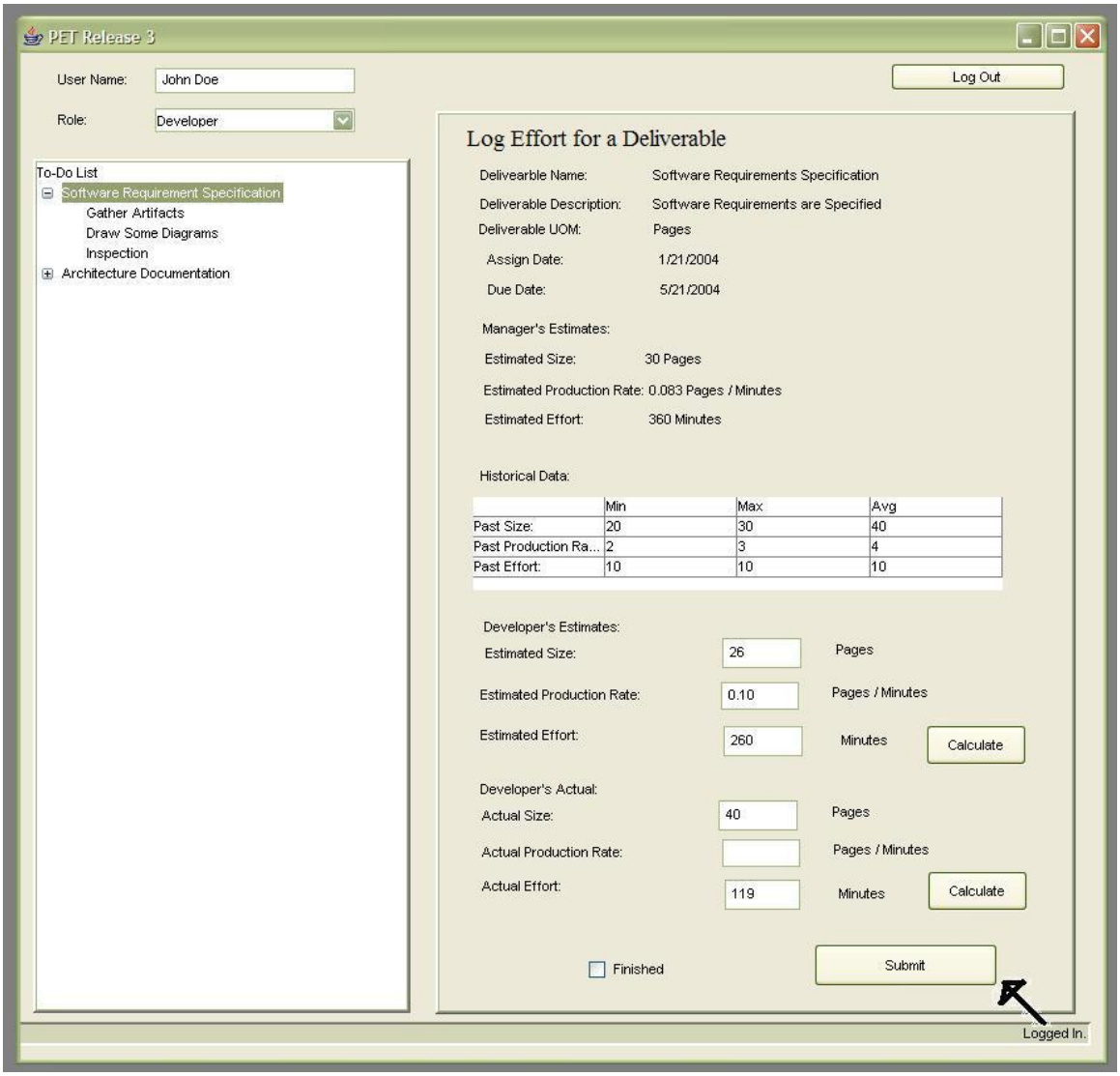

Diagrama 59 - Propuesta de UI: John Doe decide actualizar su status en un entregable en particular, su información histórica es desplegada