Identificación de Patrones de Reutilización de Requisitos de Sistemas de Información*

A. Durán Toro, A. Ruiz Cortés, R. Corchuelo Gil y M. Toro Bonilla

Departamento de Lenguajes y Sistemas Informáticos, Facultad de Informática y Estadística,
Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, España
{amador,aruiz,corchu,mtoro}@lsi.us.es

Resumen En este artículo se exponen algunos de los resultados de la aplicación de las plantillas y patrones de requisitos presentadas en la edición previa del WER [6]. Uno de los resultados más interesantes de la normalización del formato de los requisitos ha sido la posibilidad de compararlos e identificar patrones de reutilización, tanto a nivel de requisitos de cliente (*requisitos–C*, normalmente expresados en lenguaje natural) como a nivel de requisitos de desarrollador (*requisitos–D*, habitualmente modelos conceptuales), que facilitan el desarrollo y mejoran la calidad de las especificaciones de requisitos.
La relaciones de rastreabilidad entre requisitos–C, requisitos–D, e incluso elementos de menor nivel de abstracción como componentes software, ha permitido también plantear la posibilidad de reutilizar estructuras complejas, desde requisitos–C hasta código, obteniendo así una reutilización *vertical* que abarque distintos niveles de abstracción del desarrollo de software.

Palabras clave: reutilización de requisitos, patrones de requisitos

1 Introducción

El número de propuestas de reutilización en ingeniería de requisitos es aún escaso [11, 12, 13], sobre todo a nivel de requisitos de cliente o *requisitos–C* [17, 2], normalmente expresados en lenguaje natural.

En nuestro caso, la utilización en más de 80 prácticas académicas y en tres proyectos reales aún en desarrollo de las plantillas y patrones lingüísticos para requisitos–C descritos en [6, 7], ha puesto de manifiesto la posibilidad de aplicar técnicas de reutilización durante la fase de ingeniería de requisitos.

Como resultado de esta experiencia se han identificado requisitos que, con pequeñas variaciones, aparecen en un elevado número de desarrollos. A dichos requisitos, una vez generalizados al eliminar los detalles específicos, los hemos denominado *patrones de reutilización de requisitos*, o abreviadamente, *patrones–R*.

La estructura del artículo es la siguiente. En las secciones 2 y 3 se describen algunos de los patrones–R identificados de requisitos–C y D respectivamente. En la sección 4 se comentan algunos trabajos relacionados y, por último, en la sección 5 se exponen las conclusiones y el trabajo por realizar.

* Este trabajo está financiado por el proyecto "MENHIR" de la CICYT. TIC 97–0593–C05–01, Ministerio de Educación y Ciencia de España.

2 Patrones de reutilización de requisitos–C

Las plantillas y patrones–L para requisitos–C descritas en [6] distinguen tres tipos de requisitos:

- **Requisitos de información**: describen qué información debe almacenar el sistema para satisfacer las necesidades de clientes y usuarios. Identifican los conceptos relevantes sobre los que se debe almacenar información y los datos específicos que son de interés.
- **Requisitos funcionales**: describen los casos de uso [1] en los que los diferentes actores utilizan los servicios proporcionados por el sistema. Cada requisito funcional identifica el evento de activación, las pre y postcondiciones y los pasos que componen el caso de uso, así como las posibles excepciones.
- **Requisitos no funcionales**: describen aquellas características no funcionales que los clientes y usuarios desean que tenga el sistema a desarrollar.

En las siguientes secciones se describen algunos de los patrones–R identificados para los dos primeros tipos de requisitos–C, es decir, patrones–R_C.

2.1 Patrones–R_C de requisitos de información

Dentro de los requisitos de información se han identificado varios patrones–R_C. Por un lado se ha cuantificado el número de veces que cada patrón–R_C ha aparecido (ver figura 1), y por otro lado, para cada patrón se ha cuantificado el número de veces que aparece cada dato específico.

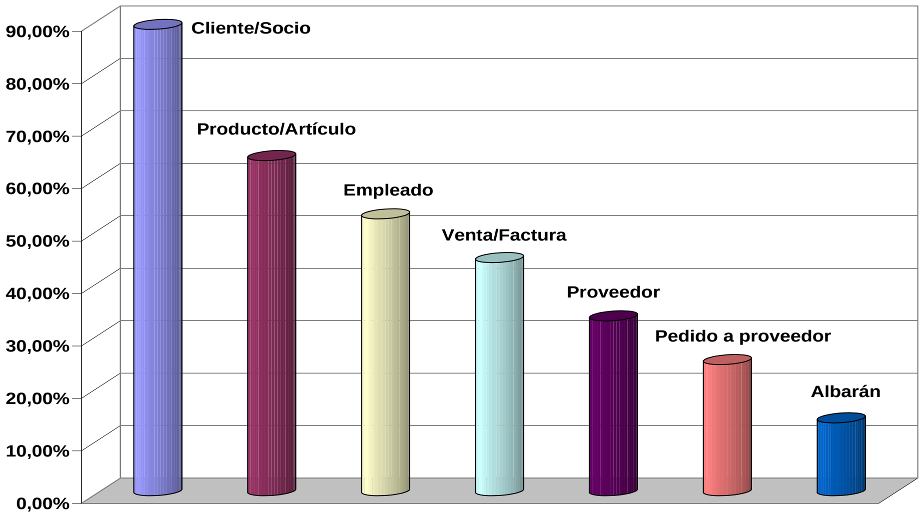

Figura1. Frecuencia de aparición de algunos patrones–R_C de requisitos de información

RI–x	**Información sobre <*clientes/socios*>**
Descripción	El sistema deberá almacenar la información correspondiente a <*los clientes/socios del negocio*>. En concreto:
Datos Específicos	• Nombre y apellidos del <*cliente/socio*> (100%) • Dirección del <*cliente/socio*> (93.75%) • Número de identificación fiscal del <*cliente/socio*> (90.63%) • Números de teléfono del <*cliente/socio*> (84.38%) • Código del <*cliente/socio*> (53.13%) • Fecha de alta/antigüedad del <*cliente/socio*> (34.38%) • Fecha de nacimiento/edad del <*cliente/socio*> (34.38%) • Sexo <*cliente/socio*> (25%) • Número de cuenta bancaria del <*cliente/socio*> (9.38%) • Dirección de correo electrónico del <*cliente/socio*> (6.25%) • Otros datos específicos del <*cliente/socio*> (81.25%)

Figura2. Patrón–R$_C$ *Cliente/Socio*

Siguiendo la clasificación de reutilización descrita en [11] y [13], en la que los requisitos se clasifican como *no reutilizables*, *directamente reutilizables* por composición y *basados en parámetros* para reutilizar por generación, los patrones–R de requisitos de información serían *directamente reutilizables*, aunque probablemente necesitarían adaptaciones específicas en cada caso.

Como ejemplo más significativo, en la figura 2 puede verse el patrón–R$_C$ correspondiente a los requisitos de información relativos a *clientes/socios*, en el que se han ordenado los distintos datos específicos según su frecuencia de aparición.

Todos los patrones–R$_C$ de este tipo de requisitos presentan un porcentaje elevado de casos en el que se incluyen otros datos específicos del sistema que se está especificando que no pueden generalizarse, lo que indica claramente la necesidad de adaptación a cada proyecto concreto. En nuestro caso, se han englobado como *otros* a aquellos datos específicos con una incidencia menor al 5%.

Los patrones–R$_C$ de información identificados responden únicamente a un análisis cuantitativo de su utilización en prácticas académicas. Lo ideal sería combinar estos datos cuantitativos, que pueden ayudar a *descubrir* patrones–R$_C$ que pasan inadvertidos, con información cualitativa proveniente de expertos en dominios de problemas específicos, de forma que se obtengan patrones–R$_C$ certificados de alta calidad que puedan estar a disposición de los ingenieros de requisitos en algún tipo de repositorio.

2.2 Patrones–R$_C$ de requisitos funcionales

En el caso de los requisitos funcionales, dada la dificultad de su análisis cuantitativo debido a su mayor complejidad y heterogeneidad, se ha optado por una análisis cualitativo aunque reforzado cuantitativamente, ya que los 4 patrones–R$_C$ de requisitos funcionales identificados aparecen en todas las especificaciones estudiadas, aunque siempre con detalles específicos en cada caso.

Aplicando la clasificación de [11, 13], estos patrones–R serían requisitos *basados en parámetros*, reutilizables por lo tanto mediante generación al darles valores concretos a los parámetros. En la terminología de Jacobson [10, págs. 126–127], serían *casos de uso parametrizados* o *plantillas de casos de uso*.

Estos cuatro patrones–R$_C$ corresponden a los típicos conceptos de *alta*, *baja*, *modificación* y *consulta* (*CRUD*, *Create*, *Read*, *Update* y *Delete* en inglés [1, pág. 401]) y describen los correspondientes cuatro casos de uso genéricos, de los que los correspondientes a la creación y a la modificación pueden verse en las figuras 3 y 4 y se describen en las siguientes secciones.

Patrón–R$_C$ *Crear/Dar de alta/Registrar* El patrón–R$_C$ *Crear/Dar de alta/Registrar* de la figura 3 describe el caso de uso típico en que un actor proporciona nueva información al sistema como respuesta a una nueva situación en el entorno del que el sistema debe mantener información.

Siguiendo varias recomendaciones para redactar casos de uso de [3, 16], se han evitado referencias específicas a aspectos de implementación o de interfaz de usuario. Más concretamente, se ha utilizado el patrón *Presentación* descrito en [4], que aconseja indicar qué datos debe mostrar o solicitar el sistema sin especificar cómo lo va a hacer.

Las situaciones excepcionales habituales en este tipo de caso de uso suelen ser que el actor intente introducir como nueva una información ya existente o que cancele el proceso durante la introducción de los datos que le solicita el sistema.

RF–x	***{ Crear, Dar de alta, Registrar }*** $<X>$	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando { *un cliente realiza una compra por primera vez, una persona solicita su ingreso como socio, . . .* }	
Precondición	La información correspondiente al nuevo $<X>$ no está almacenada todavía	
Secuencia normal	**Paso**	**Acción**
	1	El $<$*actor*$>$ solicita al sistema comenzar el proceso de *{crear/dar de alta/registrar}* un nuevo $<X>$
	2	El sistema solicita los siguientes datos correspondientes al nuevo $<X>$: $<$*datos solicitados por el sistema*$>$
	3	El $<$*actor*$>$ proporciona los datos requeridos y solicita al sistema que los almacene
	4	El sistema almacena los datos proporcionados e informa al $<$*actor*$>$ de que el proceso ha terminado con éxito
Postcondición	El sistema ha almacenado la información correspondiente al nuevo $<X>$	
Excepciones	**Paso**	**Acción**
	3	Si el sistema detecta que los datos proporcionados ya están almacenados, el sistema informa de la situación al $<$*actor*$>$ permitiéndole modificar los datos proporcionados, a continuación este caso de uso continúa
	3	Si el $<$*actor*$>$ solicita cancelar la operación, el sistema cancela la operación, a continuación este caso de uso termina

Figura3. Patrón–R$_C$ *Crear/Dar de alta/Registrar*

Patrón–R$_C$ *Modificar/Editar* El patrón–R$_C$ correspondiente a la típica modificación o edición de datos puede verse en la figura 4, en el que también se han aplicado los patrones *Especificar* y *Presentación* de [4].

El patrón–R$_C$ indica explícitamente en los pasos 4 y 5 cuáles son los datos que muestra al actor y cuáles de los datos mostrados le deja modificar, ya que habitualmente ambos conjuntos de datos no coinciden.

Una posible adaptación del patrón puede consistir en identificar distintos conjuntos de datos mostrados y datos modificables en función del tipo de actor que esté participando en el caso de uso, permitiendo de esta forma especificar requisitos de seguridad de forma integrada con los requisitos funcionales.

Una posibilidad de excepción en este caso de uso, aparte de la cancelación por parte del actor, es que la información modificada viole alguna restricción.

RF–x	***{ Modificar, Editar }*** <*X*>	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando { *un cliente/socio comunica un cambio de alguno de sus datos, …* }	
Precondición	El sistema tiene almacenada la información correspondiente al <*X*> a { modificar, editar }	
Secuencia normal	**Paso**	**Acción**
	1	El <*actor*> solicita al sistema comenzar el proceso de { *modificar*, *editar* } la información correspondiente a un <*X*>
	2	El sistema solicita que se identifique al <*X*> a { modificar, editar }
	3	El <*actor*> identifica el <*X*> a { modificar, editar }
	4	El sistema muestra los siguientes datos correspondientes al <*X*> a modificar: <*datos mostrados por el sistema*>
	5	El sistema permite al <*actor*> modificar los siguientes datos: <*datos que pueden modificarse*>
	6	El <*actor*> modifica los datos que el sistema le permite y solicita al sistema que los almacene
	7	El sistema modifica los datos correspondientes al <*X*> a modificar e informa al <*actor*> de que el proceso ha terminado con éxito
Postcondición	El sistema ha actualizado la información correspondiente al <*X*> { modificado, modificada }	
Excepciones	**Paso**	**Acción**
	6	Si <*la modificación de la información del* <X> *viola alguna restricción*>, el sistema comunica la situación al usuario y cancela la operación, a continuación este caso de uso termina
	6	Si el <*actor*> solicita cancelar la operación, el sistema cancela la operación, a continuación este caso de uso termina

Figura4. Patrón–R$_C$ *Modificar/Editar*

3 Patrones de reutilización de requisitos–D

También es posible identificar patrones–R en los requisitos orientados a los desarrolladores o *requisitos–D* [17, 2], normalmente expresados mediante modelos conceptuales. Siguiendo las relaciones de rastreabilidad establecidas con los requisitos–C, aquellos requisitos–D que modelen los patrones–R_C serán los patrones–R de requisitos–D correspondientes, abreviadamente *patrones–R_D*.

3.1 Patrones–R_D para requisitos de información

Los patrones–R_D de requisitos de información estarán formados por aquellos tipos de objetos, con sus correspondientes atributos y asociaciones, que modelan la información que describen los patrones–R_C.

Por ejemplo, para modelar el patrón–R_C *Cliente/Socio* se tendría el patrón–R_D formado por el tipo de objetos Cliente/Socio que puede verse en la figura 5 asociado al patrón–R_C citado mediante la correspondiente relación de rastreabilidad.

RI–x	**Información sobre <*clientes/socios*>**
Descripción	El sistema deberá almacenar la información correspondiente a <*los clientes/socios del negocio*>. En concreto:
Datos Específicos	• Nombre y apellidos del <*cliente/socio*> (100%) • Dirección del <*cliente/socio*> (93.75%) • Número de identificación fiscal del <*cliente/socio*> (90.63%) • Números de teléfono del <*cliente/socio*> (84.38%) • Código del <*cliente/socio*> (53.13%) • Fecha de alta/antigüedad del <*cliente/socio*> (34.38%) • Fecha de nacimiento/edad del <*cliente/socio*> (34.38%) • Sexo <*cliente/socio*> (25%) • Número de cuenta bancaria del <*cliente/socio*> (9.38%) • Dirección de correo electrónico del <*cliente/socio*> (6.25%) • Otros datos específicos del <*cliente/socio*> (81.25%)

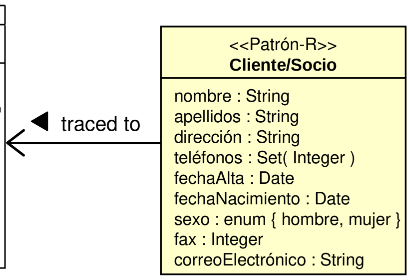

Figura5. Patrón–R_D *Cliente/Socio*

Aunque en el ejemplo la relación entre patrón–R_C y patrón–R_D es 1:1, no siempre tiene porqué ser el caso. Puede que la estructura de rastreabilidad sea más compleja y tenga la forma de una relación $m{:}n$, de forma que un grupo de requisitos–C pueda rastrearse hasta un grupo de requisitos–D.

3.2 Patrones–R_D para requisitos funcionales

Los patrones–R_D para requisitos funcionales están compuestos por descripciones de operaciones del sistema, denominadas *joint actions* en [5], que modelan la funcionalidad descrita en los casos de uso correspondientes a los patrones–R_C de requisitos funcionales.

En estas descripciones, dado un tipo de objetos T, T.new se entenderá como el conjunto de objetos de tipo T que se crean durante la operación, tal como se propone en la versión de OCL [14] descrita en [5].

También se utilizará la palabra reservada respuesta para representar la respuesta que el sistema envía al actor como consecuencia de la realización de ciertas acciones.

Patrón-R *Crear/DarDeAlta/Registrar* El patrón–R_D correspondiente al patrón–R_C *Crear/Dar de alta/Registrar* puede verse en la figura 6.

Este patrón admite diversas variaciones. Por ejemplo, puede que no todos los atributos del objeto que se crea se pasen como parámetros, algunos pueden tomar un determinado valor inicial. También es posible que no se hayan identificado atributos clave para el tipo de objetos en cuestión, en cuyo caso no habría ninguna precondición. Otra posibilidad sería crear más de un objeto a la vez, bien varios de tipo X, bien uno de tipo X y otros de otros tipos que tengan alguna asociación con los objetos del tipo X.

Operación Sistema	**{*Crear, DarDeAlta, Registrar*}<X>**
Descripción	{Crea, Da de alta, Registra} un nuevo <X>
Parámetros	p_1 : $Tipo_1$ ... p_q : $Tipo_q$ k_1 : $Tipo_k$ -- *parámetro clave* ... k_r : $Tipo_r$ -- *parámetro clave*
Precondiciones	**pre**$_1$: No existe un <X> en el sistema con el mismo valor de $k_1, \ldots, k_r$
Precondiciones (OCL)	**pre**$_1$: **not** <X>.allInstances->exists(x | x.k_1 = k_1 **and** ... **and** x.k_r = x.k_r)
Postcondiciones	**post**$_1$: Existe un nuevo <X> cuyos atributos coinciden con los parámetros **post**$_2$: Sólo se ha creado un nuevo <X> **post**$_3$: El sistema informa de que el proceso ha terminado con éxito
Postcondiciones (OCL)	**post**$_1$: <X>.new->exists(x | x.p_i = p_i **and** x.k_j = k_j) -- *i:1..q, j:1..r* **post**$_2$: <X>.new->size = 1 **post**$_3$: ***respuesta*** = "<X> { creado, dado de alta, registrado } con éxito"
Excepciones	¬ **pre**$_1$: El sistema informa de que ya existe un <X> en el sistema con los mismos datos
Excepciones (OCL)	**not pre**$_1$: ***respuesta*** = "Error: <X> ya existente"

Figura6. Patrón–R_D *Crear<X>*

Patrón-R *Modificar/Editar* El patrón–R_D correspondiente al patrón–R_C *Modificar/Editar* puede verse en la figura 7.

Este patrón–R_D también admite varias alternativas. Por ejemplo, crear varias operaciones de sistema para modificar determinada información sobre el objeto, en lugar de tener una única operación para modificar todas las características modificables, o permitir la modificación de cierta información del objeto en función del tipo de usuario que realice la operación.

Operación Sistema	***{Modificar, Editar}*<X>**
Descripción	{Modifica, Edita } un <X>
Parámetros	x : X -- *el <X> a eliminar* p_1 : Tipo_1 ... p_q : Tipo_q
Precondiciones	**pre**$_1$: *<los parámetros no violan ninguna restricción>* **pre**$_2$: *x <cumple las condiciones para poder ser modificado>*
Precondiciones (OCL)	**pre**$_1$**: not** <p_1 ... p_q *violan alguna restricción*>
Postcondiciones	**post**$_1$: Los atributos de *x* coinciden con los parámetros **post**$_2$: El sistema informa de que el proceso ha terminado con éxito
Postcondiciones (OCL)	**post**$_1$: x.p_1 = p_1 **and** ... x.p_q = p_q **post**$_2$: ***respuesta*** = "<X> { modificado, editado } con éxito"
Excepciones	¬ **pre**$_1$: El sistema informa de que los nuevos valores no son aceptables ¬ **pre**$_2$: El sistema informa de que el <X> no puede ser modificado
Excepciones (OCL)	**not pre**$_1$**:** ***respuesta*** = "Error: los nuevos valores no son aceptables" **not pre**$_2$**:** ***respuesta*** = "Error: <X> no puede ser modificado"

Figura7. Patrón–R$_D$ *Modificar<X>*

4 Trabajos relacionados

A parte del ya citado [10], en [15, págs. 222-231] se plantea también la idea de patrones de requisitos, aunque con un formato específico similar al que se usa para los patrones de diseño [9] e incluyendo los requisitos–D asociados, de forma similar a la planteada en este artículo, aunque utilizando técnicas estructuradas basadas en diagramas de flujo de datos.

Sobre la reutilización de requisitos en familias de aplicaciones similares existen trabajos previos como [8], que se basa más en modelos (requisitos–D) que en requisitos en lenguaje natural (requisitos–D).

En [11] y [13] se estudia, dentro del dominio de problemas de los sistemas de planificación de misiones de naves espaciales, la clasificación de requisitos–C en función de su reusabilidad como *no reutilizables*, *reutilizables directamente* (componentes) y *parametrizados* (generadores). Para hacer que los requisitos no reutilizables pasen a una de las otras dos categorías se proponen la tres siguientes heurísticas básicas:

- **Eliminar referencias específicas**: evitar hacer referencias específicas dentro de un requisito. Por ejemplo, si se está desarrollando el sistema X, evitar usar frases como *"El sistema X deberá ... "* y usar en su lugar *"El sistema deberá ... "*, con lo cual podrían reutilizarse requisitos similares en varios desarrollos.
- **Usar términos comunes**: utilizar nombres lo más comunes posibles dentro de un determinado dominio de problemas, relegando las diferencias específicas de cada proyecto a los respectivos glosarios de términos.
- **Separar lo específico de lo genérico**: detectar qué partes de un requisito son generales y reutilizables y qué partes son específicas, y separar éstas últimas en uno o

más requisitos aparte. Una idea parecida al patrón *Especificar* de [4] que se describe a continuación.

En [12] se exponen diez heurísticas generales sobre la introducción de la reutilización en el proceso de ingeniería de requisitos que han influido en nuestros resultados. Básicamente se propone:

- identificar familias de sistemas en los que los requisitos suelan coincidir
- desarrollar requisitos parametrizables abstractos (una idea similar a la presentada en este artículo)
- separar los aspectos específicos de los generales
- intentar identificar patrones de requisitos al trabajar en dominios específicos (es decir, patrones–R)
- intentar reutilizar también los procesos de obtención de ciertos tipos de requisitos, es decir las preguntas a realizar a los clientes y usuarios, las consideraciones a tener en cuenta a la hora de especificarlos, etc.

En la bibliografía consultada también se han hallado patrones de requisitos a un nivel de abstracción más alto. En [4] se proponen los tres siguientes:

- **Patrón Especificar**: este patrón aconseja describir *cómo* puede el usuario de un sistema seleccionar (*especificar*) una determinada información (para modificarla, eliminarla o consultarla) en un requisito separado y hacer referencia a dicho requisito cuando sea necesario.
  Por ejemplo, un requisito *X* podría establecer que "*el sistema deberá permitir al usuario seleccionar a los clientes por su número de DNI, por sus apellidos o por su número de teléfono*" y posteriormente otro que dijera que "*el sistema deberá permitir a los usuarios modificar la información correspondientes a los clientes seleccionados mediante el procedimiento descrito en el requisito X*".
  Abstrayendo y adaptando el patrón a las ideas expuestas en este artículo, podría reformularse para que aconsejara no especificar la forma en que un actor selecciona o identifica una determinada información en los casos de uso en que fuera necesario, sino sacar dicho procedimiento *factor común* en un caso de uso abstracto e *incluirlo* en los casos de uso que fuera necesario (ver figura 8).
  Otra posibilidad, que es la contemplada en los patrones–R_C descritos previamente, es no indicar *cómo* puede el usuario seleccionar o identificar una determinada información, dejando este tipo de detalles para fases posteriores del desarrollo.
- **Patrón Presentación**: este patrón, también aplicado en los patrones–R_C presentados, recomienda limitarse a indicar qué datos debe solicitar o presentar el sistema sin entrar en detalles concretos de interfaz de usuario.
- **Patrón Priorizar**: este otro patrón sugiere que, en el caso de que el usuario desee poder ordenar (*priorizar*) la información presentada por el sistema, se separen las posibles formas de ordenar dicha información en un requisito aparte y se referencie desde los que sea necesario, de forma similar al patrón *Especificar*.

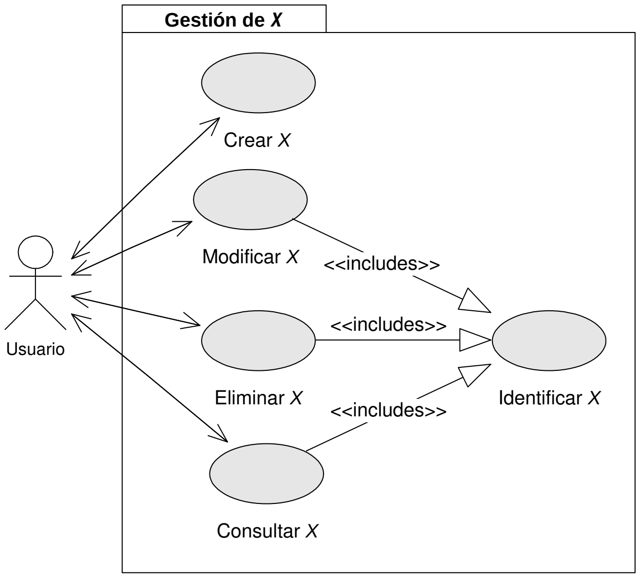

Figura8. Aplicación del patrón *Especificar* en los casos de uso

5 Conclusiones y trabajo futuro

En este artículo se han presentado algunos de los resultados de la normalización de requisitos mediante plantillas y patrones lingüísticos, principalmente los basados en la reutilización de requisitos o grupos de requisitos.

Uno de los riesgos potenciales de los patrones de requisitos es intentar forzar la realidad del problema para que encaje dentro de los patrones identificados, lo que en general es un problema de la reutilización en cualquier fase del desarrollo de software. El ingeniero de requisitos debe utilizar los patrones conociendo este riesgo, y en cualquier caso, consultar con los clientes y usuarios la posibilidad de adaptar el problema a los patrones para reducir los costes del desarrollo.

En un futuro esperamos identificar nuevos patrones–R, sobre todo a partir de la disponibilidad del prototipo de herramienta CASE para ingeniería de requisitos (figuras 9, 10) que se ha desarrollado como parte del proyecto CICYT MENHIR, lo que permitirá el tratamiento automatizado de las especificaciones de requisitos así como el desarrollo de asistentes automatizados que ayuden a aplicar las ideas presentadas en este artículo o la aplicación automática de métricas.

Referencias

[1] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison–Wesley, 1999.

[2] J. W. Brackett. Software Requirements. Curriculum Module SEI–CM–19–1.2, Software Engineering Institute, Carnegie Mellon University, 1990.

[3] A. Cockburn. Structuring Use Cases with Goals. *JOOP*, Sept. y Nov./Dic. 1997.

[4] C. Creel. Requirements by Pattern. *Software Development*, Diciembre 2000. Disponible en `http://www.sdmagazine.com/ breakrm/features/s9912f4.shtml`.

[5] D. F. D'Souza and A. C. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison–Wesley, 1999.

[6] A. Durán, B. Bernárdez, A. Ruiz, and M. Toro. A Requirements Elicitation Approach Based in Templates and Patterns. In *WER'99 Proceedings*, Buenos Aires, 1999.

[7] A. Durán, B. Bernárdez, M. Toro, R. Corchuelo, A. Ruiz, and J. Pérez. Expressing Customer Requirements Using Natural Language Requirements Templates and Patterns. In *IMACS/IEEE CSCC'99 Proceedings*, Atenas, 1999.

[8] A. Finkelstein. Re–use of Formatted Requirements Specifications. *Journal of Software Engineering*, Septiembre 1998.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[10] I. Jacobson, M. Griss, and P. Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success*. Addison–Wesley, 1997.

[11] B. Keepence, M. Mannion, and S. Smith. SMARTRe Requirements: Writing Reusable Requirements. In *Proceedings of the IEEE Symposium on Engineering of Computer–Based Systems*, 1995.

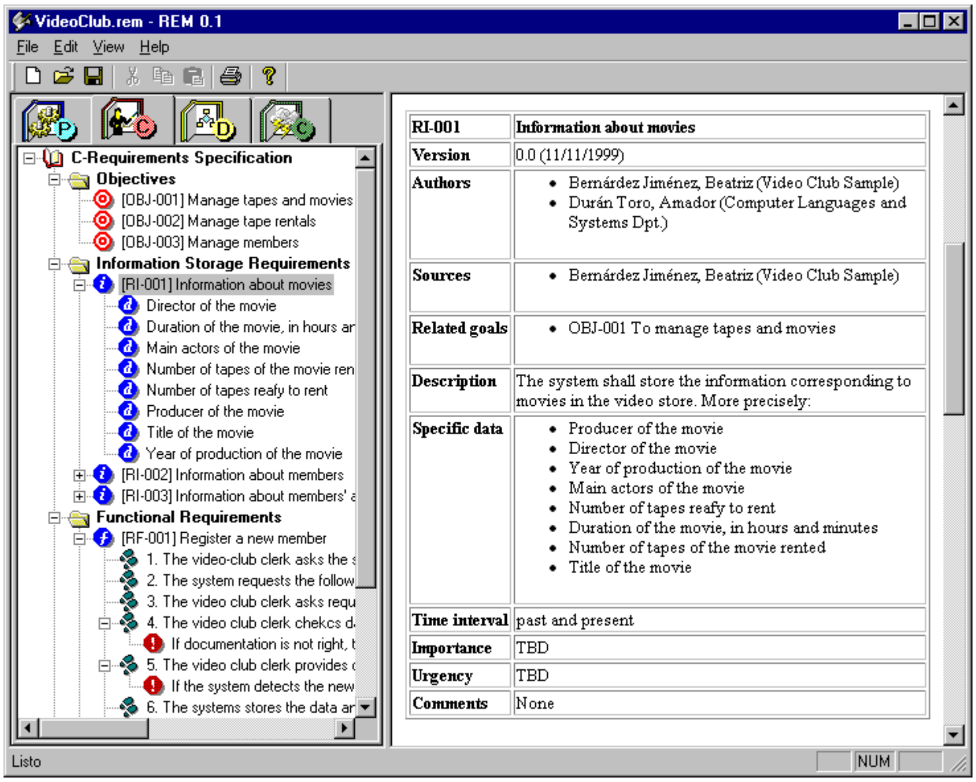

Figura9. Vista de requisitos–C de la herramienta CASE

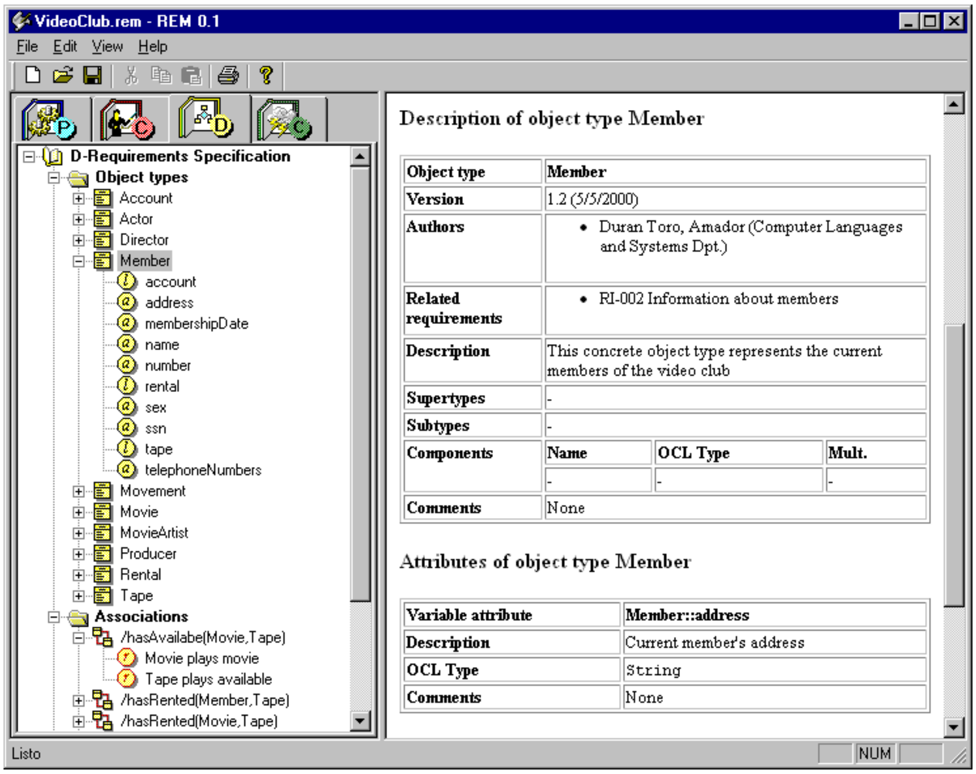

Figura10. Vista de requisitos–D de la herramienta CASE

[12] W. Lam, J. A. McDermid, and A. J. Vickers. Ten Steps Towards Systematic Requirements Reuse. *Requirements Engineering Journal*, 2:102–113, 1997.

[13] M. Mannion, H. Kaindl, and J. Wheadon. Reusing Single System Requirements from Application Family Requirements. In *Proceedings of the International Conference on Software Engineering*, 1999.

[14] Rational Software Corporation. *Object Constraint Language Specification*, 1.1 edition, Septiembre 1997. Disponible en `http://www.rational.com`.

[15] S. Robertson and J. Robertson. *Mastering the Requirement Process*. Addison–Wesley, 1999.

[16] C. Rolland and C. Ben Achour. Guiding the Construction of Textual Use Case Specifications. *Data & Knowledge Engineering Journal*, 25(1–2), 1998.

[17] H. D. Rombach. Software Specifications: A Framework. Curriculum Module SEI–CM–11–2.1, Software Engineering Institute, Carnegie Mellon University, 1990.