

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

PROCESO DE DESARROLLO

Cuando se va a construir un sistema software es necesario conocer un lenguaje de programación, pero con eso no basta. Si se quiere que el sistema sea robusto y mantenible es necesario que el problema sea analizado y la solución sea cuidadosamente diseñada. Se debe seguir un proceso robusto, que incluya las actividades principales. Si se sigue un proceso de desarrollo que se ocupa de plantear cómo se realiza el análisis y el diseño, y cómo se relacionan los productos de ambos, entonces la construcción de sistemas software va a poder ser planificable y repetible, y la probabilidad de obtener un sistema de mejor calidad al final del proceso aumenta considerablemente, especialmente cuando se trata de un equipo de desarrollo formado por varias personas.

Vamos a seguir el método de desarrollo orientado a objetos que propone Craig Larman [Larman99]. Este proceso no fija una metodología estricta, sino que define una serie de actividades que pueden realizarse en cada fase, las cuales deben adaptarse según las condiciones del proyecto que se esté llevando a cabo.

La notación que utilizaremos será la proporcionada por UML. A su vez, vamos a abarcar todo el ciclo de vida, empezando por los requisitos y acabando en el sistema funcionando, proporcionando así una visión completa y coherente de la producción de sistemas software. El enfoque que toma es el de un ciclo de vida iterativo incremental, el cual permite una gran flexibilidad a la hora de adaptarlo a un proyecto y a un equipo de desarrollo específicos. El ciclo de vida está dirigido por casos de uso, es decir, por la funcionalidad que ofrece el sistema a los futuros usuarios del mismo. Así no se pierde de vista la motivación principal que debería estar en cualquier proceso de construcción de software: el resolver una necesidad del usuario/cliente.

Visión General

Las fases que vamos a tener en cuenta son las que se presentan a continuación:

- (1) **Visión:** Define la viabilidad del esfuerzo de desarrollo del software.
- (2) **Planificación y Especificación de Requisitos:** Planificación, definición de requisitos, construcción de prototipos, etc.
- (3) **Construcción:** La construcción del sistema. Las fases dentro de esta etapa son las siguientes:
 - a. **Análisis:** Se analiza el problema a resolver desde la perspectiva de los usuarios y de las entidades externas que van a solicitar servicios al sistema.
 - b. **Diseño:** El sistema se especifica en detalle, describiendo cómo va a funcionar internamente para satisfacer lo especificado en el análisis.
 - c. **Implementación:** Se lleva lo especificado en el diseño a un lenguaje de programación.
 - d. **Pruebas:** Se llevan a cabo una serie de pruebas para corroborar que el software funciona correctamente y que satisface lo especificado en la etapa de Planificación y Especificación de Requisitos.
- (4) **Instalación:** La puesta en marcha del sistema en el entorno previsto de uso.

De ellas, la fase de Construir es la que va a consumir la mayor parte del esfuerzo y del tiempo en un proyecto de desarrollo. Para llevarla a cabo se va a adoptar un enfoque iterativo, tomando en cada iteración un subconjunto de los requisitos (agrupados según casos de uso) y llevándolo a través del análisis y diseño hasta la implementación y pruebas. El sistema va creciendo incrementalmente en cada ciclo.

Visión

Se busca comunicar el ámbito y recursos a los stakeholders del proyecto. Proporciona un enfoque general del desarrollo del software; permite conocer los riesgos, costos, planificación temporal, garantías de calidad y gestión de cambios.

Puntos a documentar:

- (1) Posicionamiento
 - a. **Oportunidad de Negocio:** Se enuncia cómo el cliente se verá beneficiado con la incorporación del nuevo sistema. Este análisis debe estudiarse dentro del marco en que se maneja nuestro cliente, no debe profundizarse en las características del sistema, sino que debe apuntar a evaluar el nuevo "posicionamiento" de nuestro cliente contra sus competidores directos.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

- b. Enunciado del Problema: Se explica cual es la situación actual del cliente, analizando efectos e impacto que afectan su nivel de competitividad, es decir, que problemas presenta hoy el no contar con un sistema como el que se desarrollará.

(2) Stakeholders

- a. Resumen: Lista de los usuarios o roles potenciales del **sistema**, no se incluyen ningún rol asociado al proyecto.
- b. Perfiles: Para cada usuario/rol identificado se marcan los aspectos principales que lo definen dentro de la organización de nuestro cliente.
- c. Necesidades Claves: Definición de los requisitos esenciales que mejorarían su productividad a través del uso del sistema a desarrollar.

(3) Definición del Proyecto

- a. Descripción del Proyecto: Describe el tipo de proyecto a ejecutarse, según sea un nuevo desarrollo, una ampliación, una mejora funcional, etc. con la justificación correspondiente.
- b. Finalidad del Proyecto: Beneficios que se obtendrán con el nuevo sistema, se analizan los procesos que se mejoran a partir de la incorporación de este.
- c. Objetivos: Lista de objetivos impuestos a la ejecución del proyecto.
- d. Alcance: Definición de los límites del proyecto. Puede enunciarse marcando los entregables resultantes.

(4) Organización del Proyecto

- a. Estructura Organizacional: Se marca la estructura jerárquica del equipo de desarrollo del proyecto. Se especifican los datos para poder contactar a cada integrante.
- b. Roles y Responsabilidades: Para cada uno de los roles definidos en la estructura se explica brevemente sus responsabilidades principales dentro del proyecto.

(5) Generalidades del Producto

- a. Perspectivas del Producto: Se explican las funcionalidades principales del producto software a desarrollar.
- b. Requerimientos Funcionales: Lista de requisitos que describen los servicios (funciones) que se esperan del sistema.
- c. Requerimientos no Funcionales: Lista de requisitos que restringen a los requisitos funcionales.

- (6) Supuestos y Restricciones: Se busca determinar todos aquellos factores que a efecto del planeamiento se consideran reales, verdaderos y ciertos e implican un cierto grado de riesgos. Esto es lo que consideramos como supuestos.
Por su parte, cualquier factor que impacte sobre el momento en que una actividad puede ser programada y/o sobre los costos finales del proyecto y/o presupuesto y que afecten al desarrollo del proyecto, lo consideraremos como una restricción.

(7) Estudio de Factibilidad

- a. Factibilidad Técnica: Se responde a ¿es posible? Es pensar si el proyecto tecnológicamente es posible, es decir, si lo que hoy presenta el mercado me sirve para desarrollar el mismo.
- b. Factibilidad Económica: Se responde a ¿es conveniente? Se debe tener en cuenta la cuantificación de todos los costos del proyecto y compararlos con todos los beneficios que trae el proyecto.
- c. Factibilidad Operativa: Se responde a ¿es oportuno? Se busca medir si las condiciones particulares del momento llevan a la ejecución del proyecto.

(8) Costos

- a. Presupuesto: Costo base asignado al proyecto.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

- b. Licenciamiento e Instalación: Explicación de la entrega de licencias que habilitan al cliente al uso del sistema junto con los detalles de instalación.

(9) Plan del Proyecto

- a. Plan de Fases: Cronograma de tiempo que se visualiza a través de un gráfico de Gantt. En él se marcan los puntos de control.
- b. Objetivos e Iteraciones: Definición de la iteraciones del proyecto, especificación de las tareas de cada iteración.
- c. Agenda del Proyecto: Para cada tarea definida, se marcan las fechas de inicio y fin extraídas del gráfico de Gantt.

Fase de Planificación y Especificación de Requisitos

Esta fase se corresponde con la Especificación de Requisitos tradicional ampliada con un Borrador de Modelo Conceptual y con una definición de Casos de Uso de alto nivel. En esta fase se decidiría si se aborda la construcción del sistema mediante desarrollo orientado a objetos o no, por lo que, en principio, es independiente del paradigma empleado posteriormente.

Actividades

Las actividades de esta fase son las siguientes:

- (1) Definir el Plan Borrador.
- (2) Crear el Informe de Investigación Preliminar.
- (3) Definir los Requisitos.
- (4) Registrar Términos en el Glosario. (Continuado en posteriores fases)
- (5) Implementar un Prototipo. (Opcional)
- (6) Definir Casos de Uso. (De alto nivel y esenciales).
- (7) Definir el Modelo Conceptual Borrador. (Puede retrasarse hasta una fase posterior)
- (8) Definir la Arquitectura del Sistema Borrador. (Puede retrasarse hasta una fase posterior)
- (9) Refinar el Plan.

El orden propuesto es el que parece más lógico, y en él los pasos 5 y 7 pueden estar en posiciones distintas.

Requisitos


Un requisito es una descripción de necesidades o aspiraciones respecto a un producto. El objetivo principal de la actividad de definición de requisitos consiste en identificar qué es lo que realmente se necesita. Esto se hace en un modo que sirva de comunicación entre el cliente y el equipo de desarrollo.

Puntos a documentar:

- (1) Propósito.
- (2) Ámbito del sistema.
- (3) Usuarios
- (4) Funciones del sistema.
- (5) Atributos del sistema.

Casos de Uso

Un Caso de Uso es un documento narrativo que describe la secuencia de eventos de un actor (un agente externo) que usa un sistema para completar un proceso [Jacobson92].

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

Caso de Uso de Alto Nivel

Plantilla

DEFINICIÓN	
CASO DE USO:	
ACTORES:	
TIPO:	
DESCRIPCIÓN:	

Descripción de la Plantilla

a. Caso de uso:

Denominación del caso de uso. Utilizar verbo en infinitivo o sustantivo. Ejemplo: Analizar solicitud o Análisis de solicitud.

b. Actores:

Roles y/o usuarios que realizan acciones.

c. Tipo:

Categorización que indica la importancia del mismo. Usar: obligatorio u opcional.

d. Descripción:


Explicación a trazo grueso del caso de uso.

Caso de Uso Expandido

Los casos de uso que se consideren los más importantes y que se considere que son los que más influyen al resto, se describen a un nivel más detallado: en el formato expandido.

Plantilla

DEFINICIÓN	
CASO DE USO:	
OBJETIVO:	
ACTORES:	
TIPO:	
BREVE DESCRIPCIÓN:	
PRECONDICIONES:	
POSTCONDICIONES:	
REGLAS:	
REQUISITOS NO FUNCIONALES ESPECÍFICOS:	
CURSO NORMAL	
Acciones	
CURSOS ALTERNATIVOS	
Acciones	

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

CASOS DE USO DEPENDIENTES	
ES USADO POR:	
ES EXTENDIDO POR:	
ES UN CASO GENERAL DE:	
CASOS DE USO DE LOS QUE DEPENDE	
USA A:	
EXTIENDE A:	
ES UN CASO PARTICULAR DE:	
OTRA INFORMACIÓN	
OBSERVACIONES:	

Descripción de la Plantilla

a. Caso de uso:

Denominación del caso de uso. Utilizar verbo en infinitivo o sustantivo. Ejemplo: Analizar solicitud o Análisis de solicitud.

b. Objetivo:

Objetivo del caso de uso.

c. Actores:

Roles y/o usuarios que realizan acciones.

d. Tipo:

Clasificación del caso de uso según quién tiene la responsabilidad de iniciar el mismo. Usar: abstracto o concreto.

e. Breve descripción:

Descripción breve que permite comprender a grandes rasgos que realiza el caso de uso.

f. Precondiciones:

Condiciones previas que se deben cumplir para que el caso de uso pueda ejecutarse.

g. Postcondiciones:

Condiciones en las cuales debe finalizar la ejecución del caso de uso.

h. Reglas:

Reglas de negocio que se aplican al caso de uso.

a. Requisitos no funcionales específicos:

Requisitos no funcionales que se deben aplicar específicamente al caso de uso.

i. Curso normal:

Lista ordenada de acciones ejecutadas por los actores y las respuestas correspondientes devueltas por el sistema.

j. Cursos alternativos:

Lista de alternativas que deben considerarse para cada acción o respuesta especificada en el curso normal.

k. Casos de uso dependientes:

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

Casos de uso que dependen de éste:

i. Es usado por:

Lista de casos de uso que lo usan.

ii. Es extendido por:

Lista de casos de uso que lo extienden.

iii. Es un caso general de:

Casos de uso que son casos particulares de éste.

l. Casos de uso de los que depende:

Casos de uso de los que depende éste:

iv. Usa a:

Lista de casos de uso que son utilizados por éste.

v. Extiende a:

Lista de casos de uso a los que extiende.

vi. Es un caso particular de:

Caso de uso del cual se deriva.

b. Observaciones:

Aclaraciones.

Identificación de Casos de Uso

La identificación de casos de uso requiere un conocimiento medio acerca de los requisitos, y se basa en la revisión de los documentos de requisitos existentes, y en el uso de la técnica de *brainstorming* entre los miembros del equipo de desarrollo.

Como guía para la identificación inicial de casos de uso hay dos métodos:

(1) Basado en actores:

- Identificar los actores relacionados con el sistema y/o la organización.
- Para cada actor, identificar los procesos que inicia o en los que participa.

(2) Basado en eventos:

- Identificar los eventos externos a los que el sistema va a tener que responder.
- Relacionar los eventos con actores y casos de uso.

Construcción del modelo de Casos de Uso

Para construir el Modelo de Casos de Uso en la fase de Planificación y Especificación de Requisitos se siguen los siguientes pasos:

- Después de listar las funciones del sistema, se definen los límites del sistema y se identifican los actores y los casos de uso.
- Se escriben todos los casos de uso en el formato de *alto nivel*. Se categorizan como obligatorios u opcionales.
- Se dibuja el Diagrama de Casos de Uso. Se relacionan los casos de uso y se ilustran las relaciones en el Diagrama de Casos de Uso.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

(4) Los casos de uso más críticos, importantes y que conllevan un mayor riesgo, se describen en el formato expandido esencial. Se deja la definición en formato expandido esencial del resto de casos de uso para cuando sean tratados en posteriores ciclos de desarrollo, para no tratar toda la complejidad del problema de una sola vez.

(5) Ordenar según prioridad los casos de uso.

Diagrama de Casos de Uso

Un *diagrama de casos de uso* es un diagrama que muestra un conjunto de casos de uso, actores y sus relaciones. Normalmente un diagrama de caso de uso contiene:

- Casos de uso
- Actores
- Relaciones de dependencia, generalización y asociación

Al igual que los demás diagramas, los diagramas de casos de uso pueden contener notas y restricciones.

Los diagramas de casos de uso también pueden contener paquetes, que se emplean para agrupar elementos del modelo en partes mayores. Ocasionalmente, se pueden incluir instancias de casos de uso en los diagramas, especialmente cuando se quiere visualizar un sistema específico en funcionamiento.

Los casos de uso también pueden organizarse especificando relaciones de generalización, inclusión y extensión entre ellos. Estas relaciones se utilizan para factorizar el comportamiento común (extrayendo ese comportamiento de los casos de uso en los que se incluye) y para factorizar variantes (poniendo ese comportamiento en otros casos de uso que lo extienden).

La generalización entre casos de uso es como la generalización entre clases. Aquí significa que el caso de uso hijo hereda el comportamiento y el significado del caso de uso padre; el hijo puede añadir o redefinir el comportamiento del padre; el hijo puede ser colocado en cualquier lugar donde aparezca el padre (tanto el padre como el hijo pueden tener instancias concretas). La generalización entre casos de uso se representa con una línea continua con una punta de flecha vacía, al igual que la generalización entre clases.

Una relación de inclusión entre casos de uso significa que un caso de uso base incorpora explícitamente el comportamiento de otro caso de uso en el lugar especificado en el caso base. El caso de uso incluido nunca se encuentra aislado, sino que es instanciado solo como parte de algún caso de uso base más amplio que lo incluye. La inclusión puede verse como que el caso de uso base toma el comportamiento del caso de uso proveedor.

La relación de inclusión se usa para evitar describir el mismo flujo de eventos repetidas veces, poniendo el comportamiento común en un caso de uso aparte (que será incluido por un caso de uso base). La relación de inclusión es básicamente un ejemplo de delegación: se toma un conjunto de responsabilidades del sistema y se captura en un sitio (el caso de uso a incluir en otros), a continuación se permite que otras partes del sistema (otros casos de uso) incluyan la nueva agregación de responsabilidades, siempre que se necesite usar esa funcionalidad.


Una relación de inclusión se representa como una dependencia, estereotipada con <<usa>>.

Una relación de extensión entre casos de uso significa que un caso de uso base incorpora implícitamente el comportamiento de otro caso de uso en el lugar especificado indirectamente por el caso de uso que extiende al base. El caso de uso base puede estar aislado pero, bajo ciertas condiciones, su comportamiento puede extenderse con el comportamiento de otro caso de uso. Este caso de uso base puede extenderse solo en cierto puntos, llamados, como era de esperar, puntos de extensión. La extensión se puede ver como que el caso de uso que extiende incorpora su comportamiento en el caso de uso base.

Una relación de extensión se utiliza para modelar la parte de un caso de uso que el usuario puede ver como comportamiento adicional del sistema. De esta forma, se separa el comportamiento opcional del obligatorio. También se puede utilizar una relación de extensión para modelar un subflujo separado que se ejecuta solo bajo ciertas condiciones. Por último, se puede utilizar una relación de extensión para modelar varios flujos que se pueden insertar en un punto dado, controlados por la interacción explícita con un actor.

Una relación de extensión se representa como una dependencia, estereotipada como <<extiende>>.

Propiedad	Extensión	Inclusión	Generalización
Comportamiento base	Caso de uso base	Caso de uso base	Caso de uso padre
Comportamiento añadido	Caso de uso de extensión	Caso de uso incluido	Caso de uso hijo
Dirección de referencia	El caso de uso extensor referencia al caso de uso base	El caso de uso base referencia al caso de uso incluido	El caso de uso hijo referencia el caso de uso padre
¿Base modificada?	La extensión modifica implícitamente el comportamiento de la base. La base debe estar	La inclusión modifica explícitamente el efecto de la base. La base puede estar bien formada o no sin	El efecto de ejecutar el padre no se modifica por el hijo. Para obtener efectos distintos, se debe

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

	formada sin extensión, pero si la extensión está presente, una instancia de la base puede ejecutar la extensión	la inclusión, pero una instanciación de la base ejecuta la inclusión.	instanciar el hijo y no el padre.
¿Instanciable?	La extensión no es necesariamente instanciable. Puede ser un fragmento	La inclusión no es necesariamente instanciable, puede ser un fragmento	El hijo no es necesariamente instanciable. Puede ser abstracto
¿Puede acceder a los atributos de la base?	Las extensiones pueden acceder y modificar el estado de la base	La inclusión puede acceder al estado de la base. La base debe proveer los atributos apropiados esperados por la inclusión	El hijo puede acceder y modificar el estado de la base (por los mecanismos usuales de herencia)
¿Puede la base ver al otro?	La base no puede ver las extensiones y debe estar bien formada en su ausencia	La base ve la inclusión y puede depender de sus efectos, pero no puede acceder a sus atributos	El padre no puede ver al hijo, y debe estar bien formado en su ausencia
Repetición	Depende de la condición	Exactamente una repetición	El hijo controla su propia ejecución

Planificación de Casos de Uso según ciclos de desarrollo

La decisión de qué partes del sistema abordar en cada ciclo de desarrollo se va a tomar basándose en los casos de uso. Esto es, a cada ciclo de desarrollo se le va a asignar la implementación de uno o más casos de uso, o versiones simplificadas de casos de uso.

Para tomar la decisión de qué casos de uso se van a tratar primero es necesario ordenarlos según prioridad. Las características de un caso de uso específico que van a hacer que un caso de uso tenga una prioridad alta son las siguientes:

- (1) Impacto significativo en el diseño de la arquitectura. Por ejemplo, si aporta muchas clases al modelo del dominio o requiere persistencia en los datos.
- (2) Se obtiene una mejor comprensión del diseño con un nivel de esfuerzo relativamente bajo.
- (3) Incluye funciones complejas, críticas en el tiempo o de nivel elevado de riesgo.
- (4) Implica bien un trabajo de investigación significativa, o bien el uso de una tecnología nueva o arriesgada.
- (5) Representa un proceso de gran importancia en la línea de negocio.
- (6) Supone directamente un aumento de beneficios o una disminución de costes.
- (7) Para realizar la clasificación se puede asignar a cada caso de uso una valoración numérica de cada uno de estos puntos, para conseguir una puntuación total aplicando pesos a cada apartado.

Fase de Análisis

En la fase de Análisis de un ciclo de desarrollo se *investiga* sobre el problema, sobre los conceptos relacionados con el subconjunto de casos de uso que se esté tratando. Se intenta llegar a una buena comprensión del problema por parte del equipo de desarrollo, sin entrar en cómo va a ser la solución en cuanto a detalles de implementación.


Cuando el ciclo de desarrollo no es el primero, antes de la fase de Análisis hay una serie de actividades de planificación. Estas actividades consisten en actualizar los modelos que se tengan según lo que se haya implementado, pues siempre se producen desviaciones entre lo que se ha analizado y diseñado y lo que finalmente se construye. Una vez se tienen los modelos acordes con lo implementado se empieza el nuevo ciclo de desarrollo con la fase de Análisis.

En esta fase se trabaja con los modelos de Análisis contruidos en la fase anterior, ampliándolos con los conceptos correspondientes a los casos de uso que se traten en el ciclo de desarrollo actual.

Actividades

Las actividades de la fase de Análisis son las siguientes:

- (1) Definir Casos de Uso Esenciales en formato expandido. (*Si no están definidos*)
- (2) Refinar los Diagramas de Casos de Uso.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

- (3) Refinar el Modelo Conceptual.
- (4) Refinar el Glosario. (*Continuado en posteriores fases*)
- (5) Definir los Diagramas de Secuencia del Sistema.
- (6) Definir Contratos de Operación.
- (7) Definir Diagramas de Estados. (*Opcional*)

Modelo Conceptual

Una parte de la investigación sobre el dominio del problema consiste en identificar los conceptos que lo conforman. Para representar estos conceptos se va a usar un Diagrama de Estructura Estática de UML, que es el Modelo Conceptual.

En el Modelo Conceptual se tiene una representación de conceptos del mundo real, no de componentes software.

El objetivo de la creación de un Modelo Conceptual es aumentar la comprensión del problema. Por tanto, a la hora de incluir conceptos en el modelo, es mejor crear un modelo con muchos conceptos que quedarse corto y olvidar algún concepto importante.

Identificación de Conceptos

Para identificar conceptos hay que basarse en el documento de Especificación de Requisitos y en el conocimiento general acerca del dominio del problema.

Otro consejo para identificar conceptos consiste en buscar sustantivos en los documentos de requisitos o, más concretamente, en la descripción de los casos de uso.

Identificación de Asociaciones

Una asociación es una relación entre conceptos que indica una conexión con sentido y que es de interés en el conjunto de casos de uso que se está tratando.

Una vez identificadas las asociaciones se representan en el Modelo Conceptual con la multiplicidad adecuada.

Identificación de Atributos

Es necesario incorporar al Modelo Conceptual los atributos necesarios para satisfacer las necesidades de información de los casos de uso que se estén desarrollando en ese momento.

Los atributos deben tomar valor en tipos simples (número, texto, etc.), pues los tipos complejos deberían ser modelados como conceptos y ser relacionados mediante asociaciones. Incluso cuando un valor es de un tipo simple es más conveniente representarlo como concepto en las siguientes ocasiones:

- (1) Se compone de distintas secciones. Por ejemplo: un número de teléfono.
- (2) Tiene operaciones asociadas, tales como validación.
- (3) Tiene otros atributos. Por ejemplo un precio de oferta puede tener fecha de fin.
- (4) Es una cantidad con una unidad. Ejemplo: El precio, que puede estar en pesos o en dólar.

Una vez definidos los atributos se tiene ya un Modelo Conceptual. Este modelo no es un modelo definitivo, pues a lo largo del Análisis y del Diseño se va refinando según se le añaden conceptos que se habían pasado por alto.

Glosario

En el glosario debe aparecer una descripción textual de cualquier elemento de cualquier modelo, para eliminar toda posible ambigüedad.

Plantilla

TÉRMINO	CATEGORÍA	DEFINICIÓN

Descripción de la Plantilla

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

a. Término:

Palabra a definir.

b. Categoría:

Modelo en el cual se aplica el término.

c. Definición:

Explicación del término, debe ser una definición no técnica.

Diagrama de Secuencia del Sistema

Además de investigar sobre los conceptos del sistema y su estructura, también es preciso investigar en el Análisis sobre el comportamiento del sistema, visto éste como una caja negra. Una parte de la descripción del comportamiento del sistema se realiza mediante los Diagramas de Secuencia del Sistema.

En cada caso de uso se muestra una interacción de actores con el sistema. En esta interacción los actores generan eventos, solicitando al sistema operaciones.

Los casos de uso representan una interacción genérica. Una instancia de un caso de uso se denomina **escenario**, y muestra una ejecución real del caso de uso, con las posibles bifurcaciones y alternativas resueltas de forma particular.

Un Diagrama de Secuencia de Sistema se representa usando la notación para diagramas de secuencia de UML. En él se muestra para un escenario particular de un caso de uso los eventos que los actores generan, su orden, y los eventos que se intercambian entre sistemas.

Para cada caso de uso que se esté tratando se realiza un diagrama para el curso típico de eventos, y además se realiza un diagrama para los cursos alternativos de mayor interés.

Los eventos del sistema deberían expresarse en base a la noción de operación que representan, en vez de en base a la interfaz particular.

Construcción de un diagrama de secuencia del sistema

Para construir un Diagrama de Secuencia del Sistema para el curso típico de eventos de un caso de uso, se siguen los siguientes pasos:

- (1) Representar el sistema como un objeto con una línea debajo.
- (2) Identificar los actores que directamente operan con el sistema, y dibujar una línea para cada uno de ellos.
- (3) Partiendo del texto del curso típico de eventos del caso de uso, identificar los eventos (externos) del sistema que cada actor genera y representarlos en el diagrama.
- (4) Opcionalmente, incluir el texto del caso de uso en el margen del diagrama.
- (5) Los eventos del sistema deberían expresarse en base a la noción de operación que representan, en vez de en base a la interfaz particular. Por ejemplo, se prefiere "finOperación" a "presionadaTeclaEnter", porque captura la finalidad de la operación sin realizar compromisos en cuanto a la interfaz usada.


Contrato de operaciones

Una vez se tienen las Operaciones del Sistema identificadas en los Diagramas de Secuencia, se describe mediante contratos el comportamiento esperado del sistema en cada operación.

Un Contrato es un documento que describe qué es lo que se espera de una operación. Tiene una redacción en estilo declarativo, enfatizando en el *qué* más que en el *cómo*. Lo más común es expresar los contratos en forma de pre y post condiciones entorno a cambios de estado.

Se puede escribir un contrato para un método individual de una clase software, o para una operación del sistema completa. En este punto se verá únicamente éste último caso.

Un Contrato de Operación del Sistema describe cambios en el estado del sistema cuando una operación del sistema es invocada.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

CONTRATO	
NOMBRE:	
RESPONSABILIDADES:	
REFERENCIA CRUZADA:	
NOTAS:	
EXCEPCIONES:	
SALIDA:	
PRECONDICIONES:	
POSTCONDICIONES:	

Descripción de la Plantilla

a. Nombre:

Nombre de la operación o rutina y parámetros.

b. Responsabilidades:

Una descripción informal de las responsabilidades que la operación debe desempeñar.

c. Referencias Cruzadas:

Número de referencia en los requisitos de funciones del sistema, casos de uso, etc.

d. Notas:

Comentarios de diseño, algoritmo, etc.

e. Excepciones:

Deben expresarse casos excepcionales e indicar el tratamiento de cada excepción.

Como sabemos una llamada a una operación tiene éxito si termina su ejecución en un estado en el que satisface el contrato de la rutina. Fracasa si no tiene éxito.

De la noción de fracaso se puede derivar una definición precisa de excepciones. Una rutina fracasa debido a algún suceso específico (desbordamiento aritmético, violación de una aserción, etc.) que interrumpe su ejecución.

Respecto al tratamiento de excepciones, hay solo dos respuestas legítimas a una excepción que ocurra durante la ejecución de una rutina:

- (1) *Reintento*: intentar cambiar las condiciones que condujeron a la excepción y ejecutar de nuevo la rutina desde el comienzo.
- (2) *Fracaso* (también conocido como "pánico organizado"): limpiar el entorno, terminar la llamada e informar del fallo a quién hiciera la llamada.

f. Salida:

Salidas que no corresponden a la interfaz de usuario, como mensajes o registros que se envían fuera del sistema. (En la mayor parte de las operaciones del sistema este apartado queda vacío).

g. Precondiciones:

Asunciones acerca del estado del sistema antes de ejecutar la operación. Establece las propiedades que se tienen que cumplir cada vez que se llame a la operación. Expresa las restricciones bajo las que una rutina funcionará correctamente.

h. Postcondiciones:

El estado del sistema después de completar la operación. Establece las propiedades que debe garantizar la operación cuando retorne.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

Se basan en el Modelo Conceptual, en los cambios que sufren los elementos del mismo una vez que se ha realizado la operación. Sin duda aparecerán nuevos conceptos, atributos o asociaciones que deberán ser agregadas al Modelo Conceptual

Es mejor usar el tiempo pasado o el pretérito perfecto al redactar una postcondición, para enfatizar que se trata de declaraciones sobre un cambio en el estado que ya ha pasado. Por ejemplo, es mejor decir “se ha creado una Sesión” que decir “crear una Sesión”.

Construcción de un Contrato

Los pasos a seguir para construir un contrato son los siguientes:

- (1) Identificar las operaciones del sistema a partir de los Diagramas de Secuencia del Sistema.
- (2) Para cada operación del sistema construir un contrato.
- (3) Empezar escribiendo el apartado de *Responsabilidades*, describiendo informalmente el propósito de la operación. Este es el apartado más importante del contrato.
- (4) A continuación rellenar el apartado de *Postcondiciones*, describiendo declarativamente los cambios de estado que sufren los objetos en el Modelo Conceptual. Puede ser que este apartado quede vacío si no cambia el valor de ningún dato de los que maneja el sistema (por ejemplo en una operación del sistema que tan solo se encarga de sacar por pantalla algo al usuario).
- (5) Para describir las postcondiciones, usar las siguientes categorías:
 - a. Creación y eliminación de instancias.
 - b. Modificación de atributos.
 - c. Asociaciones formadas y retiradas.
- (6) Completar el resto de apartados de ser requerido.

Fase de Diseño

En la fase de Diseño se crea una solución a nivel lógico para satisfacer los requisitos, basándose en el conocimiento reunido en la fase de Análisis.

Actividades

Las actividades que se realizan en la etapa de Diseño son las siguientes:

- (1) Definir los Casos de Uso Reales.
- (2) Definir Informes e Interfaz de Usuario.
- (3) Refinar la Arquitectura del Sistema.
- (4) Definir los Diagramas de Interacción.
- (5) Definir el Diagrama de Clases de Diseño. (*En paralelo con los Diagramas de Interacción*)
- (6) Definir el Modelo de Base de Datos.

Casos de Uso Reales

Un Caso de Uso Real describe el diseño real del caso de uso según una tecnología concreta de entrada y de salida y su implementación. Si el caso de uso implica una interfaz de usuario, el caso de uso real incluirá maquetas de las ventanas y detalles de la interacción a bajo nivel con los *widgets* (botón, lista seleccionable, campo editable, etc.) de la ventana.

Como alternativa a la creación de los Casos de Uso Reales, el desarrollador puede crear maquetas de la interfaz en papel, y dejar los detalles para la fase de implementación.

Diagrama de Interacción

Los Diagramas de Interacción muestran el intercambio de mensajes entre instancias del modelo de clases para cumplir las postcondiciones establecidas en un contrato.

Hay dos clases de Diagramas de Interacción:

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

(1) Diagramas de Colaboración.

(2) Diagramas de Secuencia.

De entre ambos tipos se prefieren los Diagramas de Colaboración por su expresividad y por su economía espacial (una interacción compleja puede ser muy larga en un Diagrama de Secuencia).

La creación de los Diagramas de Colaboración de un sistema es una de las actividades más importantes en el desarrollo orientado a objetos, pues al construirlos se toman unas decisiones clave acerca del funcionamiento del futuro sistema. La creación de estos diagramas, por tanto, debería ocupar un porcentaje significativo en el esfuerzo dedicado al proyecto entero.

Creación de Diagramas de Colaboración

Para crear los Diagramas de Colaboración se pueden seguir los siguientes consejos:

- (1) Crear un diagrama separado para cada operación del sistema en desarrollo en el ciclo de desarrollo actual.
- (2) Para cada evento del sistema, hacer un diagrama con él como mensaje inicial.
- (3) Si el diagrama se complica, dividirlo en diagramas más pequeños.
- (4) Usando los apartados de responsabilidades y de postcondiciones del contrato de operación, y la descripción del caso de uso como punto de partida, diseñar un sistema de objetos que interaccionan para llevar a cabo las tareas requeridas.
 La capacidad de realizar una buena asignación de responsabilidades a los distintos objetos es una habilidad clave, y se va adquiriendo según aumenta la experiencia en el desarrollo orientado a objetos.
 Booch, Rumbaugh y Jacobson definen **responsabilidad** como “un contrato u obligación de una clase o tipo”. Las responsabilidades están ligadas a las obligaciones de un objeto en cuanto a su comportamiento. Básicamente, estas responsabilidades son de los dos siguientes tipos:

a. Conocer:

- i. Conocer datos privados encapsulados.
- ii. Conocer los objetos relacionados.
- iii. Conocer las cosas que puede calcular o derivar.

b. Hacer:

- i. Hacer algo él mismo.
- ii. Iniciar una acción en otros objetos.
- iii. Controlar y coordinar actividades en otros objetos.

Por ejemplo, puedo decir que “un *Recibo* es responsable de imprimirse” (tipo hacer), o que “una *Transacción* es responsable de saber su fecha” (tipo conocer). Las responsabilidades de tipo “conocer” se pueden inferir normalmente del Modelo Conceptual.

Una responsabilidad no es lo mismo que un método, pero los métodos se implementan para satisfacer responsabilidades.

Definición de mensajes

El término mensaje alude a enviar una señal de un objeto (el emisor) a otro u otros objetos (los receptores); también puede ser la llamada a una operación aplicada a un objeto (el receptor) por parte de otro objeto (el emisor, quien hace la llamada). La implementación de un mensaje puede adoptar distintas formas, tales como una llamada a procedimiento, una comunicación de procesos entre hilos activos, el envío explícito de un evento y así sucesivamente.

Por ello, un mensaje posee un emisor, un receptor y una acción.

Dentro de una interacción, el emisor es el rol de clasificador que envía el mensaje. El receptor es el rol de clasificador que recibe el mensaje. La acción es una llamada; una señal, una operación local aplicada al emisor o una acción primitiva tal como crear o destruir. La acción incluye una lista de argumentos, una expresión para un conjunto de receptores y una referencia de la operación o señal implicadas. También puede incluir una especificación de condicionalidad e iteración de la ejecución del mensaje.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

Dentro de una interacción, los mensajes están relacionados mediante la relación predecesor-sucesor y la relación emisor-receptor. Esta última relación es aplicable a los métodos de procedimientos. Cada llamada añade un nivel de anidamiento a la secuencia. Dentro de una llamada, los mensajes se ordenan secuencialmente; existe la posibilidad de subsecuencias concurrentes.

La relación predecesor-sucesor (o de secuenciación) organiza los mensajes del hilo en una secuencia lineal. Si hay dos mensajes con un predecesor común y no están secuenciados de alguna otra forma, entonces se pueden ejecutar concurrentemente. Si un mensaje posee múltiples predecesores, tiene que esperar hasta que finalicen todos ellos. Este mensaje es un punto de sincronización.

La relación emisor-receptor (o de activación) define una estructura de procedimientos anidada. El mensaje que llama a un procedimiento (empleando una acción de llamada) es el activador de todos los mensajes que forman el cuerpo del procedimiento invocado. Entre ellos, los mensajes invocados tienen una relación predecesor-sucesor que establece el orden relativo (y puede permitir la concurrencia).

Si un mensaje es una llamada, entonces se bloquea al emisor hasta que el procedimiento invocado finaliza y retorna. Sin embargo, si el receptor maneja la operación como evento de llamada entonces el retorno se produce cuando finaliza la transición inicial, tras la cual el emisor recupera el control y el receptor puede reanudar su propia ejecución.

Las relaciones de secuenciación y de activador solo relacionan a aquellos mensajes que procesan la misma interacción.

En Diagrama de Secuencia

En un Diagrama de Secuencia, un mensaje se representa mediante una flecha continua que va desde línea de vida de un objeto (emisor) hasta la línea de vida de otro objeto (el destino). Si la flecha es perpendicular a las líneas de vida, entonces se considera que la transmisión del mensaje es instantánea o al menos muy rápida en comparación con los mensajes externos. Si la flecha es oblicua, entonces se considera que la transmisión del mensaje tiene una duración, durante la cual se podrían enviar otros mensajes. En el caso de un mensaje que vaya del objeto hasta sí mismo, la flecha puede comenzar y acabar en la misma línea de vida. Las flechas de mensajes se organizan por orden secuencial verticalmente, empezando de arriba y avanzando hacia abajo. Si dos mensajes son concurrentes, su orden relativo no es significativo. Los mensajes pueden tener números de secuencia pero dado que el orden relativo de mensajes se muestra visualmente, los números de secuencia suelen omitirse.

Retardo de transmisión: Normalmente las flechas se dibujan horizontalmente, lo cual indica que la duración necesaria para enviar el mensaje es atómica, esto es, que es breve comparación con la granularidad que tiene la interacción y nada más puede “suceder” durante la transmisión del mensaje. Si el mensaje requiere un cierto tiempo para entregarlo, durante el cual puede ocurrir algo más (tal como un mensaje en dirección opuesta) entonces la flecha de mensaje se puede inclinar hacia abajo para que la punta de la flecha quede por debajo del origen de la misma.

Bifurcaciones: Las bifurcaciones se muestran mediante múltiples flechas que parten de un mismo punto y cada una de ellas está etiquetada con su propia condición de guarda. Dependiendo de si las condiciones de guarda son o no mutuamente excluyentes, esta estructura puede representar la condicionalidad o la concurrencia.

Iteración: Los conjuntos conectados de mensajes pueden estar encerrados y marcados como iteración. Los marcadores de iteración denotan que el conjunto de mensajes puede aparecer en múltiples ocasiones. Se puede especificar la condición de continuación para un procedimiento en la parte inferior de la iteración. Si hay concurrencia, entonces algunos mensajes del diagrama pueden formar parte de la iteración y otros pueden ejecutarse individualmente.

En Diagrama de Colaboración

En un Diagrama de Colaboración, los mensajes se representan mediante una flechita con etiqueta asociada a una ruta que une los objetos emisor y receptor. La ruta es la que se emplea para acceder al objeto blanco. La flecha apunta a lo largo de la ruta en la dirección del objeto destino. En el caso de que sea un mensaje que sale del objeto para volver a sí mismo, el mensaje aparece en una ruta que vuelve al mismo objeto y el extremo que lleva al blanco posee la etiqueta <<self>>.

Se puede asociar más de un mensaje a un enlace, tanto en la misma dirección como en direcciones distintas. El orden relativo de mensajes se muestra mediante la porción de número de secuencia que aparece en la etiqueta del mensaje.

Para ambos Diagramas

La flecha de mensaje tiene en su etiqueta el nombre del mensaje (el nombre de la señal o de la operación) y los valores de sus argumentos. La flecha también puede tener en su etiqueta un número de secuencia, para mostrar la posición del mensaje dentro de la interacción global. Los números de secuencia se pueden omitir en los diagramas de secuencias, en los cuales la situación física de la flecha muestra la posición relativa del mensaje, pero en los diagramas de colaboración resultan necesarios. Los números de secuencia resultan útiles en los dos tipos de

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

diagramas para identificar los hilos de control concurrentes. También se puede poner en la etiqueta del mensaje una condición de guarda.

Tipo de flujo de control: Es posible emplear las siguientes variantes de puntas de flecha para mostrar diferentes clases de flujo de control de los mensajes.

Punta de flecha con relleno

Llamada a procedimiento u otro flujo de control anidado. La secuencia anidada completa debe finalizar antes de reanudar la secuencia de nivel externo. Se puede emplear en llamadas normales a procedimiento. También se puede usar con objetos activos concurrentemente cuando uno de ellos envía una señal y espera a que finalice una secuencia de comportamiento anidada.

Punta de flecha sin relleno

Flujo de control plano. Cada flecha muestra la progresión al próximo paso de la secuencia. En el caso de procedimientos anidados, esto se corresponde con un barrido de las hojas del árbol de acciones efectuado lateralmente desde el fondo.

Media punta de flecha

Flujo de control asíncrono. Se emplea en lugar de la cabeza de flecha hueca para mostrar explícitamente un mensaje asíncrono entre dos objetos de una secuencia de procedimiento.

Flecha discontinua con punta sin relleno

Retorno de una llamada a procedimiento. La flecha de retorno puede suprimirse, por cuanto queda implícita al final de la activación.

Otras variantes

Se pueden mostrar otras clases de control, tales como "rehusar" o "tiempo agotado", pero se tratan como extensiones del núcleo de UML.

Etiqueta de mensaje: La etiqueta tiene la sintaxis siguiente:

predecesor _{opt} condición-de-guarda _{opt} expresión-secuencia _{opt}

lista-valores-retorno := _{opt} nombre-mensaje (argumentos _{lista,})

La etiqueta denota el mensaje enviado, sus argumentos y valores proporcionados y la situación del mensaje dentro de la interacción más extensa, incluyendo el anidado de llamadas, iteración, bifurcación, concurrencia y sincronización.

Predecesor: En una colaboración, el predecesor es una lista de números de secuencia separados por comas, y seguidos por una barra (/).

número-secuencia _{lista,} /

La cláusula se omite si la lista está vacía.

Todo **número-secuencia** es una **expresión-secuencia** sin términos de recurrencia. Tiene que coincidir con el **número-secuencia** de algún otro mensaje.

El significado es que el flujo de mensajes no queda habilitado mientras no se hayan producidos todos los flujos de mensajes cuyos números de secuencia estén enumerados (un hilo puede ir más allá del flujo de mensajes y la condición de guarda seguirá cumpliéndose). Por tanto, la condición de guarda representa una sincronización de hilos.


Si observamos el mensaje que corresponde al número de secuencia precedente numéricamente es un predecesor implícito y no necesita ser enumerado explícitamente. Todos los números de secuencia que tienen igual prefijo forman una secuencia. El predecesor numérico es aquel en el cual el término final es uno menos. Esto es, el número 3.1.4.5 es predecesor de 3.1.4.6.

En un diagrama de secuencia el orden visual determina la secuenciación y se muestra una sincronización por la presencia de múltiples mensajes que llegan a un mismo objeto antes de que el objeto envíe algún mensaje propio.

Expresión de secuencia: La expresión de secuencia es una lista de términos de secuencia separados por puntos y seguida por dos puntos (:). Cada término representa un nivel de anidamiento procedural dentro de la interacción global. Si es concurrente todo el control, entonces no se produce el anidamiento. Todos los términos de la secuencia poseen la siguiente sintaxis:

etiqueta-recurrencia _{opt}

En donde **etiqueta** es **entero** o bien **nombre**

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

El entero representa el orden secuencial del mensaje dentro del nivel inmediatamente superior de llamadas a procedimientos. Los mensajes que difieren en un término entero están relacionados secuencialmente en ese nivel de anidamiento. Por ejemplo, el mensaje 3.1.4 sigue al mensaje 3.1.3 dentro de la activación 3.1.

El nombre representa un hilo de control concurrente. Los mensajes que difieren en el nombre final son concurrentes en ese nivel de anidamiento. Por ejemplo, el mensaje 3.1a y el mensaje 3.1b son concurrentes dentro de la activación 3.1. Todos los hilos de control son iguales dentro de cada profundidad de anidamiento.

La recurrencia representa la ejecución condicional o iterativa. Esto representa cero o más mensajes que ejecutan, dependiendo de las condiciones. Las opciones son:

* [cláusula de iteración] una iteración

[cláusula de condición] bifurcación

Una iteración representa una secuencia de mensajes en la profundidad de anidamiento dada. La **cláusula de iteración** se puede omitir (en cuyo caso las condiciones de iteración no estarán especificadas). La **cláusula de iteración** debería expresarse en pseudocódigo o en algún lenguaje de programación real; UML no prescribe su formato. Un ejemplo podría ser, *[i:=1...n]

Una condición representa un mensaje cuya ejecución es contingente respecto a la veracidad de la cláusula de condición. La **cláusula de condición** debería expresarse en pseudocódigo o en algún lenguaje de programación real; UML no prescribe su formato. Un ejemplo podría ser, [x > y]

Si observamos las bifurcaciones se anotan igual que una iteración sin asterisco. Podemos pensar que se trata de una interacción limitada a un solo caso.

La notación de iteración supone que los mensajes de la iteración se ejecutarán secuencialmente. También existe la posibilidad de ejecutarlos concurrentemente. La notación empleada consiste en poner un doble línea vertical, indicadora de paralelismo, después del asterisco (*||).

Si observamos en una estructura de control anidada la recurrencia no se repite en niveles internos. En cada nivel de estructura se especifica su propia iteración dentro del contexto que la contiene.

Signatura: Una signatura es una cadena que indica el nombre, argumentos y valores proporcionado por una operación, mensaje o señal. Posee las propiedades siguientes:

lista-valores-proporcionados

Se trata de una lista de nombres separados mediante comas que denota los valores proporcionados por el mensaje en la ejecución subsiguiente de la interacción global. Si el mensaje no proporciona ningún valor, entonces se omite el valor proporcionado y el operador de asignación.

nombre-mensaje

El nombre del evento surgido en el objeto destino (suele ser el evento que solicita la realización de una operación). Puede implementarse de varias maneras, **una** de las cuales es una llamada a una operación. Si se implementa mediante una llamada a procedimientos, entonces se trata del nombre de la operación y esa operación tiene que estar definida en la clase del receptor, o debe ser heredada por él. En otros casos puede ser el nombre de un evento que surge en el objeto receptor. En la práctica habitual, con sobrecarga de procedimientos, para identificar a una operación se necesita tanto el nombre del mensaje como la lista de tipos de los argumentos.

lista-argumentos

Se trata de una lista de argumentos separados mediante comas, encerrada entre paréntesis. Se puede utilizar los paréntesis aun cuando la lista esté vacía. Cada argumento es una expresión en pseudocódigo o bien en un lenguaje de programación adecuado (UML no lo prescribe). La expresión puede utilizar los valores proporcionados por mensajes anteriores (del mismo alcance) y expresiones de navegación que partan del objeto original (esto es, atributos del mismo o enlaces que parten de él y rutas alcanzables desde ellos).

Ejemplos

Lo que sigue son ejemplos de la sintaxis de etiquetas de mensaje de control.

2: visualizar(x, y) Mensaje simple

1.3.1: p:=buscar(especificaciones) Llamada anidada que proporciona un valor

[x < 0] 4: invertir(x, color) Mensaje condicional

	INGENIERÍA EN INFORMÁTICA – PLAN 2003	
	DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

3.1 *: update() Iteración

A3, B4 / C2: copy(a, b) Sincronización de hilos

Diagrama de Clases de Diseño

Un Diagrama de Clases de Diseño muestra la especificación para las clases software de una aplicación. Incluye la siguiente información:

- (1) Clases, asociaciones y atributos.
- (2) Interfaces, con sus operaciones y constantes.
- (3) Métodos.
- (4) Navegabilidad.
- (5) Dependencias.

A diferencia del Modelo Conceptual, un Diagrama de Clases de Diseño muestra definiciones de entidades software más que conceptos del mundo real.

Relaciones de dependencia para representar visibilidad entre clases

Cuando una clase conoce a otra por un medio que no es a través de un atributo (una asociación con la navegabilidad adecuada), entonces es preciso indicar esta situación por medio de una dependencia.

Un objeto debe conocer a otro para poder llamar a uno de sus métodos, se dice entonces que el primer objeto tiene “visibilidad” sobre el segundo. La visibilidad más directa es por medio de atributo, cuando hay una asociación entre ambas clases y se puede navegar de la primera a la segunda (un atributo de la primera es un puntero a un objeto de la segunda). Hay otros tres tipos de visibilidad que hay que representar en el diagrama de clases mediante relaciones de dependencia:

- (1) **Parámetro:** Cuando a un método de una clase se le pasa como parámetro un objeto de otra clase, se dice que la primera tiene visibilidad de parámetro sobre la segunda. La relación de dependencia entre ambas clases se etiqueta con el estereotipo <<parámetro>> (<<parameter>> en inglés).
- (2) **Local:** Cuando en un método de una clase se define una variable local que es un objeto de otra clase, se dice que la primera tiene visibilidad local sobre la segunda. La relación de dependencia entre ambas clases se etiqueta con el estereotipo <<local>>.
- (3) **Global:** Cuando hay una variable global en el sistema, y un método de una clase llama a un método de esa variable global, se dice que la clase tiene visibilidad global sobre la clase a la que pertenece la instancia que es una variable global. La relación de dependencia entre ambas clases se etiqueta con el estereotipo <<global>>.

No es necesario representar la relación de dependencia entre clases que ya están relacionadas por medio de una asociación, que se trata de una “dependencia” más fuerte. Las relaciones de dependencia se incluyen tan solo para conocer qué elementos hay que revisar cuando se realiza un cambio en el diseño de un elemento del sistema.

Construcción de un Diagrama de Clases de Diseño

Para crear un Diagrama de Clases de Diseño se puede seguir la siguiente estrategia:

- (1) Identificar todas las clases participantes en la solución software. Esto se lleva a cabo analizando los Diagramas de Interacción.
- (2) Representarlas en un diagrama de clases.
- (3) Duplicar los atributos que aparezcan en los conceptos asociados del Modelo Conceptual.
- (4) Añadir los métodos, según aparecen en los Diagramas de Interacción.
- (5) Añadir información de tipo a los atributos y métodos.
- (6) Añadir las asociaciones necesarias para soportar la visibilidad de atributos requerida.
- (7) Añadir flechas de navegabilidad a las asociaciones para indicar la dirección de visibilidad de los atributos.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003 DISEÑO AVANZADO SOFTWARE – 10º CUATRIMESTRE	
	PROCESO DE DESARROLLO	VERSIÓN: 1.7 VIGENCIA: 19-09-2007

(8) Añadir relaciones de dependencia para indicar visibilidad no correspondiente a atributos.

No todas las clases que aparecían en el Modelo Conceptual tienen por qué aparecer en el Diagrama de Clases de Diseño. De hecho, tan solo se incluirán aquellas clases que tengan interés en cuanto a que se les ha asignado algún tipo de responsabilidad en el diseño de la interacción del sistema. No hay, por tanto, una transición directa entre el Modelo Conceptual y el Diagrama de Clases de Diseño, debido a que ambos se basan en enfoques completamente distintos: el primero en comprensión de un dominio, y el segundo en una solución software. En la fase de Diseño se añaden los detalles referentes al lenguaje de programación que se vaya a usar. Por ejemplo, los tipos de los atributos y parámetros se expresarán según la sintaxis del lenguaje de implementación escogido.

Navegabilidad

La navegabilidad es una propiedad de un rol (un extremo de una asociación) que indica que es posible “navegar” unidireccionalmente a través de la asociación, desde objetos de la clase origen a objetos de la clase destino. Se representa en UML mediante una flecha.

La navegabilidad implica visibilidad, normalmente visibilidad por medio de un atributo en la clase origen. En la implementación se traducirá en la clase origen como un atributo que sea una referencia a la clase destino.

Las asociaciones que aparezcan en el Diagrama de Clases deben cumplir una función, deben ser necesarias, si no es así deben eliminarse.

Las situaciones más comunes en las que parece que se necesita definir una asociación con navegabilidad de A a B son:

- (1) A envía un mensaje a B.
- (2) A crea una instancia B.
- (3) A necesita mantener una conexión con B.

Visibilidad de Atributos y Métodos

Los atributos y los métodos deben tener una visibilidad asignada, que puede ser:

- (1) Visibilidad pública.
- (2) Visibilidad protegida.
- (3) Visibilidad privada.

También puede ser necesario incluir valores por defecto, y todos los detalles ya cercanos a la implementación que sean necesarios para completar el Diagrama de Clases.

Fase de Implementación y Pruebas

Una vez se tiene completo el Diagrama de Clases de Diseño, se pasa a la implementación en el lenguaje de programación elegido.

El programa obtenido se depura y prueba, y ya se tiene una parte del sistema funcionando que se puede probar con los futuros usuarios, e incluso poner en producción si se ha planificado una instalación gradual.

Una vez se tiene una versión estable se pasa al siguiente ciclo de desarrollo para incrementar el sistema con los casos de uso asignados a tal ciclo.

BIBLIOGRAFÍA

- [Booch99] G. Booch, J. Rumbaugh, I. Jacobson (1999) “El Lenguaje Unificado de Modelado”. Addison Wesley Iberoamericana. España
[Larman99] C. Larman (1999). “UML y Patrones”. Prentice Hall. España