Transformación de Especificación de Requisitos en Esquemas Conceptuales usando Diagramas de Interacción

Emilio Insfrán[1], Elena Tejadillos[2], Sofía Martí[2], Margarita Burbano[1]

[1]Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia. 46022 Valencia, España.
{einsfran|mburbano}@dsic.upv.es

[2]CARE Technologies
Polígono Industrial Juyarco, Partida Madrigueres 44. 03700 Denia, España.
{etejadillos|smarti}@care-t.com

Resumen. En el ámbito de la Ingeniería del Software, diversos métodos han sido propuestos para la especificación y modelado de requisitos y el modelado conceptual. En este trabajo se presenta un Proceso de Análisis de Requisitos (PAR) que permite refinar una Especificación de Requisitos representada mediante Casos de Uso utilizando Diagramas de Secuencia extendidos con estereotipos de UML. Se define además, un conjunto de reglas que facilita la trazabilidad de los elementos de la Especificación de Requisitos a una representación equivalente en el Esquema Conceptual.

1 Introducción

El tiempo es un factor esencial a la hora de desarrollar un proyecto. A menudo, el tiempo invertido en realizar una especificación detallada de requisitos se considera tiempo perdido si no se obtiene una ventaja tangible como resultado de dicha especificación. Por este motivo, considerando el ciclo de vida de desarrollo de software, las primeras fases (toma de datos y análisis del sistema) formalmente representan tiempos muy pequeños comparados con las fases de diseño, programación, pruebas y mantenimiento. En este artículo, se introduce un conjunto de técnicas para la especificación de los requisitos funcionales y además un proceso que sistemáticamente descompone estos requerimientos de usuario en especificaciones más detalladas que constituyen el esquema conceptual del sistema deseado.

Dos de las contribuciones más importantes de este trabajo son:

- **La mejora de la predictibilidad**, en el sentido de tener un esquema conceptual que es una representación precisa de los requisitos del usuario.
- **El incremento de la productividad**, ya que la tarea de Especificación de Requisitos funcionales tiene una trazabilidad directa con el esquema conceptual. Esto eli-

mina considerablemente el *gap* existente entre ambos permitiendo poner mayor esfuerzo en la especificación y validación de requisitos y además, mantener la consistencia entre los requisitos funcionales y el esquema conceptual.

En particular, nuestro trabajo define un *Modelo de Requisitos* [1], [2], el cual captura aspectos funcionales y de interacción actor/sistema. Esto se organiza mediante el uso de tres técnicas complementarias: *Misión del Sistema* (MI), *Árbol de Refinamiento Funcional(ARF)* y *Modelo de Casos de Uso(MCU)*, los cuales serán explicados con mayor detalle en las siguientes secciones.

Cuando se trabaja con sistemas complejos, resulta difícil obtener en un único paso un Esquema Conceptual, que represente una solución al problema, a partir de la descripción de los requisitos de usuario [3]. Por esta razón, se ha introducido un nuevo nivel denominado *Proceso de Análisis de Requisitos (PAR)* a fin de introducir mayor nivel de detalle e interpretar la información contenida en el Modelo de Requisitos estableciendo mecanismos directos de trazabilidad de forma sistemática. En la Figura 1 se muestra las relaciones entre el modelo de Requisitos y el Modelo Conceptual a través del PAR.

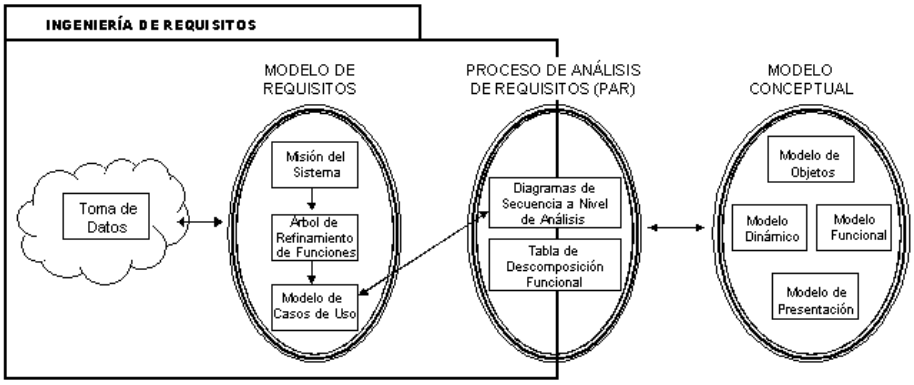


Fig. 1. Relaciones entre Modelos

La trazabilidad de requisitos ha sido identificado en la literatura como un factor de calidad (una característica que el proceso debería tener y es considerada como un requisito no-funcional) [4]. Por ejemplo, los estándares de desarrollo de sistemas del gobierno de EEUU (MIL-STD-2167-A y MIL-STD-498 [5]) requieren el desarrollo de documentos de trazabilidad de requisitos.

En [6] la trazabilidad está definida como una técnica usada para "proveer una relación entre requisitos, el diseño y la implementación final del sistema". En [7] se establece que "la trazabilidad da asistencia esencial en el entendimiento de las relaciones que existen entre y a través los requisitos, el diseño y la implementación". Estas relaciones permiten a los diseñadores mostrar que el diseño cumple con los requisitos

funcionales y ayudan al reconocimiento temprano de aquellos requisitos que no son satisfechos por el diseño

Desde nuestro punto de vista, la trazabilidad entre modelos presentada en este trabajo, conduce a la reducción de los efectos negativos de dos de los problemas más comunes que actualmente existen en los métodos de desarrollo del software:

- los ingenieros del software no saben si un Esquema Conceptual representa los requisitos funcionales definidos por el usuario. Estos requisitos están normalmente representados de forma no estructurada o semi-estructurada, con una difícil trazabilidad hacia los niveles de abstracción más detallados.
- cuando se llega a la fase de programación, el valor del esfuerzo realizado en la especificación de requisitos y en el Modelado Conceptual no están del todo claros. Esto es principalmente debido a que no es posible producir un código fuente que funcione de forma equivalente al esquema conceptual desarrollado.

En nuestra propuesta, el entorno de desarrollo es el proporcionado por OO-Method [8] [9] [10]. De esta forma se cubre todo el ciclo de vida de desarrollo de software. El Proceso de Análisis de Requisitos permite refinar la especificación de requisitos (Casos de Uso) y define un conjunto de reglas que asegura que cada elemento del Modelo de Requisitos tendrá una representación equivalente en el Esquema Conceptual. En el marco de OO-Method, este Esquema Conceptual será representado en la plataforma de implementación correspondiente siguiendo el paradigma de Generación de Código Basado en Modelo [11] como se presenta en [9] [12].

Este trabajo se encuentra organizado de la siguiente forma: Tras esta introducción, en la sección 2 se presenta el Modelo de Requisitos en el que se presenta cada una de las técnicas utilizadas para la representación de los requisitos funcionales de usuario. En la sección 3, se presenta el Proceso de Análisis de Requisitos (PAR) con las técnicas y las guías definidas para la obtención del Esquema Conceptual. En la sección 4 se explica un ejemplo de la propuesta y finalmente, en la sección 5, las conclusiones.

2 Modelo de Requisitos

El Modelo de Requisitos hace uso para su construcción de tres técnicas complementarias entre sí que capturan fielmente las características del sistema que se desea construir. Estas son: la *Misión del Sistema* (MS), el *Árbol de Refinamiento Funcional* (ARF) y el *Modelo de Casos de Uso* (MCU).

El sistema plantea un objetivo principal que debe ser satisfecho. Las interacciones entre el entorno y el sistema deben dar respuesta a este propósito. Para cumplir el objetivo principal del sistema se organiza un *Árbol de Refinamiento de Funciones* (ARF) en el que se determina el propósito o misión del sistema (la raíz del árbol) y se identifican todas las funciones (cada una de las interacciones externas) del sistema. Esta identificación de funciones se realiza mediante un proceso iterativo de depura-

ción de las áreas funcionales (nivel 1 del ARF) hasta llegar a las funciones que son las unidades básicas de interacción con los usuarios/actores del sistema[1]. El ARF es el punto de entrada para la construcción del Modelo de Casos de Uso.

En el siguiente nivel de abstracción, cada función elemental del ARF (nodo hoja) se representa por tanto como un caso de uso. Para la construcción del MCU se definen los actores (usuarios) que participan en su realización. La especificación de un caso de uso describirá la secuencia de acciones que el sistema debe ejecutar para alcanzar cada uno de los requisitos planteados. Esta secuencia ordenada de acciones se estructuran en términos de *pasos* de forma que el conjunto de pasos expresado de manera coherente y secuencial constituye la totalidad de la especificación de forma similar a como se propone en [13] [14] y [15].

3 Proceso de Análisis de Requisitos (PAR)

Una vez obtenido el Modelo de Requisitos, se define una nueva fase en la que los escenarios descritos en la especificación de casos de uso son refinados y expresados en términos de responsabilidades. Estas responsabilidades son descritas como clases que cooperan entre sí para la obtención de las funcionalidades especificadas en los casos de uso. Para expresar las responsabilidades a este nivel de detalle y capturar los elementos que posteriormente se convertirán en elementos del Esquema Conceptual, se utiliza un tipo de diagrama de interacción: el *Diagrama de Secuencia (DS)*.

En la literatura se encuentra información acerca de la notación y semántica a utilizar en este tipo de diagramas de interacción, pero se hace muy poco énfasis en los métodos de construcción y de extracción de información[2] para otros modelos utilizados en el proceso de desarrollo del software. En UML [16] se recomienda el uso del DS para modelar algún aspecto del comportamiento del sistema y puede ser usado en el contexto de todo el sistema, un subsistema o para un caso de uso. Algunos autores [17], [18] proponen que al menos un DS debe construirse por caso de uso. Otros indican que esto no es necesario ni práctico y que deberían construirse sólo cuando la complejidad del comportamiento del caso de uso lo requiera [19]. El aporte principal de este trabajo consiste en proponer una guía para la transformación de una la Especificación de Requisitos en un Esquema Conceptual utilizando los DS.

3.1 El Diagrama de Secuencia

El Diagrama de Secuencia muestra las clases que participan para la realización de un caso de uso siguiendo la notación del Lenguaje de Modelado Unificado (UML) [16]. Cada clase participante etiqueta una línea temporal. La comunicación entre las clases participantes para cada responsabilidad individual se muestra utilizando mensa-

[1] Este concepto de función coincide con la definición de Caso de Uso de [17] [14].
[2] Trazabilidad entre la Especificación de Casos de Uso y el Modelo Conceptual.

jes. Cada mensaje está compuesto por el nombre de un servicio y sus argumentos. Adicionalmente cada mensaje puede ser etiquetado de acuerdo con su naturaleza definiendo los estereotipos que definimos a continuación:

Mensaje Signal: representa una interacción entre el actor y la interfaz del sistema. Puede consistir en introducción de datos, selección de opciones o para mostrar resultados. Este tipo de elementos es utilizado para el diseño del interfaz de usuario.

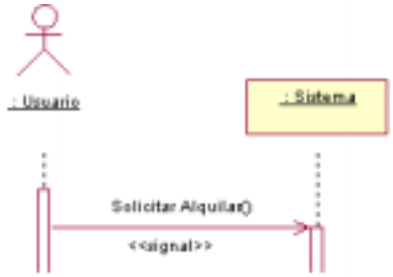


Fig. 2. Ejemplo de mensaje Signal

Mensaje Service: representa la modificación del estado de las instancias de la clase receptora del mensaje. Pueden crear nuevas instancias (new), borrar (destroy) o modificar el estado de un objeto existente (update).

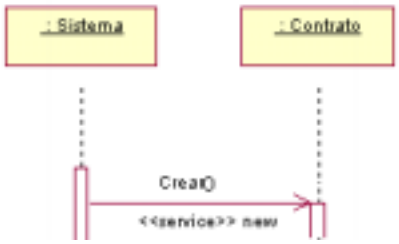


Fig. 3. Ejemplo de mensaje Service

Mensaje Query: representa consultas sobre el estado de objetos. La clase emisora puede ser cualquier clase del sistema o incluso la clase sistema (en este caso el resultado de la consulta puede ser utilizado para la construcción del interfaz de usuario).

Fig. 4. Ejemplo de mensaje Query

Mensaje Select: se utiliza para representar que un objeto realiza una interacción con otro objeto para establecer/eliminar una relación estructural entre ellos. Habitualmente se utiliza cuando un objeto se crea y necesita estructuralmente relacionarse con otros para su existencia (aunque podría ocurrir en cualquier momento de su vida).

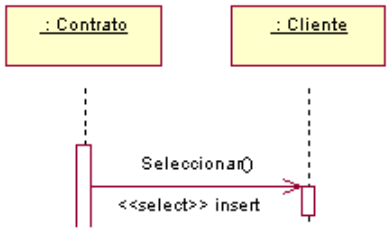


Fig. 5. Ejemplo de mensaje Select

Además, siguiendo la notación UML se pueden describir las interacciones habituales entre clases de objetos como interacciones condicionales, bucles, etc.

De esta forma, basado en la especificación textual del caso de uso se define uno o varios[3] Diagramas de Secuencia que describan su comportamiento añadiendo los estereotipos definidos a los mensajes correspondientes. De esta forma, el caso de uso puede ser revisado basado en los resultados del DS y viceversa, hasta que los dos modelos queden *sintonizados* apropiadamente. En la sección 4 se muestra un ejemplo de una especificación detallada de Diagrama de Secuencia.

3.2 Guías para la construcción del Esquema Conceptual

Este punto ofrece una guía que permite establecer la trazabilidad entre los elementos del Diagrama de Secuencia y el Esquema Conceptual. El proceso de construcción del Esquema Conceptual se basa en el análisis de los distintos Diagramas de Secuen-

[3] Se utiliza un Diagrama de Secuencia para cada caso de uso primario, y otro para cada caso de uso que utilizado en las relaciones *include* y *extend* del modelo de casos de uso.

cia que se obtienen a partir de la especificación de los casos de uso. Básicamente los elementos que serán obtenidos son los siguientes:

- Clases
- Atributos
- Servicios: transacciones y eventos
- Relaciones de asociación entre clases
  - Cardinalidades
- Relaciones de herencia entre clases

Obtención de clases.

Las clases que forman el Esquema Conceptual se obtienen a partir de las clases que aparecen en los Diagramas de Secuencia a excepción de la clase *system* que representa la frontera entre el sistema en desarrollo y el entorno, por tanto, no se explicita en el Esquema Conceptual.

Obtención de atributos.

Los atributos de las clases se obtienen de los argumentos de los mensajes intercambiados entre las clases de objetos. Así, de forma genérica, podemos decir que si entre dos clases se intercambia un mensaje de tipo servicio en el cual se hace referencia a un argumento, este argumento será representado como atributo de la clase destino del mensaje, mientras que si se trata de un valor de retorno en un mensaje de tipo consulta, se representará como un atributo derivado en la clase emisora del mensaje. Adicionalmente, se crea un atributo identificador para cada una de las clases.

Obtención de servicios.

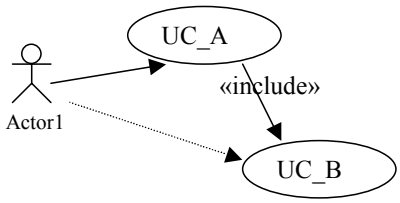
Como eventos de las clases vamos a tener aquellos que aparecen en los mensajes del tipo <<service>>. Así, si entre dos clases se intercambia uno de estos mensajes, la clase destinataria del mensaje es la que va a tener el servicio que se especifica. Además, si el estereotipo tiene la propiedad *new* o *destroy* el evento será de creación o destrucción respectivamente. Todos los eventos definidos de esta forma tendrán un carácter privado, es decir, no podrán ser activados explícitamente por actores del sistema.

Una *transacción* representará una unidad de ejecución y contendrá todos los eventos obtenidos a partir de los mensajes del tipo <<service>> que se intercambien en el Diagrama de Secuencia. De esta forma, un caso de uso se representa por un Diagrama de Secuencia y todo Diagrama de Secuencia será representado como una transacción, por lo que el caso de uso será representado por la transacción correspondiente. Los actores del caso de uso utilizados en el Diagrama de Secuencia serán los que tendrán visibilidad de ejecución sobre esta transacción. La transacción estará definida en la clase destino del primer mensaje del tipo <<service>> emitido por la clase *system*.

En cuanto a las relaciones *include* y *extend*, precisan de un tratamiento distinto:

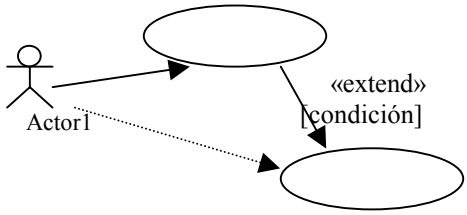
- **Relación include**

El caso de uso UC_A, generará la transacción TRUC_A que será pública. El caso de uso UC_B generará la transacción TRUC_B en cuanto a su carácter, este va a depender de si el actor puede realizar una llamada a dicho caso de uso o no. Si la puede realizar será pública y en caso contrario será privada. Por tanto la primera transacción TRUC_A estará formada por los servicios identificados en su diagrama de secuencia, más la llamada a la transacción TRUC_B. Por otro lado tendremos que la transacción TRUC_B tendrá sólo los servicios identificados en su diagrama de secuencia.



- **Relación extend**

El caso de uso UC_A, generará la transacción TRUC_A y el caso de uso UC_B generará la transacción TRUC_B. En la transacción TRUC_B se debe verificar que se cumpla la condición asociada a la relación extend y en este caso deberá invocar a la transacción TRUC_A[4].



Obtención de relaciones de Asociación entre clases.

La clave para establecer una relación de asociación entre dos clases cualesquiera está en los mensajes que son intercambiados entre éstas en los distintos diagramas de secuencia de toda la especificación de requisitos. Así, de forma genérica, decimos que una clase A está en relación de asociación con una clase B (distintas de la clase *system*), cuando existe algún diagrama de secuencia en el que aparecen ambas clases y entre ellas existe un mensaje del tipo:

– **<<query>>** un objeto necesita conocer información sobre el contenido de otro objeto,

[4] Esta invocación es de tipo operacional, es decir, no cumple la propiedad "todo-nada" indicada en la definición de transacción de OO-Method.

- **<<select>>** un objeto necesita establecer o eliminar una relación estructural con otro objeto de la misma u otra clase. Este es un tipo muy importante de interacción entre objetos no solamente van a intercambiar información sino que tambien sus vidas estarán sincronizadas de acuerdo a la definición de asociación, agregación o composición resultante,
- **<<service/new>>** un objeto crea otro objeto instancia de la misma u otra clase del sistema.

Cardinalidades. Dependiendo del origen y destino de los mensajes así como de los objetos participantes en las interacciones se pueden determinar las cardinalidades respectivas. Por ejemplo, si en todo el sistema sólo existe un mensaje de la clase A a la clase B del tipo <<select>> con propiedades mínimo 1 y máximo 1, quiere decir que un objeto de la clase A necesita relacionarse como mínimo y como máximo con un objeto instancia de la clase B.

Si el mensaje se encuentra dentro de un bloque condicional, la cardinalidad mínima es siempre cero, ya que es posible que no se cumpla la condición del bloque.

Relaciones de Herencia.

En el Diagrama de casos de uso se representan explícitamente las relaciones de herencia entre actores. Si estos actores participan en los Diagramas de Secuencia asociados a los casos de uso, los actores se representarán como clases y las relaciones de herencia se convertirán en relaciones de herencia entre dichas clases. Las demás relaciones de herencia entre las clases del dominio del negocio se definen explícitamente entre las clases ya identificadas en el Diagrama de Clases del Esquema Conceptual, manteniéndose la trazabilidad con la clase origen.

4 Del Diagrama de Secuencia al Esquema Conceptual

A continuación se describe un ejemplo de análisis de un Diagrama de Secuencia para obtener una parte del Modelo de Objetos. Es importante hacer notar que esto forma parte del Proceso de Análisis de Requisitos y establece una trazabilidad explícita desde el Modelo de Requisitos al Modelo Conceptual.

Del análisis de todos los Diagramas de Secuencia se obtienen las clases pertenecientes al Esquema Conceptual. En el ejemplo de la Fig. 6 se muestra el Diagrama de Secuencia correspondiente un Caso de Uso de venta de artículos, que permite al cajero realizar una venta que contiene un detalle de cada artículo vendido, así como la cantidad.

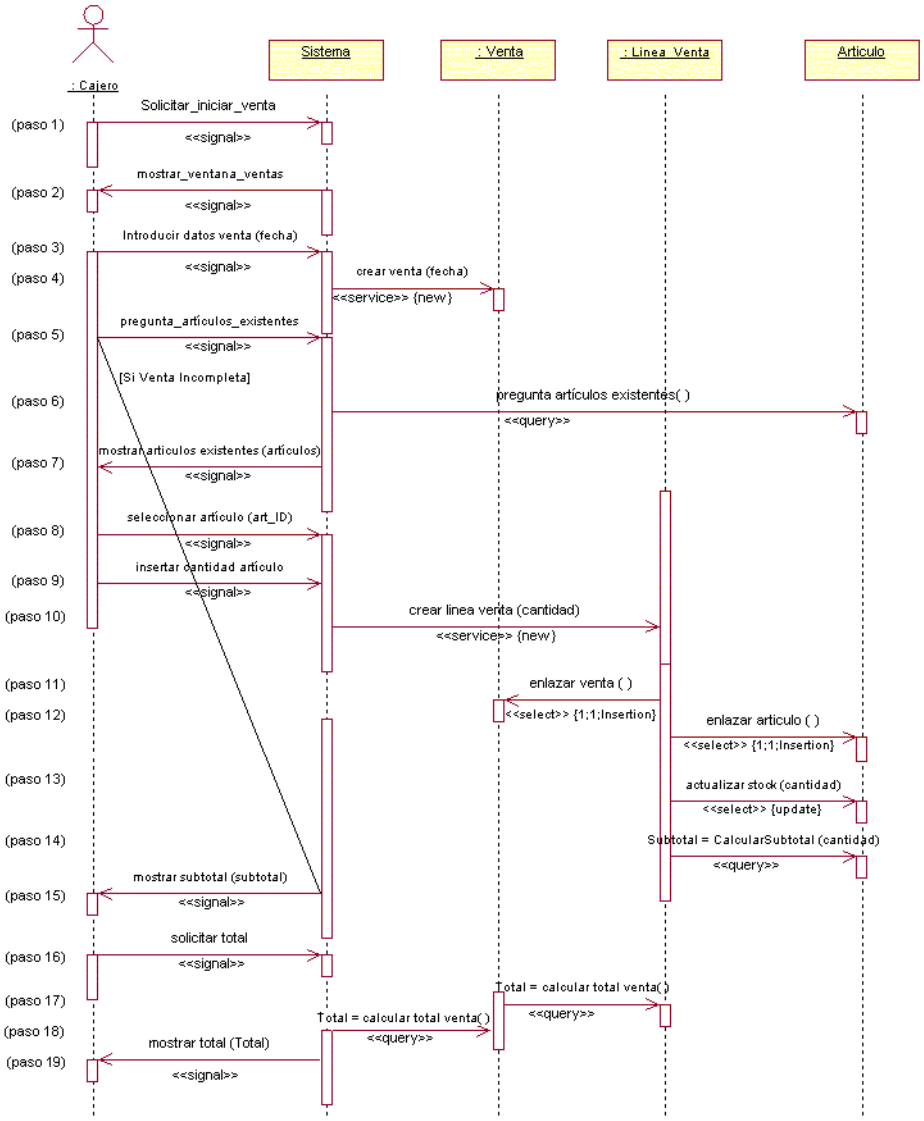


Fig. 6. Diagrama de Secuencia completo para la Venta de Artículo

Para que sea más sencillo seguir el proceso de obtención del Modelo de Objetos a partir del DS, los mensajes con estereotipos no relevantes a este proceso no son considerados. En particular los mensajes con estereotipo <<*signal*>> que especifica la funcionalidad de la interfaz y los mensajes de tipo <<*query*>> que son enviados por la clase sistema para realizar consultas sobre población de clases. Estos mensajes se utilizarán para obtener otros modelos como es el Funcional y el de Presentación (que no se contemplan en este artículo).

El ejemplo simplificado obtenido a partir de la Fig. 6 se puede observar en la Fig. 7:

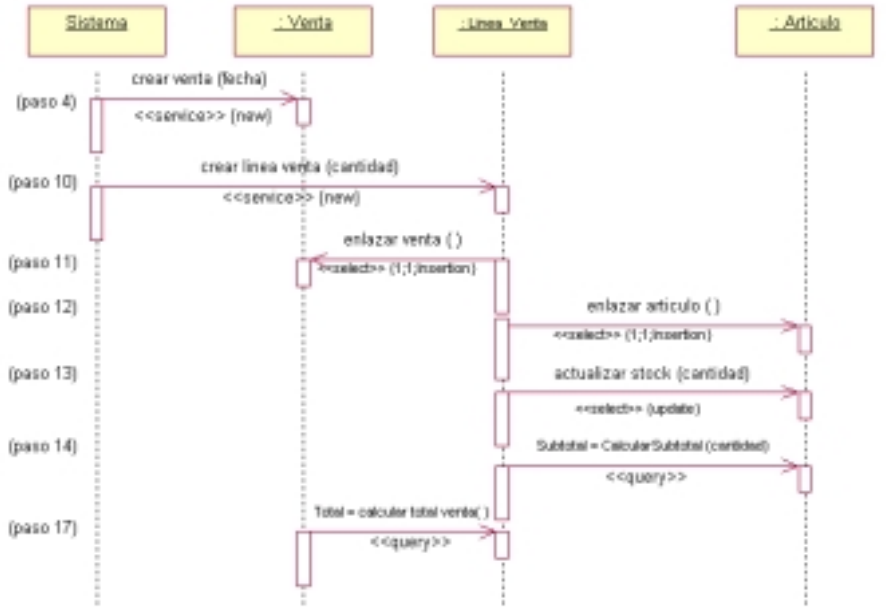


Fig. 7. Diagrama de Secuencia parcial para la Venta de Artículo

A continuación, se aplicará la guía de construcción del Esquema Conceptual para el DS descrito:

Obtención de clases.

Analizando el DS, obtenemos tres clases: *Venta*, *LineaVenta* y *Artículo* que corresponden con las clases identificadas en el DS. La clase Sistema no se añade, ya que se utiliza como clase auxiliar para la especificación del DS.

Obtención de atributos.

Para la clase *Venta*, tendremos los siguientes atributos:

- *Venta_ID*, atributo identificador de la clase.
- *Fecha*, que es el argumento del mensaje *crear_venta* (paso 4).
- *Total*, que es el valor devuelto de la consulta *Calcular_Total* (paso 17)

Para la clase *LineaVenta*, tendremos:

- *LineaVenta_ID*, atributo identificador de la clase. Siempre aparece.
- *Cantidad*, procedente del mensaje *crear_linea_venta*(paso 10)
- *Subtotal*, obtenido del valor de retorno de la consulta *CalcularSubtotal* (paso 14).

Para la clase *Articulo*, obtendremos los siguientes atributos:

- *Articulo_ID*, atributo identificador de la clase.

- *Cantidad*, procedente del mensaje *Actualizar_Stock* (paso 13).

Obtención de servicios.
Para la clase *Venta*:
- *crear_venta()*, evento de creación privado (paso 4)
- *CREAR_VENTA_ARTICULOS()*, transacción de creación que el agente Cajero, podrá iniciar.

Para la clase *LineaVenta*:
- *crear_linea_venta()*, evento de creación privado (paso 10)

Para la clase *Articulo*:
- *actualizar_stock()*, evento privado.

Obtención de relaciones de asociación entre clases.
- Relación de asociación entre *Venta* y *LineaVenta*, deducida del mensaje de selección *enlazar_venta* (paso 11).
  - Cardinalidades.
    - La *LíneaVenta* necesita una venta como mínimo y como máximo. Esto se obtiene analizando las cardinalidades de este evento de selección *enlazar_venta* y debido a que no hay ninguna cardinalidad en otro caso más relajada.
    - En cuanto a la venta debido a que la interacción con líneas de venta, se encuentra en una iteración (donde no se indica el máximo ni mínimo de iteraciones), tiene las cardinalidades mínima a 0 y máxima a M.
- Relación de asociación entre *LineaVenta* y *Articulo*, deducida del mensaje de selección *enlazar_articulo* (paso 12)
  - Cardinalidades:
    - La línea de venta necesita un artículo como mínimo y como máximo. Esto se obtiene analizando las cardinalidades de este evento de selección *enlazar_articulo* y debido a que no hay ninguna cardinalidad en otro caso más relajada.
    - En cuanto al artículo debido a que la interacción con líneas de venta no define cardinalidades tiene como mínima 0, y como máxima M.

El resultado de este análisis es resumido en la Tabla de Descomposición Funcional (CRUD[5]). En la tabla se especifica qué acción (creación, lectura, actualización y borrado) se realiza sobre una clase dentro de cada caso de uso. Se puede obtener de forma automática recorriendo todos los DS del sistema. De esta forma se reúne la información de forma más compacta.

[5] La tabla CRUD es la abreviatura en inglés de C-Creation R-Read, U-Update and D-Delete.

Esta tabla es única para todo el sistema y se puede ver claramente qué funcionalidades afectan a cada clase ya que buscar dicha información en los DS puede resultar muy difícil (el sistema tendrá al menos tantos DS como casos de uso se definan).

Para el DS analizando, se obtiene la siguiente tabla CRUD:

	Caso de Uso: Ventas	...
Venta	**C** *(crear_venta)*	
LineaVenta	**C** *(crear_linea_venta)*	
Articulo	**R** *(calcular_subtotal)* **U** *(actualizar_stock)*	
...		

Si se lee verticalmente, se observan todos los servicios comprendidos en un caso de uso. Mientras que si se lee horizontalmente, se observan los posibles servicios pertenecientes a una clase[6].

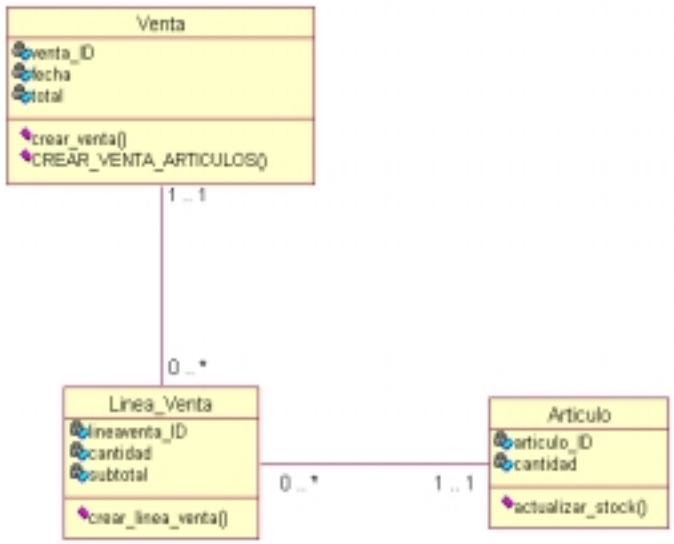


Fig. 8. Esquema Conceptual parcial obtenido a partir del Caso de Uso Venta de Artículos

Finalmente, en la Figura 8 se presebnta el Esquema Conceptual obtenido a partir del DS analizado. El Esquema Conceptual completo, será obtenido una vez analizados todos los Diagramas de Secuencia correspondiente a los Casos de Uso del Modelo de Requisitos. De esta forma se garantiza la trazabilidad del Modelo de Requisitos al

[6] En la tabla, entre paréntesis y en cursiva se encuentra el nombre del mensaje al que se hace referencia en el DS.

Esquema Conceptual y que, en el entorno de generación automática de código provisto por OLIVA NOVA Model Execution® se permite obtener una aplicación funcionalmente equivalente a la información especificada en el Modelo de Requisitos.

5 Conclusiones

En este trabajo se ha presentado una extensión a OO-Method para cubrir de forma precisa la fase de Ingeniería de Requisitos permitiendo una transición guiada hacia el Esquema Conceptual. Entender *qué* requiere el usuario y disponer de mecanismos que permitan su especificación de forma sencilla y a la vez que facilite su integración con las demás fases del proceso de desarrollo de software es un punto clave en la Ingeniería del Software. Además, trabajando en un contexto de generación de código se hace posible disponer de elementos de trazabilidad concretos que llevan desde los requisitos hasta el código.

Esta aproximación ha sido utilizado con éxito en algunos proyectos piloto de tamaño medio en el contexto de un proyecto de I+D entre la Universidad Politécnica de Valencia y CARE Technologies S.A. Actualmente estamos trabajando en el desarrollo de una herramienta CASE para la captura de los elementos del Modelo de Requisitos y la conversión semi-automática a Esquemas Conceptuales como automatización del Proceso de Análisis de Requisitos presentado en este trabajo. Esta herramienta utilizará XML como mecanismo de comunicación con la herramienta CASE de OO-Method (OLIVA NOVA Model Execution®). En una primera versión, esta herramienta sólo tendrá facilidades para exportar el Esquema Conceptual obtenido del PAR a OLIVA NOVA Model Execution®. Para una segunda versión se prevé la comunicación bidireccional con la intención de soportar el ciclo de vida en espiral y donde ciertas modificaciones realizadas en el esquema Conceptual se vean directamente reflejadas en la especificación de requisitos a distintos niveles de abstracción (Diagramas de Secuencia, Casos de Uso o Árbol de Refinamiento de Funciones). De esta forma, se establecen los mecanismos de trazabilidad entre requisitos, el modelado conceptual y los componentes de software cubriendo completamente el ciclo de vida del software.

Bibliografía

1. Insfrán E., Pastor O., Wieringa R. ***Requirements Engineering-Based Conceptual Modeling.*** Requirements Engineering Journal, num. 2 , vol. 7 . ISSN: 0947-3602, 2002.
2. Insfrán E., Pelechano V., O. Pastor. ***Conceptual Modeling in the Extreme*** Information and Software Technology, Volume 44, Issue 11, 15 August 2002, Pages 659-669.
3. Regnell B., Kimbler K., y Wesselen A. ***Improving the Use Case Driven Approach to Requirements Engineering***. In *Second IEEE International Symposium on Requirements Engineering*. IEEE, Marzo 1995.
4. Roetzheim W.H. ***Developing Software to Government Standards.*** Prentice Hall, 1991.
5. DoD-2167a. ***Military Standard – Defense System Software Development.*** US Department of Defense, 1988.

6. Edwards M., Howell S. ***A methodology for system requirements specification and traceability for large real- time complex systems.*** Technical Report, Naval Surface Warfare Center, 1991.
7. Palmer J. D. ***Traceability in Software Requirements Engineering***, R.H. Thayer and M. Dorfman (Eds), IEEE Computer Society Press, 1997. (364:374).
8. Pastor O., Insfrán E., Pelechano V., Romero J., Merseguer J. ***OO-Method: An OO Software Production Enviroment Combining Conventional and Formal Methods.*** En *9th Conference on Advanced Information Systems Engineering* (CAiSE'97), 145-159, Barcelona, España, 1997. Springer-Verlag. LNCS (1250).
9. Pastor O., Insfrán E., Pelechano V. and Ramírez S. ***Linking object-oriented conceptual modeling with object-oriented implementation in Java.*** In *Database and Expert System Applications (DEXA'97),* 132-142, Tolouse, Francia, 1997. Springer-Verlag. LNCS (1308).
10. Pastor O., Gomez J., Insfrán E., Pelechano V. ***The OO-Method approach for information systems modeling: From object-oriented conceptual modeling to automated programming Information Systems Journal.*** ISSN 0306-4379. Vol. 26. Pages 507-534. 2001.
11. Bell R. ***Code Generation from Object Models. Embedded Systems Programming.*** Vol. 11, March. Pages 23-33.1998.
12. Pelechano V., Pastor O., Insfrán E. ***Automated Code Generation of Dynamic Specializations. An Approach Based on Design Patterns and Formal Techniques.*** Data and Knowledge Engineering (DKE). Vol. 40 Nr. 3 Pages 315-354. ISSN: 0169-023X 2002.
13. Constantine L. ***Essential Modeling: Use Cases for User Interfaces.*** ACM Interactions, 2(2): 34-36. April, 1995.
14. Larman C. ***Applying UML and Patterns***. Prentice Hall. Upper Saddle River NJ, 1998.
15. Wir93 WIRFS-BROCK R. ***Designing Scenarios: Making the Case for a Use Case Framework***. Smalltalk Report, p. 9-20. Nov/Dec, 1993.
16. OMG ***Unified Modeling Language Specification***. Version 1.4. http://www.omg.org/uml
17. Jacobson I., Christerson M., Jonsson P. & Overgaard G. ***Object Oriented Software Engineering, a Use Case Driven Approach.*** Addison-Wesley. Reading, Massachussetts, 1992.
18. Eriksson H.E., Penker M. ***UML Toolkit.*** John Wiley and Sons. 1998.
19. Pooley R., Stevens P. ***Using UML: Software Engineering with Objects and Components.*** Addison Wesley, 2000.