Una estrategia para alcanzar Correctitud y Completitud, minimizando la ambigüedad en una SRS escrita con UML

Juan Manuel Luzuriaga, jluzuria@uncoma.edu.ar
Universidad Nacional del Comahue – Facultad de Economía y Administración
Buenos Aires 1400 - Neuquén Capital (8300) - ✍ 4490312

Resumen

Es sabido que la correctitud, completitud y no ambigüedad son atributos de calidad mas que deseables en una SRS [Davis92] (Especificación de Requerimientos de Software). Aquí presento una propuesta de cómo escribir una SRS con UML maximizando dichos atributos, mediante una estrategia de utilización de diagramas UML que nos llevan de una manera natural a un producto con un alto grado de correctitud, completitud y no ambiguo.

1 - Introducción

Una casa bien diseñada es mucho más que un montón de paredes, puestas juntas para sostener en alto a un techo que proteja de las inclemencias del tiempo. Cuando una persona discute con un arquitecto el diseño de una casa para su familia, tiene muy en cuenta como se utilizara la casa. Este análisis es un ejemplo de análisis basado en casos de uso. Se consideran las diferentes formas en que se utilizara la casa, y estos casos de uso guiaran la arquitectura. Muchas familias tendrán las mismas categorías de casos de uso (las casas se utilizan para comer, dormir, criar a los niños y guardar recuerdos). Pero cada familia tendrá sus propios casos de uso, que podrían ser especiales o variaciones de los básicos. Las necesidades de una familia grande, por ejemplo, son diferentes a las de un adulto soltero que acaba de finalizar sus estudios. Estas variaciones son las que mayor impacto tienen en la forma que finalmente tendrá la casa.
El error en que se incurre habitualmente es considerar que la especificación de requerimientos esta lista en un gran porcentaje cuanto contamos con todos los casos de uso, luego explicaremos porque esto es un error.
En tal sentido nuestra propuesta se basa en la estrategia de cómo combinar los diagramas UML de Casos de Uso, Secuencia, Clases, Estados y Actividades para restarle ambigüedad a la especificación aumentando correctitud y completitud. Para ello veremos en primer lugar que nuestra estrategia, al igual que el Proceso Unificado Rational [Jac98], postula una arquitectura céntrica en los casos de uso, y a partir de ellos iremos logrando completitud y restando ambigüedad a medida que construimos el resto de los diagramas. Pero para que esto ocurra debemos definir lo mas acertadamente posible (maximizar correctitud) los casos de uso, ya que son la fuente principal de información para construir el los diagramas subsiguientes, para que esta definición sea altamente correcta se deberán considerar los factores que mencionaremos mas adelante

2.1 - Por que no utilizar solamente casos de uso

- Esta orientado a describir solamente la secuencia de eventos que resultan del uso del sistema por parte de un actor, quedando en una zona gris el conjunto de eventos que engloban las "actividades de custodia" [Men84]
- Existe una tendencia natural a confundir los actores que activan al sistema con los operadores del mismo. Ej: un operador "data entry" nunca será un actor, sin embargo pareciera ser el iniciador de la entrada de datos (salvo que estemos describiendo interfaces)
- Si bien detecta a los actores que activan el sistema, no identifica claramente la situación por la cual el sistema reacciona, sino que lo hace a través de quien efectúo el estimulo.
- Como el lenguaje natural es ambiguo por naturaleza, se corre el riesgo de que la descripción del caso de uso tenga mas de una interpretación (ambigüedad)

Para resolver las cuestiones planteadas precedentemente debemos complementar los casos de uso con diagramas de: interacción , actividades, estados y clases

3 - Estrategia propuesta

- Introducción con una descripción del dominio específico escrita en lenguaje natural, de manera tal que le permita a los primeros lectores de la SRS saber de que se trata el sistema.
- Diagramas de Caso de uso , a partir de los cuales por cada caso de uso debe documentarse una descripción en lenguaje natural, utilizando el léxico del dominio específico, conteniendo la información propuesta en el siguiente Template [Cock01]:

CASO DE USO #	< Nombre corto representativo del acierto de realizar este caso de uso>	
Evento	<Nombre del evento que dispara el caso de uso>	
Acierto	<Breve descripción de lo que se lograría ejecutando este caso de uso>	
Alcance y nivel	<Alcance del caso de uso (quien y que)> <Nivel del caso de uso, ej: Síntesis, Tarea primaria, Subfuncion>	
Precondición	<Cual es el entorno esperado para poder comenzar con el caso de uso>	
Condicion de finalización exitosa	<El estado en que queda el contexto luego de la finalización exitosa del caso de uso>	
Condicion de finalización con falla	<El estado en que queda el contexto si el caso de uso falla>	
Actores primarios y secundarios	<Nombre y descripción del rol de los actores primarios y secundarios>.	
Flujo de entrada	Información proporcionada por el actor	
Flujo de salida	Información recibida por el actor	
Trigger	<Acción que se ejecuta al comienzo de l caso de uso>	
Descripción	**Paso**	**Acción**
	1	<Descripción de cada paso en castellano estructurado, según ocurra en el caso de uso>
	2	<...>
	3	<...>
Extensiones	**Paso**	**Acciones de bifurcación**
	1a	<Condición que causa la bifurcación> : <Acción o nombre del sub caso de uso>

Información Relacionada	<Nombre del caso de uso>
Prioridad:	<Cuan critica es para el sistema o la organización>
Performance	<La cantidad de tiempo que la ejecución del caso de uso debiera tomar>
Frecuencia	<Con que frecuencia se espera que ocurra>
Canales de comunicación con los actores	<Tipo de canales de comunicación con los actores, ej: interactivo, Archivos estáticos, Bases de datos, timeouts>
Cuestiones abiertas	<Lista de cuestiones esperando decisión, que afectan este caso de uso >
Observaciones...	Lo que se necesite aclarar o documentar>

Superordinados	<Opcional, Nombre de los casos de uso que utilizan a éste >
Subordinados	<Opcional, Nombre de los casos de uso que son utilizados en éste >

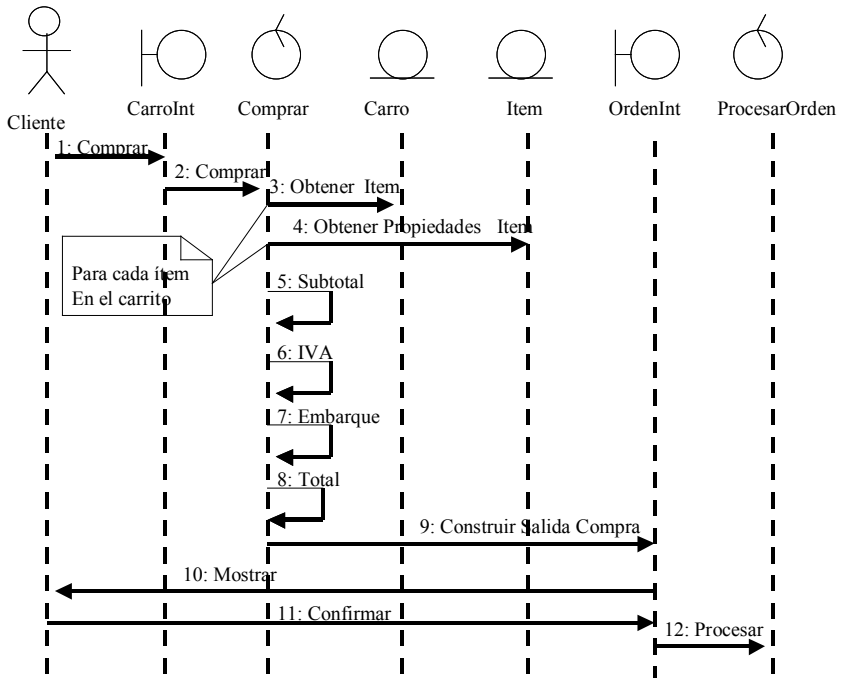
- Por cada entrada "descripción" en la documentación de cada caso de uso realizar una análisis sintáctico de sujeto y predicados, donde el/los sustantivos del sujeto son clases candidatas del mundo real, y el/los verbos del predicado representan las responsabilidades que se traducen en acciones o actividades que realiza el objeto en cuestión. A partir de allí se construye un diagrama de secuencias que muestre la secuencia temporal de eventos que se producen en la interacción de los objetos cuando cada uno cumple con sus responsabilidades. Para ilustrar dicho proceso , consideremos la siguiente descripción del caso de un uso definido para el dominio de e-comerce

CASO DE USO 1	Comprar	
Evento	Selección opción de compra	
Acierto	Comprar los productos seleccionados	
Alcance y nivel	Clientes registrados con productos seleccionados en el carrito de compras, Nivel 1	
Precondición	El cliente debe estar registrado , debe haber iniciado la compara y productos en el carrito	
Condición de finalización exitosa	El cliente compró los productos elegidos en el carrito de compras	
Condición de finalización con falla	El cliente no esta registrado, no inició la compra o no tiene productos seleccionados en el carrito	
Actores primarios y secundarios	Cliente	
Flujo de entrada	A definir en etapa posterior	
Flujo de salida	A definir en etapa posterior	
Trigger	Ninguno	
Descripción	**Paso**	**Acción**
	1	El cliente le dice al sistema que quiere comprar
	2	El sistema examina el contenido del carrito de compras y produce una lista de los ítem que están listos para ser adquiridos calculando según sus propiedades el subtotal, iva, embarque y el total
	3	El cliente confirma la orden de compra y le pide al sistema que la procese

Produciendo el análisis sintáctico obtenemos como clases candidatas a
Cliente , sistema, carrito de compras, lista de ítems, orden de compras
Y como responsabilidades surgen:

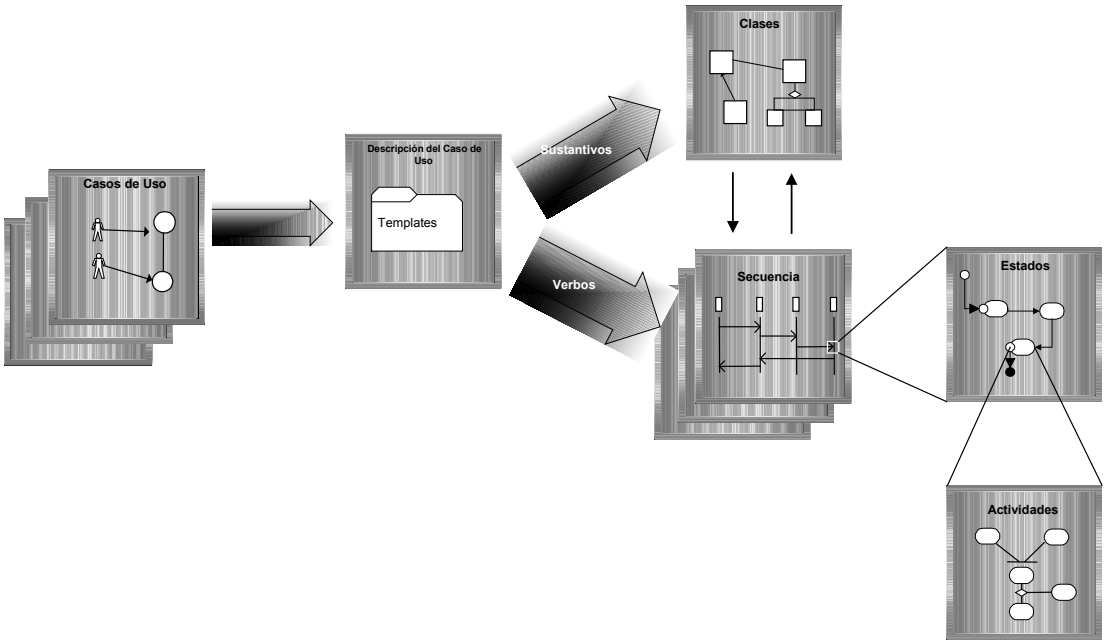
Cliente: decir, confirmar, pedir

Sistema: examinar, producir lista de ítems, procesar orden de compra



- Por cada diagrama de interacción, evaluar cada línea de vida de los objetos identificando los distintos estados y transiciones a partir de la llegada o salida de un mensaje. Esto nos dará por resultado las actividades (futuros métodos) que deberá realizar cada objeto.
- Aquellas actividades que sean complejas en cuanto a su lógica, deberemos describirlas utilizando diagramas de Actividades cuando existan alternativas y/o tareas que deban hacerse en forma concurrente o paralela.
- A partir de la información obtenida en estos diagramas, construimos un diagrama de clases que refleje la estructura que soportará la funcionalidad especificada en los diagramas anteriores.

Podemos visualizar gráficamente la estrategia a partir de la siguiente figura:



7 – Conclusiones

Como es sabido, el UML por ser un lenguaje, no nos proporciona la metodología de utilización de los mismos, ni la forma de combinarlos para escribir una SRS, y en cierto aspecto el RUP tampoco prescribe explícitamente la forma de construir dichos diagramas. En tal sentido, confío en que la estrategia presentada pueda ser de gran utilidad para quien se inicie en el uso del UML para la especificación de requerimientos, que usualmente se encuentra abrumado por la cantidad de símbolos y diagramas tan bien definidos por "los tres amigos" (Booch, Rumbghaut, Jacobson), sin saber como combinarlos en una SRS.

Referencias

[Cock01] Write effective uses cases ,Alistair Cockburn, Addison Wesley, 2001
[Davis92] Software Requirements, Prentice Hall, 1992, Alan Davis
[Men84] Essential Systems Analysis , MC Menamin – Palmer , Yourdon Press,1984
[Jac98] The Unified Software Development Process, Jacobson I., Booch G., Rumbaugh J., Addison Wesley, 1998