Definición de modelos de objetos a partir de sus responsabilidades

Juan Francisco Giró
Dpto. Ing. Sistemas de Información, Facultad Regional Córdoba, UTN
Dpto de Estructuras, Fac. Ciencias Exactas, Físicas y Naturales, UNC

Abstract
A cuarenta años de la identificación de la Crisis de Software se comprueba que ciertos conceptos fundamentales no están claros para muchos miembros de la comunidad informática. Entre otras manifestaciones, pueden mencionarse la pérdida de identidad de las etapas de Análisis y Diseño de un sistema y la falta de procedimientos claros para la definición de modelos de objetos. En este trabajo se estudia el alcance y posibles causas de estas manifestaciones y se presenta un procedimiento para la definición de modelos de objetos que aseguren su trazabilidad. Finalmente, se presenta un caso de estudio y se discuten los resultados obtenidos.

Palabras Clave
Modelos Conceptuales. Casos de Uso. Responsabilidades. Modelos de Objetos. Trazabilidad. Diagrama de Clases.

1. Introducción

Hay dos aspectos, fundamentales y estrechamente vinculados entre sí, que parecen no recibir adecuada atención por parte de muchos de los miembros de la comunidad informática: 1) La pérdida de identidad de las etapas de Análisis y Diseño al abordarse un problema y 2) La ausencia de un procedimiento claro que haga posible el tránsito ordenado desde una a otra etapa, es decir del Análisis al Diseño, cuando se adopta un enfoque orientado a objetos.

Esta circunstancia despertó la inquietud por identificar las causas del problema y procurar contribuir a esclarecer estos temas, o por lo menos propiciar su debate, convirtiéndose en la fuente de inspiración de este trabajo.

El tratamiento de estas y otras ideas centrales de la Ingeniería de Software pueden quedar enmarcados en la celebración del 40º aniversario de la identificación de la llamada Crisis de Software[1]. Las manifestaciones de esta crisis fueron los pobres resultados obtenidos con los desarrollos de sistemas, muchos de ellos de dudosa calidad, que eran difícilmente completados en los plazos y presupuestos previstos. Fue entonces cuando un grupo de estudios de la NATO consideró que estas deficiencias crónicas de los proyectos de sistemas podrían ser superadas, adaptando y aplicando al software las prácticas que eran de uso común en las ramas tradicionales de la ingeniería.

Cabe aquí también recordar que veinte años más tarde, en su célebre artículo "No Silver Bullet", Frederick Brooks [3] advirtió que la Crisis de Software no había sido resuelta, y posteriormente, a mediados de los '90, Wayt Gibbs [4] volvió sobre este tema reconociendo que la Crisis de Software se mantenía aún presente y sin mayores cambios. En este contexto, parece válido preguntarse cuál es la situación actual y en que medida los aspectos antes señalados son una manifestación en uno u otro sentido.

El resto de este trabajo se organiza de la siguiente manera. En la *Sección 2* se revisan conceptos de modelado de sistemas y se considera la identificación de objetos, prestando especial atención a los *Casos de Uso* por ser la técnica de modelado dominante en la actualidad. Luego, en la *Sección 3* se propone una metodología que contribuya a superar las dificultades identificadas en la sección anterior. En la *Sección 4* se aplica la metodología propuesta a un caso real y se muestran algunos

[1]El término Crisis de Software fue establecido por F.L.Bauer [1] en la primer Conferencia de Ingeniería de Software de la NATO en Garmisch, Alemania. Con posterioridad, Edsger Dijkstra [2] fue el primero en hacer referencia a la Crisis de Software.

resultados de ese proceso. Finalmente, en la *Sección 5* se presentan las conclusiones de este trabajo e ideas para continuarlo en el futuro.

2. Desde el Análisis al Diseño

Es conveniente comenzar por concentrar la atención en el modelado de sistemas, donde se comprueban mensajes contradictorios que no parecen contribuir a sentar las bases sólidas que son necesarias para resolver problemas. Hasta hace algunos años, la necesidad de una clara distinción conceptual entre las etapas de análisis y diseño, en el ciclo de vida del desarrollo de software, estaba fuera de toda discusión. En la actualidad, sin embargo, parecería que tal distinción viene perdiendo vigencia, de la mano de las metodologías ágiles y la de algunos autores que ofrecen técnicas de desarrollo de sistemas en las que no está claro el lugar que ocupan ambas etapas.

Inmediatamente, podría formularse un interrogante: ¿hasta qué punto puede esperarse que se dé respuesta apropiada a un problema que todavía no ha sido completamente reconocido? Fue René Descartes [5] quien estableció con toda claridad que "el análisis de una cuestión en sus aspectos más simples se antepone a la síntesis, o recomposición ulterior de los conocimientos, para llegar con la ayuda del razonamiento hasta el conocimiento de lo complejo" y no hay nada que indique que este ordenamiento natural pueda ser cambiado.

Tradicionalmente se identifican cuatro modelos básicos: 1) Modelo conceptual (lo que el usuario sabe o debería saber sobre el sistema deseado), 2) Modelo de Análisis (la descripción de lo que el sistema debe hacer), 3) Modelo de Diseño (la descripción de cómo el sistema será implementado) y 4) Modelo Operativo (la materialización del propio sistema), a los que suelen sumarse algunos otros modelos auxiliares. El proceso de convertir el Modelo Conceptual en el Modelo de Análisis está reservada a la etapa de Análisis, y el de convertir el Modelo de Análisis en el de Diseño, a la etapa de Diseño. Antes tiene lugar la fundamental etapa de Elicitación de Requerimientos, cuya finalidad es contribuir a hacer posible el Modelo Conceptual. En este contexto, debe observarse que si fuese posible asegurar la completitud de este primer modelo, el desarrollo evolutivo perdería buena parte de su justificación.

Todos los modelos del ciclo de vida del desarrollo de software prevén éstas etapas iniciales de ingeniería de requerimientos, que son seguidas por las de diseño, programación y testeo. A las etapas iniciales se les reconoce particular importancia, por ser las encargadas de definir el primer modelo conceptual del sistema a ser desarrollado, y es donde se realizará la identificación, análisis y documentación de los requerimientos. En este contexto, el Análisis debe orientarse al problema e identificar sus características esenciales, prescindiendo de cualquier aspecto que esté vinculado a posibles soluciones. Será el Diseño el que tendrá posteriormente la finalidad de encontrar la mejor solución, apoyarse en criterios de optimización y considerar todas las posibles restricciones, incluidas las tecnológicas.

En resumen, obsérvese que en el momento en que se deja de procurar entender un problema para comenzar a idear la forma de resolverlo, se abandona la etapa de Análisis y se comienza con la de Diseño. Así, el resultado final que se obtenga estará decididamente influenciado por esta trascendental decisión.

El punto de partida para la definición de los modelos conceptuales es establecer con toda claridad los límites del sistema. Luego, en su descripción, se deben considerar sus tres dimensiones o perspectivas relativamente ortogonales, que son: 1) Dimensión funcional, con las transformaciones que debe realizar el sistema, 2) Dimensión estática, con las descripciones y propiedades de su estructura y 3) Dimensión dinámica, con el comportamiento individual y colectivo de sus componentes. Por lo tanto, las posibles estrategias generales para la elaboración de un modelo quedarán establecidas según el orden en que estas propiedades sean reconocidas al estudiarse el dominio del problema. Así, a lo largo del tiempo, se han ensayado estrategias dirigidas por procesos, dirigidas por datos y dirigidas por comportamiento, y las ventajas e inconvenientes de cada una es tema de un debate todavía abierto.

Obviamente, en todos los casos, la definición de las propiedades de los mismos elementos desde las tres dimensiones ortogonales debe exhibir consistencia, requisito fundamental para la obtención de modelos balanceados, según la terminología empleada por Yourdon [6]. Estos modelos deben, además, presentar otras propiedades que son consideradas igualmente esenciales, esperándose que sean completos, correctos, precisos, no ambiguos y trazables. Para alcanzarlas, los diferentes paradigmas del desarrollo de sistemas han presentado sus propuestas destinadas a la obtención de este primer modelo conceptual.

Naturalmente, los *Casos de Uso* son una referencia obligada por representar la técnica de modelado dominante en la actualidad. Por su gran difusión, los propios *Casos de Uso* no requieren de mayores explicaciones. Sin embargo, resulta curioso comprobar el gran desconocimiento que hay sobre lo que puede esperarse de ellos y, en particular, sobre la forma de transitar hacia un modelo de objetos.

Los *Casos de Uso* son una abstracción sobre un conjunto de escenarios que describe la funcionalidad de un sistema, por lo que a través de todos sus *Casos de Uso* el modelo funcional debería quedar completamente especificado.

Cabe aquí agregar que Constantine propuso incrementar el nivel de abstracción de los *Casos de Uso* para dar lugar a los denominados *Casos de Uso Esenciales* [7]. Para ello incorporó conceptos del *Análisis Esencial* de McMenamin y Palmer [8], que ya había servido de base a la versión original de los *Casos de Uso* de Jacobson [9]. De esta manera, los *Escenarios*, *Casos de Uso* y *Casos de Uso Esenciales* representan sucesivos modelos de abstracción, idealización y generalización.

Se presenta aquí una aparente contradicción con profundas connotaciones históricas: luego de explorarse numerosas alternativas, el enfoque de objetos incorporó una técnica ajena al paradigma para la definición del modelo de análisis: los *Casos de Uso*. Esto no debería sorprender si se considera que los conceptos de análisis (descomposición y examen crítico) y de orientación a objetos (encapsulamiento de atributos y capacidades) en realidad se contraponen [10]. Una forma de resolver este conflicto fue postergar el encapsulamiento tanto como fuese posible, lo que implicó volver a sacar el modelado conceptual fuera del paradigma de objetos, adoptando para ello métodos que dispongan de la necesaria flexibilidad y poder expresivo. Bajo este razonamiento, se retornó a la utilización de modelos no orientados a objetos en la especificación de requerimientos, para luego derivar de éstos los modelos de objetos.

El propio Jacobson deja este punto en claro al decir que "el modelo de *Casos de Uso* representa una visión externa del sistema, mientras que el modelo de objetos es una visión interior del mismo" [11].

Si bien la naturaleza funcional de los *Casos de Uso* no constituye de por sí un problema, es el principal motivo de confusión cuando son utilizados en un contexto de desarrollo orientado a objetos, ya que las funciones no son objetos y los objetos no son funciones. Los *Casos de Uso* permitirán una visión externa, orientada al usuario de un sistema, que nunca podrá ser tomada como base para definir su arquitectura de objetos. En efecto, esta última arquitectura diferirá significativamente de su arquitectura de descomposición funcional. Sin embargo, hay autores como Kendall K. y Kendall J. [12], que no contribuyen a esclarecer el tema al sostener que "El *UML* se basa esencialmente en una técnica orientada a objetos conocida como modelado de *Casos de Uso*" (Pg.699).

En respuesta a esta apreciación, son oportunas las consideraciones de Craig Larman, que dice: "No hay nada de orientación a objetos en los *Casos de Uso*… Esto no es un defecto, pero sí un punto que debe ser esclarecido… Sin embargo, como se apreciará, los *Casos de Uso* son una puerta de entrada a las actividades de Análisis y Diseño Orientados a Objetos" [13].

Una vez que se han completado los *Casos de Uso* que corresponden a cierto problema, la situación es la siguiente:

a) Sólo se dispone de una incipiente definición de las propiedades estáticas del modelo, que son las obtenidas al estudiarse su dimensión funcional.

b) Se carece de información sobre las propiedades dinámicas o dimensión de comportamiento.
c) Ha quedado claramente establecida una brecha, o gap, entre el modelo funcional desarrollado hasta el momento y el modelo de objetos requerido para ingresar en la fase de diseño.

Sobre este último aspecto, Fernandez y Lilius [14] reconocen que "el salto desde los *Casos de Uso* y *Escenarios* a las *Clases* es, en nuestra opinión, muy grande... Este paso requiere demasiado ingenio, ya que no hay ninguna relación directa evidente entre *Casos de Uso* y *Clases*".

Como se verá, la bibliografía no contribuye demasiado a superar este problema, tal como lo testimonian algunas referencias que se brindan a continuación. Para comenzar, puede citarse a Bruce P. Douglass [15], quién reconoce la necesidad de saltar esta brecha entre paradigmas y admite que no hay ninguna forma automática que permita pasar de una visión funcional, de caja negra de un sistema, a una vista orientada a objetos. En este caso, la metodología propuesta por Douglass, denominada Ropes [16], postula que hay alrededor de una docena de estrategias diferentes que permiten definir los objetos con suficiente grado de aproximación. Similarmente, Ian Sommerville [17] reconoce que hay varias formas de identificar objetos y, en la práctica, "se utilizan para ello diversas fuentes de conocimiento" (Pg 297). Por su parte, Kendall y Kendall [12] propone "desarrollar Diagramas de Secuencia y de Colaboración a partir de los escenarios de los *Casos de Uso*" (sin indicar cómo) y luego recomienda "buscar sustantivos en los *Casos de Uso*…, por ser objetos potenciales" (Pg. 696). Arlow y Neustadt [18] también reconocen la existencia de varias técnicas para la identificación de clases, similares a las de las propuestas anteriores.

Muchas de estas estrategias están inspiradas en un trabajo de Abbott [19], denominado Análisis Textual, según el cual: a) los nombres son candidatos a quedar representados por clases, b) los verbos representan operaciones y c) los adjetivos representan atributos. Al margen de la opinión que merezca la rigurosidad de esta técnica de identificación de objetos, está claro que se apoya en una especificación escrita de los requerimientos y no está tan claro cómo se la vincula a los *Casos de Uso* definidos con anterioridad. Al igual que en otras técnicas, como el "brainstorming" o el "role playing" propuestos por Bellin y Suchman [20], todas ellas procuran identificar primero los objetos, para luego asignarles sus atributos y sus métodos, en un proceso que podría denominarse de explosión de propiedades. Luego, definidos los objetos, se procederá a identificar las clases.

En resumen, el desarrollo de los *Casos de Uso* brindará un modelo que representará las propiedades del sistema en una sola dimensión, sin identificación directa de objeto alguno, y menos aún de una estructura de clases. Con mucha frecuencia, se desarrollan modelos de *Casos de Uso* en los que no parece advertirse que el esfuerzo realizado no conduce racionalmente a la Fase de Diseño. En otros casos, para cubrir esta brecha se recurre a heurísticas de las más variadas, y con ello nadie podrá asegurar que el modelo a ser obtenido será consistente, completo, correcto, preciso, no ambiguo ni trazable.

Esta última propiedad del modelo, su trazabilidad, merece un párrafo aparte. Rubin y Goldberg [21] reconocieron que un indicador fundamental de la madurez de una metodología la da la posibilidad de que sus resultados puedan ser validados y verificados, y "la clave que permite alcanzar ese nivel de madurez es la trazabilidad". En efecto, la ausencia de trazabilidad impedirá la objetiva comprobación de las restantes propiedades.

Sin embargo, estas aparentes falencias de los *Casos de Uso* no son tales, ya que debe advertirse que sólo falta convertir el modelo funcional al paradigma de objetos y completarlo en las dimensiones restantes. En realidad, el verdadero problema sólo es el de cubrir la brecha entre paradigmas, ya que tanto la dimensión estática como la dinámica se completarán con mucha facilidad una vez que se haya definido el diagrama de clases. Por lo tanto, aquí puede concluirse en que la correcta identificación de las clases es crucial en un modelado de objetos.

Tal como ya fue anticipado, la mayoría de las técnicas son explosivas, en el sentido que

identifican objetos para luego asignarles sus propiedades. Sin embargo, hay también propuestas implosivas, mucho menos difundidas pero probablemente más racionales, que recomiendan operar en sentido inverso.

Los trabajos de Biddle y otros [22, 23 y 24] son un ejemplo de este enfoque implosivo, y se apoyan en la identificación de las responsabilidades del sistema para la definición del modelo de objetos. Otra precursora en centrar la atención en las responsabilidades fue Rebecca Wirfs-Brock, con su "diseño dirigido por responsabilidades" [25], sólo que con un enfoque explosivo en que los objetos debían ser identificados con anticipación.

Aquí es necesario recordar que, en el modelado de objetos, las responsabilidades quedan expresadas por sentencias que describen servicios, ya sean referidas a acciones que deben ser realizadas o conocimientos que deben mantenerse y proveerse. Obsérvese que una responsabilidad no es lo mismo que un método, ya que los métodos son implementados para hacerse cargo de las responsabilidades. Además, la conversión de responsabilidades en métodos depende fuertemente de la granularidad de la responsabilidad. En efecto, las responsabilidades son implementadas a través de métodos que actúan en forma independiente, o colaboran con otros métodos y objetos, en esquemas interrelacionados que pueden llegar a ser muy complejos. Así, el hilo conductor entre las responsabilidades y los objetos pasa necesariamente por los métodos.

3. Una forma de superar la brecha

3.1 Ideas centrales

La propuesta que aquí se presenta recurre a las responsabilidades del sistema para construir un puente entre el modelo funcional y el modelo de objetos, perfeccionando un procedimiento presentado con anterioridad [26] y que tiene un antecedente en Meilir Page-Jones [27], que utilizaba los eventos del Análisis Esencial para definir sub-modelos que conducían a los objetos, en un método que denominó "Synthesis".

El procedimiento propuesto tiene bastante en común con los mencionados trabajos de Biddle y otros [22, 23 y 24], y su principal diferencia es que la identificación de las responsabilidades se hace a partir de las actividades esenciales del modelo funcional, que pueden ser perfectamente determinadas tanto desde los *Casos de Uso Esenciales* [7], el *Análisis Esencial* [8] o también desde los *Casos de Uso* [9]. Sin que puedan establecerse reglas definitivas, los *Casos de Uso* parecen más apropiados para modelar sistemas de gestión, los *Casos de Uso Esenciales* para modelar interfases y el *Análisis Esencial* para sistemas técnicos y embebidos. Se pretende dar así respuesta al modelado de este último tipo de sistemas, de creciente importancia, y todavía muy poco contemplados por las técnicas de modelado tradicionales.

Nótese que, con este enfoque, se recurre a las responsabilidades para establecer un camino bidireccional entre ambos modelos, apoyándose en el postulado básico de que las capacidades de un sistema son intrínsecas, y obviamente independientes de su modo de representación. Expresando este concepto desde ambos paradigmas, puede decirse que: a) La funcionalidad global de un sistema queda completamente definida por sus *Actividades Esenciales* y b) El conjunto de todas las responsabilidades, que están distribuidas en sus clases, representa lo que el sistema es capaz de realizar [26]. Esta tarea de establecer vínculos entre las *Actividades Esenciales* y los objetos se puede facilitar enormemente con un ordenamiento en el que se identifiquen primero las sucesivas responsabilidades asociadas al modelo funcional y se definan progresivamente los objetos que tendrán a su cargo cada una de estas responsabilidades. Simultáneamente, a medida que los objetos son identificados, se procede a determinar la estructura de clases más conveniente recurriendo para ello a las célebres fichas "CRC" de *Clases, Responsabilidades y Colaboraciones*.

Estas fichas CRC fueron originalmente propuestas como una herramienta para la enseñanza del paradigma de objetos por Beck y Cunningham [28] y alcanzaron amplia difusión a partir del libro de Rebecca Wirfs-Brock [25]. Constituyen una valiosa herramienta informal que contribuye al ordenamiento de las clases y a identificar las relaciones entre ellas, ya que un objeto puede cumplir con sus responsabilidades

por sí mismo o solicitando a su vez la asistencia a otro objeto, es decir, solicitando la necesaria colaboración.

Luego, a partir de un diagrama de clases ya consolidado, y de las exigencias temporales del problema, se confeccionan los Diagramas de Secuencias y Diagramas de Estados. Los primeros documentan el comportamiento dinámico del sistema a nivel de objetos y los segundos, en un mayor nivel de detalle, se refieren a las condiciones que deben contemplar sus métodos.

Así, el modelo finalmente obtenido y normalmente denominado Modelo de Análisis, se apoya en las *Actividades Esenciales* para la definición funcional, el *Diagrama de Clases* para la definición estática-estructural y su comportamiento (modelo dinámico) queda representado en los mencionados *Diagramas de Secuencia* y *Diagramas de Estados*. De esta manera se define un procedimiento claramente encuadrable como implosivo, en el que se hace evolucionar el modelo desde los atributos hacia las clases.

Recién ahora, una vez que se ha completado el Modelo de Análisis, queda abierto el camino para proseguir con la fase de diseño, que completará el Diseño Arquitectónico para proseguir con el Diseño Intermedio y finalmente el de Detalle. Aquí, se buscará identificar la mejor solución posible que sea compatible con las posibilidades que ofrece la tecnología y otras restricciones que puedan presentarse.

3.2 Procedimiento Propuesto

El procedimiento que se presenta se limita a cubrir la brecha entre paradigmas, es decir a la identificación de objetos a partir de un modelo funcional. Para su aplicación, es obviamente necesario haber completado el modelo funcional, que estará definido en base a las actividades esenciales. Para ello, habrá que identificar los eventos a los que el sistema debe responder y para ello resulta muy ilustrativo un diagrama de contexto, ya sea que se utilicen los *Casos de Uso Esenciales* [7] o el *Análisis Esencial* [8]. En caso de haberse apoyado el modelo funcional en los *Casos de Uso* [9] o *Escenarios*, habrá que alcanzar mayor nivel de abstracción e identificar las actividades esenciales del sistema. Una vez cumplido este requisito, los pasos a cumplir son los siguientes:

a) Identificar las Actividades Esenciales y reconocerlas como Responsabilidades Esenciales del sistema estudiado.
b) Reconocer al Sistema estudiado como un único objeto, que dispone de un conjunto de métodos que se corresponden con cada una de las Responsabilidades Esenciales antes identificadas. Es decir, este objeto queda representado por el diagrama de contexto del sistema.
c) Encontrar las Responsabilidades Básicas que están involucradas en cada una de las Responsabilidad Esenciales. Para ello, se construye un modelo funcional fragmentado destinado a representar internamente al sistema. La elaboración de este modelo funcional fragmentado se realiza bajo un proceso de perfeccionamiento progresivo, en el que nuevos eventos y actividades pueden ser reconocidos a medida que se los completa [29].
d) Incorporar las Responsabilidades Básicas al único objeto identificado hasta ahora.
e) Ordenar y revisar las Responsabilidades Básicas, a efectos de identificar responsabilidades delegables, comunes o afines, que justifiquen la definición de nuevos objetos que las contengan.
f) Continuar el proceso de identificación de responsabilidades delegables y la justificación de nuevos objetos, hasta que la estructura de responsabilidades y colaboraciones se estabilice. Para ello, resulta muy útil implementar procedimientos que emulen las facilidades que ofrecen las fichas CRC [20].
g) Representar la estructura del sistema en un Diagrama de Clases.
h) Representar la dimensión dinámica del problema mediante *Diagramas de Secuencia* y *Diagramas de Estados*. Para ello, será necesario acudir a la especificación del problema y a su modelo conceptual.

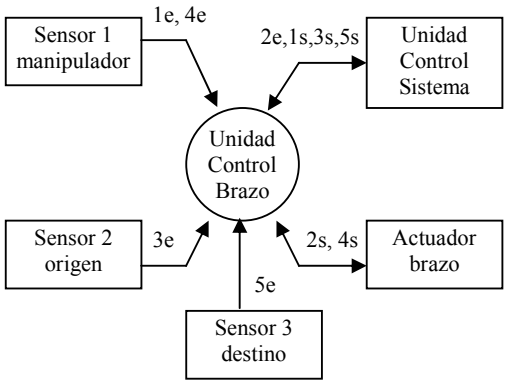
Debe observarse que el procedimiento propuesto incluye, tanto para el modelo funcional como para el modelo de objetos, procesos de ajustes

progresivos que tiene por finalidad el perfeccionamiento gradual de ambos modelos.

Obsérvese además que con este procedimiento se asegura la trazabilidad entre las actividades esenciales y la estructura de clases, que está determinado por el vínculo bidireccional que fue establecido entre ambas. En efecto, las clases tienen asignadas responsabilidades, las que a su vez fueron identificadas a partir de las actividades esenciales y particiones de la memoria esencial.

4. Aplicación a un caso de estudio

Se tomó como caso de estudio la unidad de control del brazo de robot de un centro de mecanizado. Este centro de mecanizado está formado una cinta transportadora de entrada, varias máquinas, otra cinta transportadora de salida, un brazo de robot encargado de manipular las piezas y las correspondientes unidades de control, todas ellas interconectadas. Las operaciones de mecanizado son realizadas según el orden previsto por las fichas de proceso para cada tipo de pieza, las que se identifican con códigos de barras. La coordinación del centro de mecanizado fue objeto de un estudio anterior de Díaz y otros [30] y aquí se centra la atención en el control del brazo, cuyo diagrama de contexto se presenta en la Figura 1.



Referencias: Xe – Estímulo de evento X
Xs – Respuesta a evento X

***Figura 1**- Diagrama de Contexto de la Unidad de Control del brazo de robot.*

En este caso, el diagrama de contexto fue obtenido empleando el Análisis Esencial [8] y a continuación se aplicó el procedimiento presentado hasta arribar al diagrama de clases del sistema. Como resultados del Análisis Esencial se obtuvieron los eventos reconocidos por el sistema y las actividades esenciales asociadas a sus estímulos, los que se presentan en la Tabla 1.

***Tabla 1** – Listado de eventos reconocidos y actividades esenciales previstas*

N	Evento	Actividad esencial
1	Sensor 1 indica manipulador libre	Comunicar a Unidad de Control brazo disponible
2	Unidad de Control ordena traslado de pieza.	Recibir y almacenar origen y destino de pieza. Mover brazo a punto de origen.
3	Sensor 2 indica brazo en origen.	Informar a Unidad de Control brazo en posición.
4	Sensor 1 indica pieza en manipulador.	Sujetar pieza y trasladarla a destino.
5	Sensor 3 indica brazo en destino	Informar a Unidad de Control brazo en posición. Liberar pieza.

Siguiendo el procedimiento propuesto, se reconocieron las Actividades Esenciales como Responsabilidades Esenciales y todas ellas fueron asignadas a un único objeto, que representó la Unidad de Control del brazo (pasos "a" y "b").

Luego se identificaron las Responsabalidades básicas, mostradas en la Tabla 2, que también fueron incorporadas al único objeto hasta aquí definido (pasos "c" y "d").

***Tabla 2** – Listado de Responsabilidades Básicas*

Evento	Responsabilidades Básicas
1	Reconocer señal de sensor 1 (libre)
	Comunicar a UC brazo disponible
2	Reconocer mensaje de Unidad Control
	Registrar en memoria origen y destino
	Reconocer señal de sensor 1 (libre)
	Definir trayectoria y mover brazo
3	Reconocer señal de sensor 2
	Comunicar a UC brazo en origen
4	Reconocer señal de sensor 1 (ocupado)
	Sujetar pieza
	Definir trayectoria y mover brazo
5	Reconocer señal de sensor 3
	Liberar pieza
	Comunicar a UC operación cumplida

Una vez incorporadas al único objeto todas las responsabilidades que pudieron definirse hasta ese momento, se continuó con la tarea de delegarlas e identificar objetos (pasos "e" y "f").

Para ello, se adoptó como norma que en cada ciclo de ajuste se definía un único nuevo objeto, por lo que alcanzar una estructura estable con un total de siete clases demandó otros tantos ajustes. Se enumeran las clases que fueron identificadas (en negrita) con sus métodos, y las relaciones entre ellas se representan en la figura 2.

- **Unidad de Control del brazo**
  - Manipulador libre
  - Recibe orden trasladar pieza
  - Brazo a posición de origen
  - Brazo a posición destino
  - Liberar pieza y operación cumplida
- **Reconocimiento de sensores**
  - Reconoce condición de sensor
- **Interfaz con Unidad de Control Central**
  - Comunica condición a Unidad de Control
  - Reconoce orden de Unidad de Control
- **Actuador sobre piezas**
  - Sujeta y libera pieza
- **Operador del brazo**
  - Almacena datos origen y destino
  - Define trayectoria
  - Mueve brazo
- **Comunicaciones**
  - Recibe mensaje
  - Transmite mensaje
- **Controla movimiento brazo**
  - Verifica posición y corrige rumbo

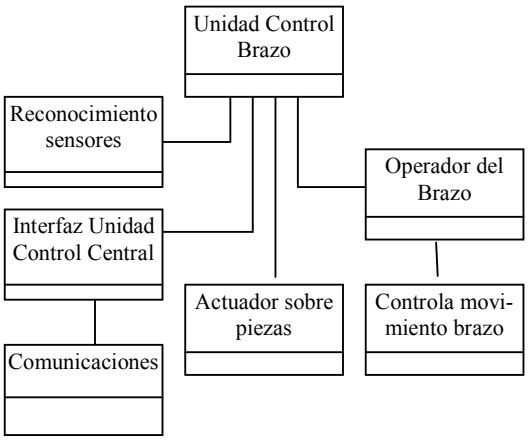


***Figura 2**- Diagrama de Clases resumido.*

Se alcanzó así la estructura de clases para el sistema en estudio a través de un procedimiento sistemático que, en primera instancia, no presenta gran dependencia de la experiencia previa de quién lo aplica, sino más bien de la calidad del modelo funcional que le sirve de base.

Además, al quedar establecido un vínculo claro entre el modelo funcional y la estructura de clases se garantiza una verdadera trazabilidad en ambos sentidos. En efecto, con mucha facilidad pueden establecerse las relaciones entre las *Actividades Esenciales* y sus clases relacionadas, como es el caso del ejemplo representado en la Figura 3.

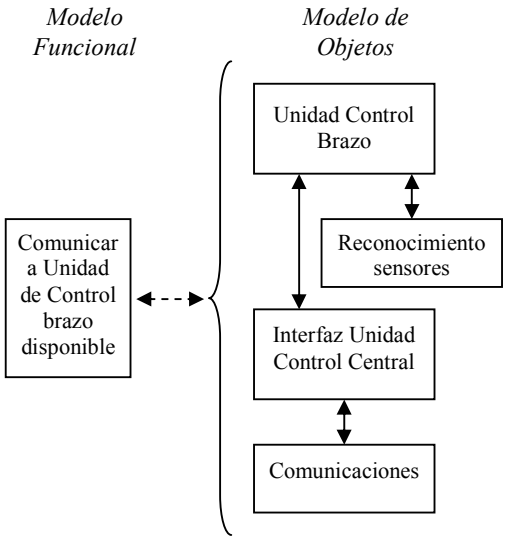


***Figura 3** Trazabilidad entre una Actividad Esencial y las clases relacionadas*

Otro aspecto estudiado fue el esfuerzo que demanda su efectiva aplicación en casos complejos, y se concluyó en la conveniencia de disponer de herramientas de gestión (*CASE*) que faciliten la administración de los recursos y sus relaciones en ambos modelos.

Para terminar, cabe reconocer que todas las ventajas del procedimiento presentado, así como sus debilidades y limitaciones, solo podrán ser determinadas a partir de su aplicación en diferentes casos y condiciones.

5. Conclusiones y Trabajo Futuro

Que a cuarenta años de la identificación de la crisis de software, prestigiosos autores de

libros de gran difusión presenten ideas confusas, y algunas veces hasta erróneas, es un hecho nada auspicioso. Más bien parecería una confirmación de las denuncias de Gibbs [4], quién trece años atrás advertía que la crisis de software se había hecho crónica. Esto da también lugar a plantear un interrogante: ¿si no se ha logrado establecer definitivamente un concepto tan primario, como es el de la definición de objetos, qué margen hay para introducir efectivamente nuevas ideas, tales como son los sistemas autoadaptativos, el Self Healing o la Programación Orientada a Aspectos (AOP)? Menos aún cuando, como en el caso de está última, se vulnera una propiedad hasta ahora considerada esencial en los objetos: el encapsulamiento. Retomando lo ya expresado en la introducción de este trabajo, parecería muy necesaria una profunda evaluación de la situación actual y su proyección futura.

Pasando ahora al procedimiento propuesto para identificar objetos y asegurar la trazabilidad de los modelos, puede decirse que mostró buenos resultados en los casos en que fue aplicado. Los próximos pasos serán: emplearlo en el desarrollo de modelos de sistemas progresivamente más complejos y desarrollar herramientas que faciliten su aplicación.

Además, y tal como ya fue comentado, se espera también confirmar que este método es adecuado para el tratamiento de sistemas de tiempo real y sistemas embebidos, que a pesar de su creciente difusión, son todavía muy poco contemplados por las técnicas tradicionales de modelado.

En relación a ello, se considera que a la hora de la evaluación y comparación de métodos, el requisito excluyente debe ser el aseguramiento de la calidad de los resultados. En efecto, es bien sabido que el costo de subsanar errores en las etapas tempranas del ciclo de vida de un sistema supera ampliamente el costo extra asociado a un correcto modelo de análisis. Además, aquí hay que tener presente que en las ingenierías tradicionales, que la ingeniería de software pretende emular, no se concibe una gran obra que no esté precedida del esfuerzo necesario para asegurar la mejor solución posible.

Cabe finalmente reconocer que el procedimiento presentado es, seguramente, sólo uno más de las numerosísimas propuestas que se han hecho con esta misma finalidad. Es decir, que el problema no estaría en la falta de ideas, sino en que estas ideas trasciendan los ámbitos de investigación y sean efectivamente aplicadas en la actividad cotidiana.

Agradecimientos

A la Magíster Adriana Martín, de la Univ. Nacional del Comahue (Neuquén) y a la Ing. Natalia Mira, del Instituto Universitario Aeronáutico (Córdoba), por sus contribuciones leyendo este trabajo y aportando sus valiosas observaciones.

Referencias

[1] Bauer, F.: "NATO Software Engineering Conference". Report of a conference sponsored by the NATO Science Committee, *Garmisch, Germany*. Edited by P. Naur and B. Randell (1968).

[2] Dijkstra, E.: "The Humble Programmer". ACM Turing Award Lecture (EWD340), published in the Communications of the ACM (1972).

[3] Brooks, F.: "No Silver Bullet, Essence and Accidents of Software Engineering". Computer Magazine. Vol. 20, No. 4, 10-19 (1987).

[4] Gibbs W.: "Software's Chronic Crisis". Trends In Computing. Scientific American, 86 (1994).

[5] Descartes, R.: "El Discurso del Método". (1637).

[6] Yourdon, E.: "Modern Structured Analysis". Prentice-Hall Inc (1993).

[7] Constantine, L.: "What do users want? Engineering Usability into Software". Window Technical Journal (1995).

[8] McMenamin, S. , Palmer, J.: "Essential Systems Analysis". Prentice Hall (1984).

[9] Jacobson, I.: "Object-Oriented Software Engineering: A Use Case Driven Approach". Addison-Wesley. Reading, Mass (1992).

[10] Bustos Reinoso, G.: "Modelado Orientado a Objetos: Una Evaluación Crítica". Revista Facultad de Ingeniería, Nº 10 (Octubre). Facultad de Ingeniería, Pontificia Universidad Católica de Valparaíso, Valparaíso, Chile (1999).

[11] Jacobson, I.: "The Use-Case Construct in Object-Oriented Software Engineering". Cap. 12, J. M. Carroll (ed.). Scenario-Based Design. Wiley, NY. (1995).

[12] Kendall, K., Kendall, J.: "Análisis y Diseño de Sistemas". 6a Edición, Pearson (2005).

[13] Larman, C: "Applying UML and Patterns, an introduction to Object Oriented Analysis and Design and the Unified Process". 2o edition. Prentice Hall (2002).

[14] Fernandez, J., Lilius, J.; "Functional and Object-Oriented Views in Embedded Software

Modeling". Proc. Of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (2004).

[15] Douglass, B.: "Real Time UML". Second Edition. The Component Software Series. Addison-Wesley (2000).

[16] Douglass, B.: "Ropes, Rapid Object-Oriented Process for Embedded Systems". I-Logic Inc (1999).

[17] Sommerville, I.: "Ingeniería de Software". 7ª edición, Pearson, (2005).

[18] Arlow, J., Neustadt, I.: "UML 2". Ed. Anaya (2005).

[19] Abbott, R.: "Program design by Informal English Descriptions". Communications of the ACM. Vol 26, No.11 (1983).

[20] Bellin, D., Suchman, S.: "The CRC Card Book". Addison-Wesley (1997).

[21] Rubin, K., Goldberg, A.: "Getting to Why". Journal of Object-Oriented Programming, Vol. 6, No. 4, July-Aug., pp. 5, 8-10, 13 (1993).

[22] Biddle, R., Noble, J., Tempero, E.: "Essential Use Cases and Responsability in Object-Oriented Development". Proc.of the Australasian Computer Science Conference. Melbourne. Australia (2002).

[23] Biddle, R., Noble, J., Tempero, E.: "Sokoban: a system object case study". Proc.of Technology of Object-Oriented Languajes and Systems. Sydney. Australia (2002).

[24] Biddle, R., Noble, J., Tempero, E.: "From Essential Use Cases to Objects". Constantine & Lockwood, Ltd. Use 2002 Proceedings (2002).

[25] Wirfs-Brock, R., Wilkerson, B., Wiener, L.: "Designing Object-Oriented Software". Prentice Hall (1990).

[26] Giró, J.: "Análisis Esencial Orientado a Objetos, una metodología alternativa para modelar sistemas de tiempo real". ASSE 2005, Jaiio 34. 291-305 (2005).

[27] Page-Jones, M.: "The Synthesis Method". Panel Report: From Events to Objects. Addendum to the Proceedings, OOSPLA'92. Vancouver, Canada. 55-59 (1992).

[28] Beck, K., Cunningham, W.: "A Laboratory for Teaching Object-Oriented Thinking". Proceedings OOPSLA'89, 1-6 (1989).

[29] Ruble, D.: "Practical Analysis & Design for Client / Server & GUI Systems". Prentice-Hall Inc (1997).

[30] Diaz, F., Etchegoyen, J., Giró, J.: "Modelo de operación de un brazo de robot". Curso de Técnicas y Herramientas. Magíster en Ingeniería de Software. Universidad Nacional de La Plata (2002).

Datos de Contacto

Juan Francisco Giró
Departamento de Ingeniería de Sistemas de Información, Facultad Regional Córdoba, Universidad Tecnológica Nacional
juanfgiro@gmail.com