INTEGRACIÓN INFORMAL DE MODELOS EN UML

Guillermo Bustos
gbustos@ucv.cl
Escuela de Ingeniería Industrial - Universidad Católica de Valparaíso
Av. Brasil 2241 - Casilla 4059 Valparaíso - Chile

Resumen

Este artículo presenta un conjunto de reglas flexibles para integrar diversos diagramas UML cuando usados principalmente para modelar sistemas de información o de negocios. Se presentan ordenadamente reglas para cada par de diagramas posibles con algunos ejemplos para ilustrar las relaciones. Estas reglas son útiles para estructurar y mejor aprovechar las representaciones con UML que deban hacer analistas de sistemas y diseñadores de procesos de negocio.

Abstract

This paper introduces a set of informal and flexible rules to integrate UML diagrams mainly used in information and business systems modeling. Rules are described for each possible couple of diagrams. Examples are added to illustrate the relationships between models. These rules are useful to structure and improve the UML representations for system analysts and business process designers.

1 Introducción

Ya no cabe duda que UML (***U**nified **M**odeling **L**anguage*) [1] se ha transformado en un estándar *de facto* para lo que a representaciones de *software* orientado a objetos se refiere. Desde su formulación inicial por la OMG[1] en 1997, su adopción por parte de la industria ha sido creciente. En estos años se ha discutido mucho sobre el tema, han habido seminarios, se siguen publicando artículos (como este mismo) y libros sobre el tema y varias herramientas CASE[2] ya lo soportan.

UML propone, entre otras cosas, un conjunto de diagramas que representan un *software* desde distintos puntos de vista. Por ejemplo, un Diagrama de Casos de Uso muestra la funcionalidad que ofrece desde la perspectiva de los usuarios externos al sistema, un Diagrama de Clases representa la estructura estática que las clases poseen y un Diagrama de Interacción representa cómo los objetos intercambian mensajes. Sin embargo, un aspecto que no ha sido tratado muy frecuentemente es cómo estos modelos se integran, es decir cuáles son las relaciones explícitas posibles entre estos diferentes diagramas cuando están describiendo un mismo sistema.

En este contexto, este artículo propone algunas reglas generales de integración para los diferentes diagramas que pueden usarse para representar un *software*. Sin prejuicio de lo anterior,

[1] ***O**bject **M**anagement **G**roup*
[2] ***C**omputer-**A**ided **S**oftware **E**ngineering*

las relaciones son planteadas de manera informal y flexible, pensando en la utilidad que estos diagramas pueden tener para representar también sistemas de información y sistemas de negocios, donde existe una mayor preocupación por los procesos y no se requiere tanta precisión como para un *software*.

El artículo se estructura con una sección inicial que revisa brevemente el lenguaje UML, asumiendo que el lector ya conoce los diagramas. La siguiente sección presenta las reglas de integración propuestas para cada par de diagramas posibles. Finalmente se entregan las conclusiones.

2 El Lenguaje UML

UML fue planteado originalmente en 1997 como una propuesta de estandarización para sistemas orientados a objetos, frente a la "guerra de métodos" que existía en ese entonces. UML es un lenguaje visual estándar para describir estructuras de sistemas basados en *software* [2]. Se utiliza para visualizar estructuras de *software*, especificar decisiones de análisis, diseño e implementación, construir código en cualquier lenguaje de programación y documentar completamente sistemas de *software* [2].

Es importante destacar que UML provee elementos para representar visualmente los sistemas de *software*, pero no provee ninguna metodología o proceso para guiar el desarrollo de los mismos. Es decir, UML es sólo una notación gráfica. Su aplicabilidad va desde sistemas técnicos, de tiempo-real, distribuidos, de *software* básico, hasta sistemas de información y de negocios [3].

UML presenta 3 bloques de construcción [2]:

1. Cosas: que son los elementos principales en los modelos de las cuales se quiere decir algo.
2. Relacionamientos: que expresan relaciones entre las cosas.
3. Diagramas: que agrupan colecciones relevantes de cosas relacionadas.

Además ofrece reglas semánticas para nombres, alcance y visibilidad, entre otros, y mecanismos tales como especificaciones, adornos y extensibilidad.

Para los efectos de este artículo, son relevantes los diagramas UML (según la versión 1.3) que serán brevemente descritos a continuación[3]:

- **Diagrama de Clases** (**DCla**): Describe las clases que existen en el sistema y las relaciones estructurales entre las mismas. Considera 3 tipos de estructuras: asociaciones, todo/parte y herencia.
- **Diagrama de Interacción** (**DInt**): Describe la interacción en términos de mensajes entre los objetos de las clases. Existen 2 tipos de DInt: el **Diagrama de Colaboración** (**DCol**), que organiza la interacción espacialmente, y el **Diagrama de Secuencia** (**DSec**), que enfatiza la organización temporal de la interacción.
- **Diagrama de Estado** (**DEst**): Describe el comportamiento genérico que exhiben los objetos de una clase. Representa los estados, que se entienden como pasos en el comportamiento del objeto, y sus respectivas transiciones.
- **Diagrama de Casos de Uso** (**DCU**): Describe los casos de uso posibles en el sistema y las relaciones entre ellos. Representa una porción de funcionalidad que el sistema entrega a

[3] Se asume que el lector ya se encuentra familiarizado con estas representaciones.

cada tipo de actor externo. Las relaciones pueden ser de asociación, extensión, generalización e inclusión. Cada caso de uso debe también describirse con una **Documentación de Casos de Uso** (**DoCU**).

- **Diagrama de Actividades** (**DAct**): Describe el flujo de trabajo (*workflow*) entre actividades. Las actividades pueden organizarse de manera secuencial, condicionada y concurrente. Adicionalmente, pueden usarse carriles verticales (*swimlanes*) para mostrar los responsables de cada actividad.

3 Algunas Reglas de Integración

UML no provee explícitamente reglas de consistencia entre los diferentes diagramas que representen un sistema. Esto se debe principalmente a que se busca privilegiar la flexibilidad de uso, es decir, permitir la utilización del(los) diagrama(s) más apropiado(s) para lograr la representación que se desea.

No obstante lo anterior, la integración de los modelos debe ser adecuadamente hecha con el fin de tener la consistencia necesaria a toda construcción de múltiples modelos. La literatura no es muy numerosa en lo que se refiere a la formalización de esta consistencia. Algunos autores en [4], [5], [6], [7] y [8] han desarrollado algunos esfuerzos principalmente enfocados en la verificación de consistencia para herramientas de apoyo a la construcción de *software*, pero generalmente se centran en unos pocos modelos con gran detalle y, en particular, el DAct muchas veces no es considerado.

Sin el ánimo de ser sistemático en la definición de reglas de consistencia (lo cual sólo puede alcanzarse mediante la formalización matemática de UML), se propone a continuación un conjunto de reglas de integración que dan un panorama amplio de cómo relacionar los diagramas UML en el modelado orientado a objetos. Estas reglas son más bien flexibles, para permitir el modelado de sistemas de información y de negocios y no sólo de *software*, razón por la cual el DAct pasa a ser un modelo importante.

La tabla 1 muestra el mapa de reglas con el producto cartesiano de los diagramas considerados. Cada entrada de la tabla define el número de la sección que a seguir describe la forma de relacionar ambos diagramas o un diagrama y su documentación.

Tabla 1 – Mapa de reglas de integración de UML.

Diagramas	DCla	DInt	DEst	DCU	DoCU	DAct
DCla		3.1	3.2	3.3		3.4
DInt	3.1		3.5	3.6		
DEst	3.2	3.5				3.7
DCU	3.3	3.6			3.8	3.9
DoCU				3.8		
DAct	3.4		3.7	3.9		

3.1 DCla vs. DInt

Las reglas para estos diagramas pueden enunciarse como sigue [8], [9], [10] (ver figura 1):

- Todos los objetos que interactúan en un DInt son instancias de alguna clase del DCla.
- Todos los mensajes en un DInt deben corresponder a las operaciones de los objetos receptores, cuya clase está en el DCla.
- Las vías de comunicación entre objetos en el caso de un DCol corresponden a asociaciones en el DCla (con la sola excepción de los automensajes).

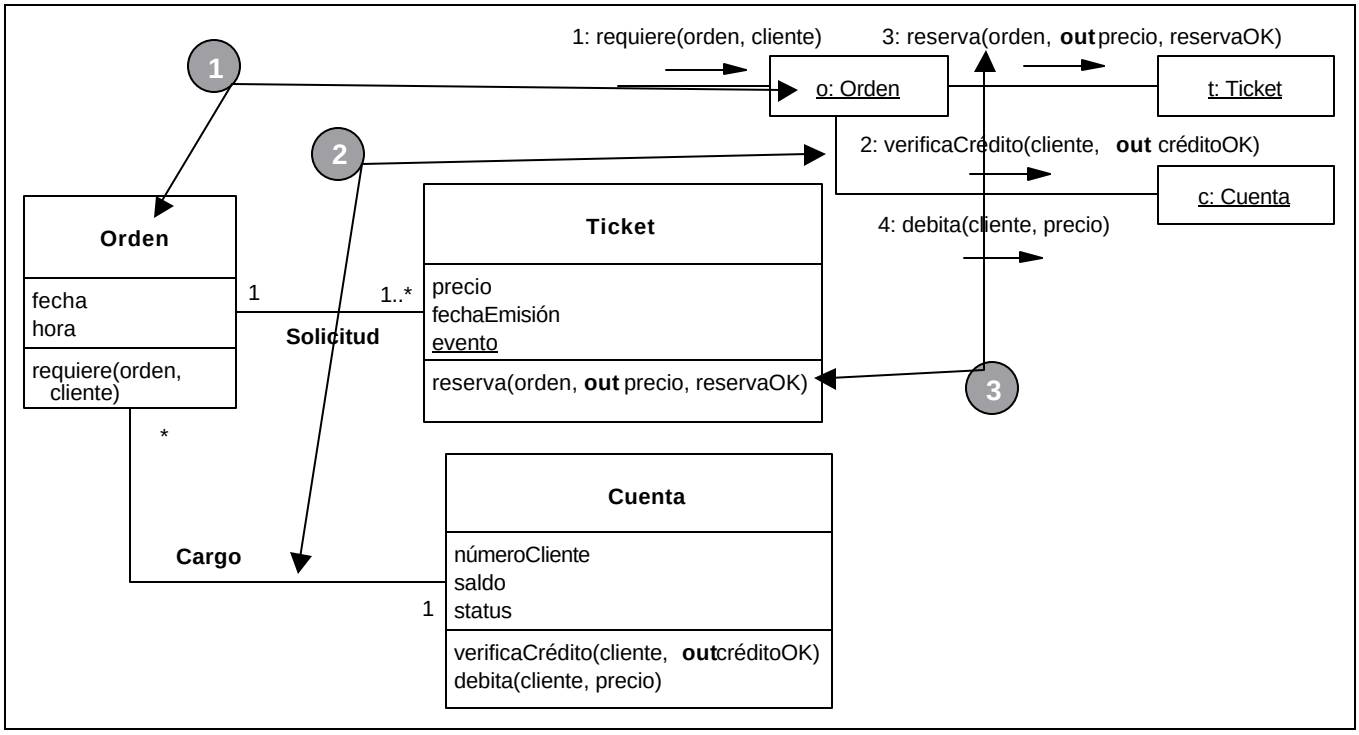

Figura 1 – Relaciones entre DCla y DInt (DCol en este caso): (1) clase (DCla) & objeto (DCol); (2) asociación (DCla) & vía de comunicación (DCol); y (3) operación (DCla) & mensaje (DCol).

3.2 DCla vs. DEst

Las reglas para estos diagramas pueden enunciarse como sigue [5], [10] (ver figura 2):

- Todo DEst describe el comportamiento de los objetos de una clase del DCla. Sin embargo lo inverso no siempre ocurre dado que muchas clases pueden exhibir un comportamiento trivial y no requieren necesariamente de un DEst.
- Todo evento en un DEst debe corresponder a una de las operaciones de la clase respectiva en el DCla.
- Toda acción en un DEst de una clase A debe corresponder a una operación de una clase B. En el DCla deben aparecer las clases A y B asociadas (con la sola excepción de que A pueda ser igual a B).

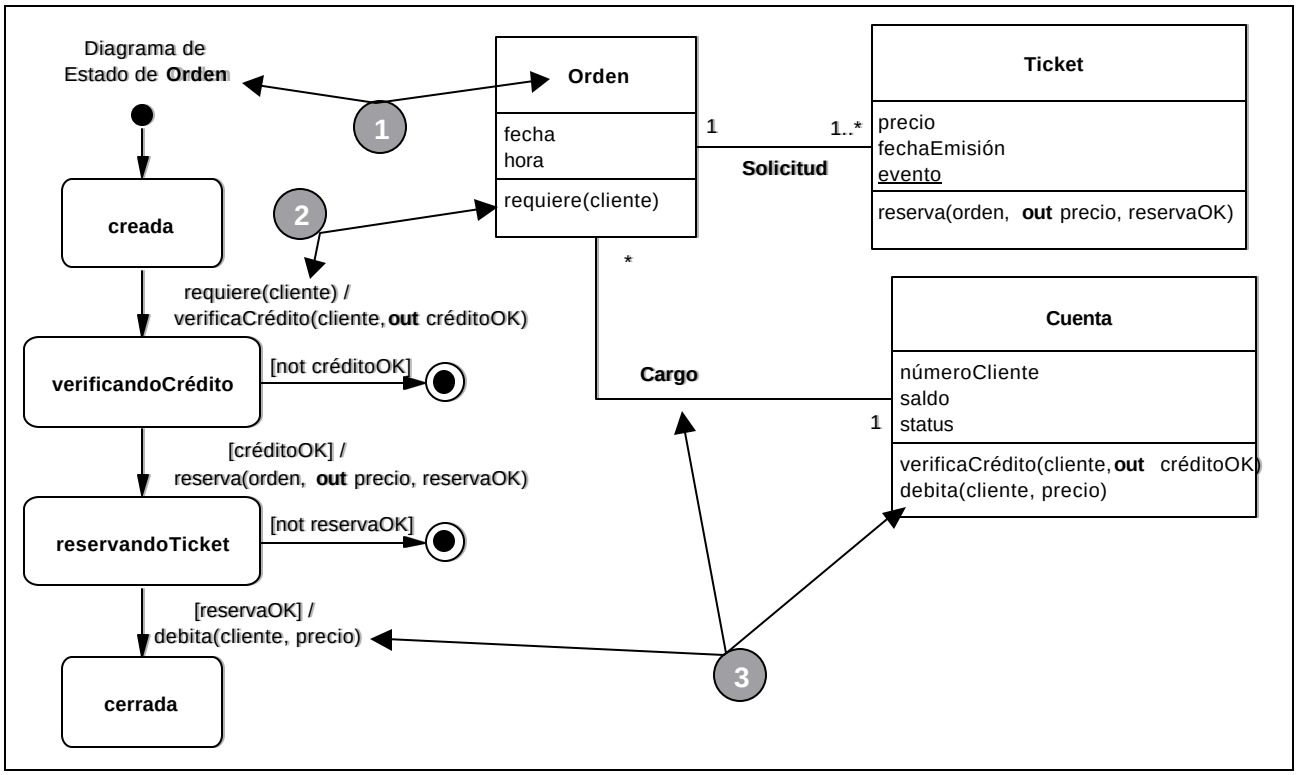

Figura 2 – Relaciones entre DCla y DEst: (1) clase (DCla) & comportamiento (DEst); (2) operación (DCla) & evento (DEst); y (3) operación y asociación (DCla) & acción (DEst).

3.3 DCla vs. DCU

La regla para estos diagramas puede enunciarse como sigue: el DCla puede representar la estructura de las clases involucradas en:

1. Un Caso de Uso (CU),
2. Todos los CU de un DCU,
3. Todos los CU de un actor, y
4. Todos los CU del sistema.

En los casos 1, 2 y 3 se recomienda finalmente integrar los DCla parciales para obtener el DCla de todo el sistema.

No existe correlación interna entre el DCla y el(los) CU de los DCU ya que ambos modelos se complementan: el DCU tiene una visión externa, asumiendo que el sistema es una caja negra, y el DCla tiene una visión interna, abriendo esta caja negra desde el punto de vista estructural. Se espera que haya coherencia de tal forma que las clases del DCla satisfagan los CU del DCU.

3.4 DCla vs. DAct

El DAct puede utilizarse para describir operaciones poco conocidas o muy complejas algorítmicamente, que aparecen en las clases del DCla [9]. Siendo éste el caso, en el DAct se debe manipular consistentemente las propiedades del objeto (atributos y otras operaciones) y los argu-

mentos de la operación descrita. La figura 3 presenta un ejemplo de esta relación entre DCla y DAct.

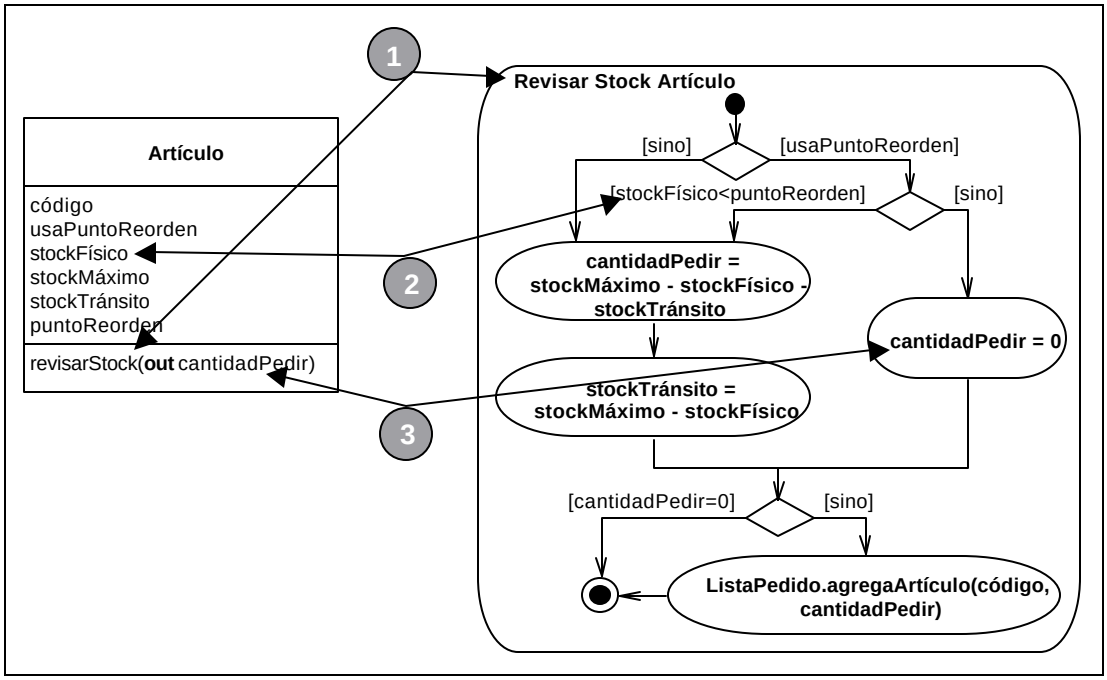

Figura 3 – Relaciones entre DCla y DAct: (1) operación (DCla) & todo el DAct; (2) atributo (DCla) & variable (DAct); y (3) argumento de operación (DCla) & variable (DAct).

3.5 DInt vs. DEst

Las reglas para estos diagramas pueden enunciarse como sigue [5], [11] (ver figura 4):

- Todo DEst describe el comportamiento de los objetos que interactúan en un DInt. Sin embargo lo inverso no siempre ocurre (al igual que con el DCla vs. DEst), dado que muchos objetos pueden exhibir un comportamiento simple y no requerir un DEst.
- Todo evento en un DEst debe corresponder a un mensaje entrante al objeto correspondiente en el DInt.
- Toda acción en un DEst debe corresponder a un mensaje saliente del objeto respectivo en el DInt.

3.6 DInt vs. DCU

El DInt puede usarse para representar la interacción de los objetos de:

1. Un CU.
2. Todos los CU de un actor.
3. Todos los CU del sistema.

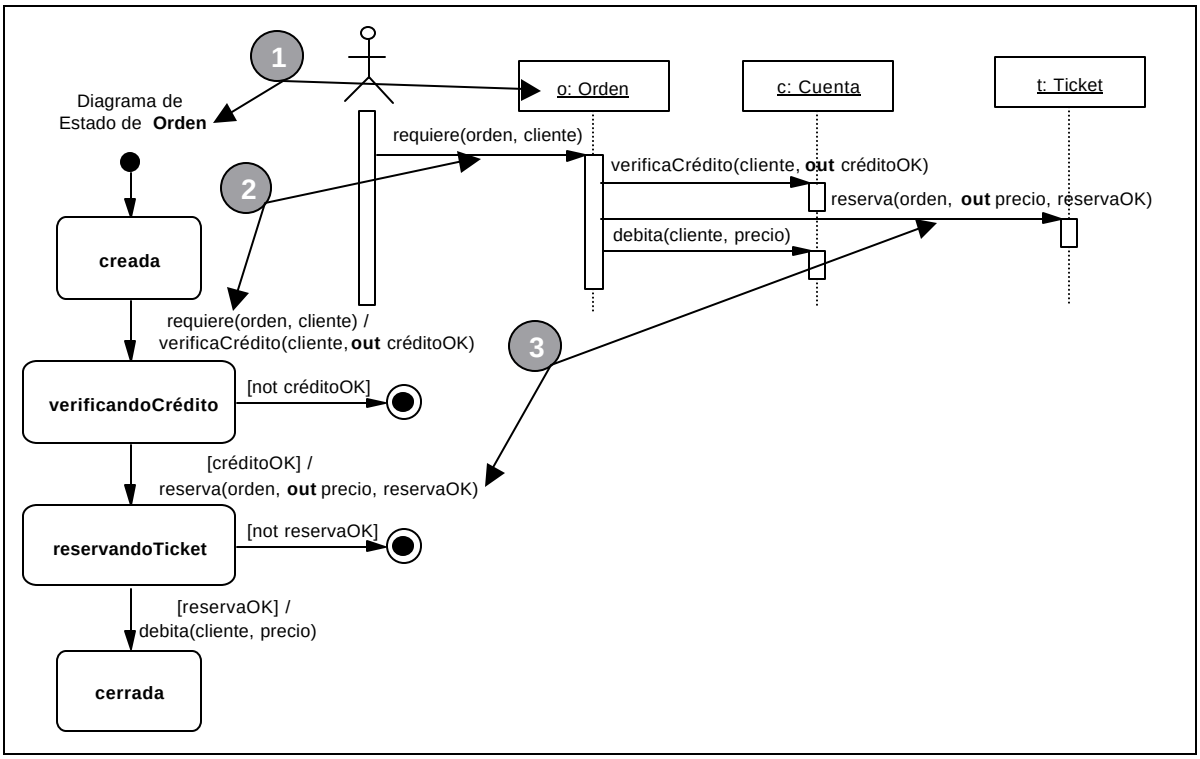

Figura 4 – Relaciones entre DInt (DSec en este caso) y DEst: (1) objeto (DSec) & comportamiento (DEst); (2) mensaje entrante (DSec) & evento (DEst); y (3) mensaje saliente (DSec) & acción (DEst).

En los casos 1 y 2 no se recomienda integrar los DInt parciales dado que sólo aumentaría la complejidad de la representación sin obtener claros beneficios. En estos casos se debe considerar las siguientes consistencias (ver figura 5):

- El actor en el DCU corresponde al emisor del DInt.
- El mensaje inicial en el DInt debe provenir del actor en el DCU.

El caso 3 no se recomienda, también razones de complejidad, sin embargo existe una alternativa más simple que consiste en usar el DInt para representar sólo la interacción del actor del CU con el sistema, visto este último como una caja negra y por lo tanto, sin incluir ningún objeto del sistema [11]. En esta alternativa, las mismas consistencias de los casos 1 y 2 deben ser consideradas.

3.7 DEst vs. DAct

El DAct se define como un caso especial de un DEst, donde sólo se representan los estados que poseen actividades, destacando justamente estas últimas. No obstante esta concepción estructural del DAct a partir del DEst no impide identificar diferentes grados de relación entre estos diagramas cuando usados para modelar un sistema. En este sentido un DAct puede representar actividades:

- No asociadas a ningún DEst.
- De los estados de un DEst (caso de dependencia ya indicado).
- De los estados de varios DEst.

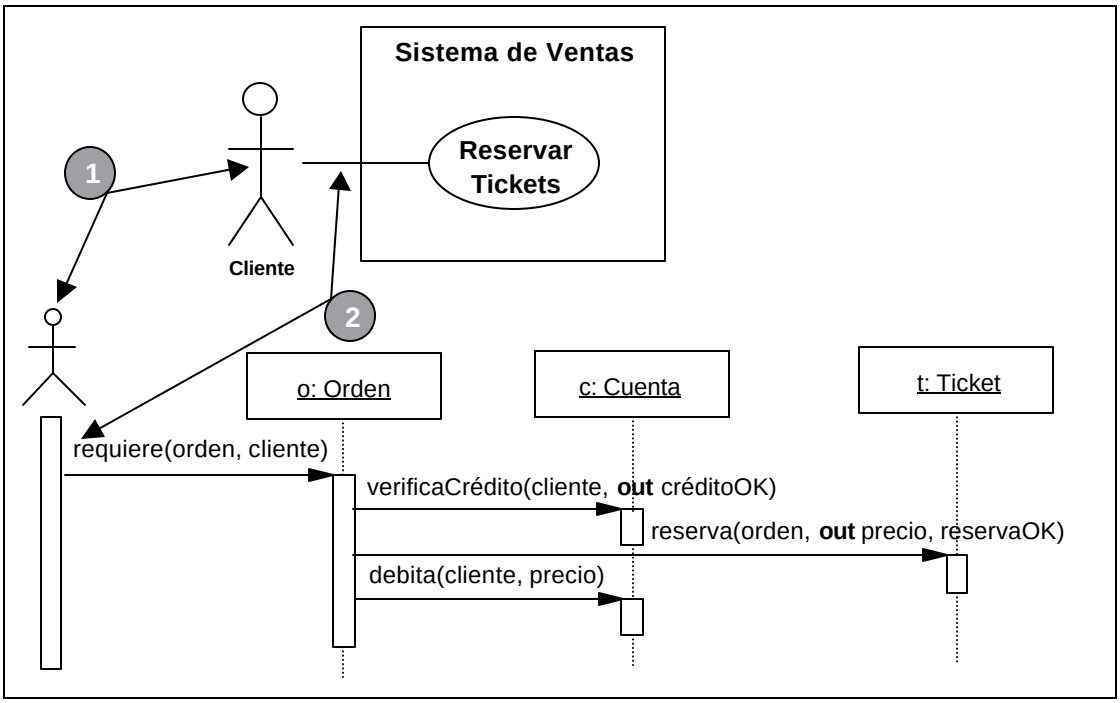

Figura 5 – Relaciones entre DInt (DSec en este caso) y DCU: (1) emisor (DSec) & actor (DCU); y (2) mensaje inicial (DSec) & asociación (DCU).

3.8 DCU vs. DoCU

Existen varias alternativas para enfrentar la descripción de los CU en un DCU. Si se opta por una documentación más textual, puede usarse pre y post-condiciones, tablas de decisión o lenguaje estructurado con alternativas, entre las formas más comunes. Sin embargo, esta documentación puede hacerse por medio de otros diagramas, tales como el DInt (caso mostrado en la sección 3.6), el DCla (caso de la sección 3.3) y el DAct (caso de la sección 3.9 a seguir).

Estas alternativas pueden combinarse, así por ejemplo puede usarse un DInt y una descripción con lenguaje estructurado con alternativas [11], u optar por usar un DAct complementado con pre y post-condiciones. Incluso, para algunos propósitos, puede usarse un DAct y un DInt para documentar un DCU si se estima necesario.

3.9 DCU vs. DAct

El DAct puede usarse para describir:

- Un CU (ver figura 6).
- Grupos de CU (por ejemplo de un mismo actor).
- Todos los CU del sistema.

Aunque las relaciones anteriores son recomendables por su complementariedad, también es posible usar los DAct y DCU de manera completamente independiente.

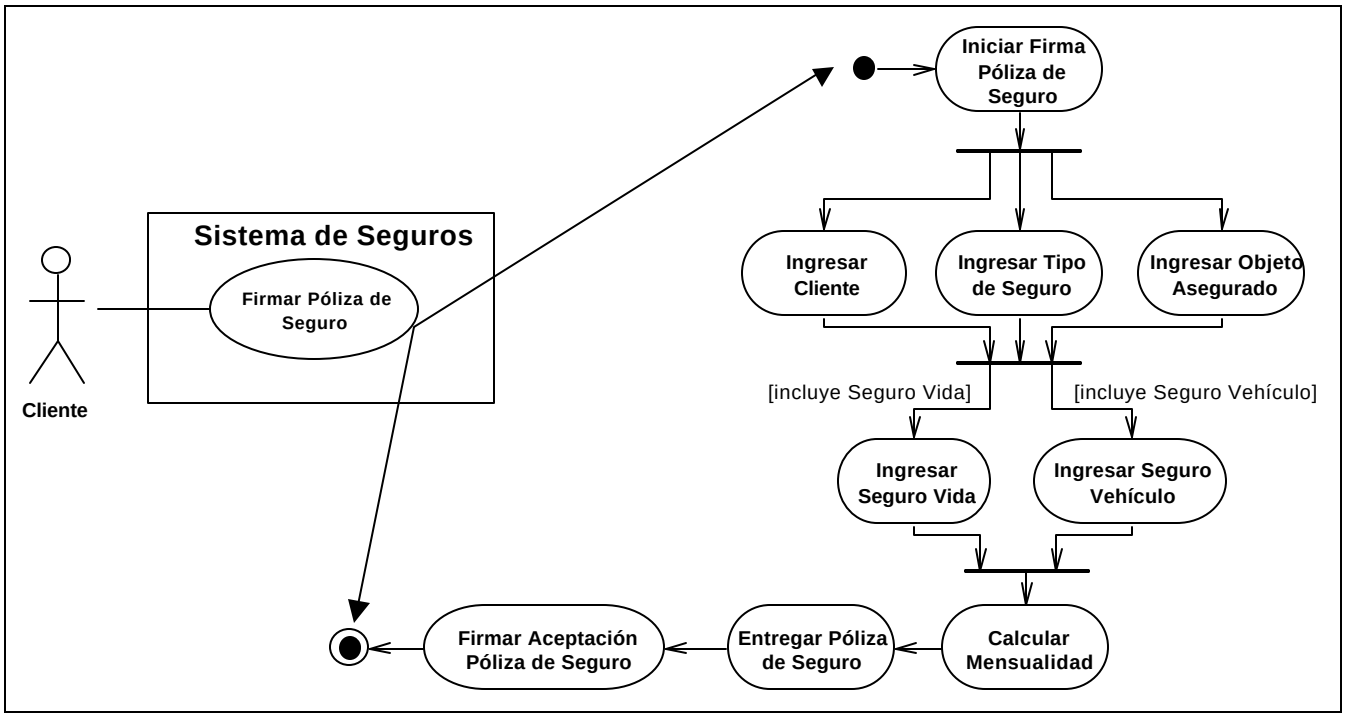

Figura 6 – Relaciones entre un CU de un DCU y DAct.

4 Conclusiones

Se ha presentado un conjunto informal de reglas para integrar distintos diagramas UML. Estas reglas son flexibles en el sentido que consideran varias alternativas de relación para algunos pares de diagramas.

La principal utilidad de estas reglas es la de permitir estructurar representaciones usando UML por parte de analistas de sistemas y de diseñadores de procesos de negocios. Además, se puede explotar de mejor forma cada uno de los diagramas, al conocer su rol parcial dentro de un contexto integrado, donde se le reconoce a cada modelo su complementariedad.

No obstante lo anterior, no se debe entender que se requiere usar todos los diagramas en todos los sistemas que se deseen modelar. Muchas veces 2 ó 3 diagramas son suficientes para lograr la representación con el detalle necesario para un proceso de negocio o un sistema de información. Por esto, en [2] se indica que “se puede modelar el 80% de los problemas usando sólo el 20% de UML.”

Finalmente, estas reglas por su carácter informal, pueden ser perfeccionadas tanto con la adición de nuevas alternativas de relación entre los diagramas, como por la vía de la formalización, para tener un grado de precisión similar al que se está alcanzando para especificar sistemas de *software*.

Bibliografía

[1] UML – Unified Modeling Language. Disponible vía web en http://www.uml.org/

[2] G. Booch, J. Rumbaugh, I. Jacobson; The Unified Modeling Language User Guide; Addison Wesley, 1999.

[3] H. Eriksson, M. Penker; UML Toolkit; John Wiley & Sons, 1998.

[4] R. Breu et al.; "Towards a Formalization of the Unified Modeling Language". In Proceedings of European Conference on Object-Oriented Programming ECOOP' 97, Jyväskylä (Finlandia), June 1997.

[5] R. S. Ferreira Resende; B. Lagoeiro Araujo; "Consistencia de Diagramas UML". Memorias IDEAS 2000 – Jornadas Iberoamericanas de Ingeniería de Requisitos y Ambientes de *Software*, Cancún (México), Abril 2000.

[6] A. Egyed; "Semantic Abstraction Rules for Class Diagram". In Proceedings of the 15th IEEE International Conference on Automated *Software* Engineering – ASE, Grenoble (Francia), 2000.

[7] N. Medvidovic et al.; "*Software* Model Connectors: Bridging Models Across the *Software* Lifecycle". In Proceedings of the 13th International Conference on *Software* Engineering and Knowledge Engineering – SEKE, Buenos Aires (Argentina), June 2001.

[8] A. Tsiolakis; "Integrating Model Information in UML Sequence Diagrams". In Proceedings of the Satellite Workshops of the 28th International Colloquium on Automata, Languages and Programming – ICALP, Creta (Grecia), July 2001.

[9] M. Fowler, K. Scott; UML Distilled; Addison-Wesley, 2000, 2nd ed.

[10] M. Page-Jones; Fundamentals of Object-Oriented Design in UML; Addison-Wesley, 2000, 1st ed.

[11] C. Larman; Applying UML and Patterns; Prentice Hall PTR, 2002, 2nd ed.

Autor
Guillermo Bustos R.
Ingeniero Civil en Informática
Doctor en Ciencia de la Computación
Académico de la Escuela de Ingeniería Industrial