Reutilización de Requisitos en el modelo Mecano ⋆

Miguel A. Laguna[1], Francisco J. García[2], Oscar López[1] y José M. Marqués[1]

[1] Universidad de Valladolid
{mlaguna,olopez,jmmc}@infor.uva.es
[2] Universidad de Salamanca
fgarcia@gugu.usal.es

Resumen Los requisitos de un Sistema de Información se presentan de muy diversas formas. Su almacenamiento en un repositorio y las relaciones entre sí y con otros elementos reutilizables pueden condicionar el éxito o fracaso de todo un programa de reutilización. Es necesario introducir un grado mínimo de normalización en los requisitos para que puedan ser realmente reutilizables, lo que se consigue inicialmente mediante la utilización de patrones. Por otro lado, es preciso obtener conjuntos consistentes de requisitos relacionados con elementos reutilizables de otros niveles de abstracción para obtener una interfaz externa de reutilización de elementos más complejos. Por último, se proponen guías para encontrar pautas comunes a distintos elementos y obtener requisitos genéricos adaptables mediante parametrización a múltiples situaciones (lo que implica parametrización de los elementos de diseño o implementación relacionados). En este artículo se avanzan soluciones a estos problemas, basadas en la experiencia con repositorios.

Palabras clave Ingeniería de requisitos, reutilización de requisitos, requisito genérico, escenario, caso de uso, flujo de trabajo, Modelo de Mecano

1 Introducción

En el contexto de la reutilización sistemática del software, el almacenamiento de elementos reutilizables en repositorios es uno de los enfoques que mejores resultados proporciona. La aproximación seguida por el grupo GIRO (Grupo de Investigación en Reutilización y Orientación al Objeto de la Universidad de Valladolid) se encuadra dentro de este enfoque y propone un modelo de reutilización basado en elementos software reutilizables de grano grueso, los *mecanos*, constituidos a su vez como una estructura de elementos de grano fino (*assets*) clasificados en tres niveles de abstracción (análisis, diseño e implementación) e interrelacionados entre sí. Los mecanos construidos durante la fase de desarrollo para reutilización se crean como configuraciones estáticas de *assets*. Sin embargo, se puede dotar de mayor flexibilidad al proceso de reutilización si se ofrece la posibilidad de formar automáticamente un mecano respondiendo a las necesidades del desarrollador con reutilización. Ambas posibilidades se encuentran definidas en el Modelo Mecano [13]. La composición automática se traduce, en parte,

⋆ Este trabajo está financiado por el proyecto ”DOLMEN” de la CICYT, TIC2000-1673-C06-05, Ministerio de Ciencia y Tecnología.

en un problema clásico de clasificación y recuperación de información (figura 1). En este sentido, la información que se aporta a través de los atributos de los *assets* que forman un mecano puede utilizarse directamente en métodos de clasificación basados en texto libre (que utilizan un vocabulario no controlado) o en facetas, introducidas por Prieto-Díaz [33]. El uso de modelos ontológicos como en [2] o el razonamiento basado en casos (Case-Base Reasoning) [14] apuntan nuevas posibilidades de solución al problema fundamental de la búsqueda de elementos individuales en repositorios. Por otro lado, una forma complementaria de clasificación de los mecanos consiste en ofrecer información sobre la funcionalidad global que recogen, es decir, sobre el conjunto de sus requisitos funcionales. Esta agrupación en *assets* de grano grueso tiene como dificultad añadida la enorme diversidad existente de tipos de requisitos funcionales.

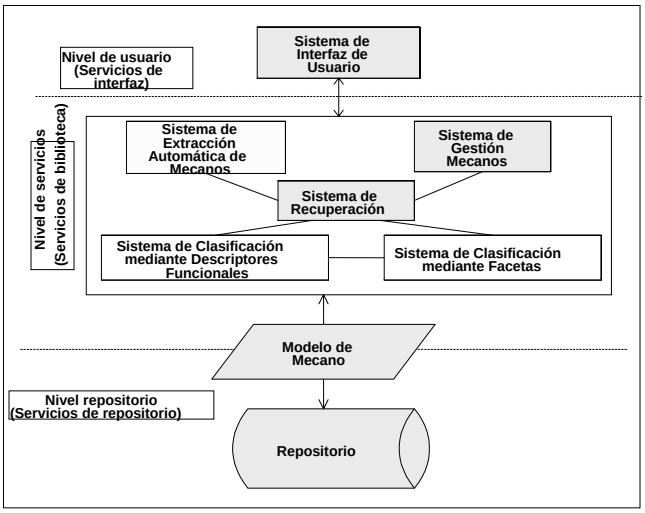


Figura1. Arquitectura del repositorio GIRO

La búsqueda de elementos reutilizables y la composición automática de mecanos es más efectiva si se lleva a cabo partiendo de los requisitos propuestos por los usuarios y localizando en el repositorio aquellos elementos que se ajusten total o parcialmente a esos requisitos. Dicho de otra manera, la puerta de acceso al repositorio para el desarrollador con reutilización debe ser una herramienta que le permita, bien expresar directamente sus requisitos, bien seleccionar por dominios, líneas de producto o palabras clave de entre los cientos de *assets* que puede contener el repositorio. Además, es previsible que se pueda refinar la selección en varias fases: en una primera fase se tienen en cuenta sólo los requisitos funcionales y en fases sucesivas, los requisitos no funcionales. Este planteamiento lleva a la conclusión de que el éxito o fracaso de la reutilización basada en repositorios depende en gran medida de cómo se traten los re-

quisitos, tanto en su fase de definición para reutilización como en su búsqueda en la fase de definición con reutilización.

Los requisitos recuperados del repositorio pueden necesitar una adaptación posterior. En ocasiones, los requisitos se pueden utilizar sin modificación (caso frecuente en requisitos de seguridad, de rendimiento y, en general, requisitos no funcionales). En esta situación un enfoque de ingeniería de dominio, tal como el que propone FODA [20] y sus sucesores [23], es el más adecuado puesto que únicamente se trata de escoger entre varias posibilidades predefinidas. Sin embargo, es mucho más interesante y frecuente el hecho de encontrar requisitos que son parcialmente reutilizables y necesitan algún tipo de adaptación. Por lo tanto, el problema de la reutilización de requisitos se puede desdoblar en dos fases: encontrar un conjunto de requisitos potencialmente reutilizables para un problema dado (fase de comparación) y adaptar los requisitos de modo que sirvan adecuadamente para nuestros propósitos (fase de adaptación).

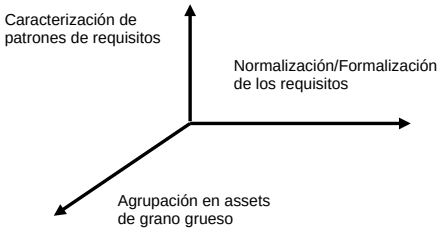


Figura2. Posibles avances en la reutilización de requisitos

Para avanzar en la solución de estos problemas se necesitan avances en varios frentes que se pueden abordar de forma independiente (figura 2). En primer lugar es imprescindible una estandarización de los requisitos capturados y almacenados en el repositorio para facilitar su tratamiento. Esta normalización permite la comparación individual de los *assets* presentes en el repositorio con las necesidades de los usuarios pero, habitualmente, no es suficiente en situaciones reales. Es preciso relacionar los *assets* de requisitos entre sí (además de con otros de distintos niveles de abstracción) para que formen un todo más consistente y al mismo tiempo proporcionen una imagen más completa de la funcionalidad de un determinado mecano. De este modo se dispone de una forma más directa de recuperar y combinar mecanos en el repositorio. Estas consideraciones son aplicables a la primera fase de resolución del problema, la fase de comparación. En cuanto a la fase de adaptación, es posible apoyar a los desarrolladores de forma efectiva, extrapolando la idea de los patrones arquitectónicos [5] o los patrones de diseño [12] a los requisitos. Lógicamente la extensión del concepto de patrón a todos los niveles de abstracción tendrá consecuencias sobre el modelo Mecano, en particular la adopción de mecanos genéricos o mecanos-patrón.

En el resto del artículo se exploran estas posibilidades: La sección 2 trata los distintos tipos de *assets* de requisitos, sus relaciones y los antecedentes sobre su reutilización. El apartado 3 aborda la necesidad de normalizar los requisitos y algunas soluciones que proponemos, incluyendo la transformación de requisitos en otros de distinto tipo. La sección 4 trata de la manera de relacionar y agrupar requisitos para formar *assets* de grano grueso. La sección 5 propone la definición e inclusión en el repositorio de requisitos genéricos con los mismos objetivos que los patrones de diseño, pero en un nivel de abstracción más alto. Además, se explora la posibilidad de instanciar mecanos completos a partir de requisitos genéricos conectados a otros *assets* de análisis, diseño e implementación. Las conclusiones y el trabajo futuro cierran el artículo.

2 Tipología, descripción y estructura de los requisitos

Dentro del conjunto de *assets* que pueblan un repositorio de reutilización, la mayoría de ellos tienen un cierto grado de formalización (código, especificaciones de diseño, especificaciones formales, etc.). Sin embargo, cuando se trata con requisitos es frecuente que éstos estén basados en lenguaje natural, con todos los problemas de ambigüedad, incertidumbre y falta de definición que trae consigo. La descripción y clasificación de requisitos se pueden enfocar de múltiples formas. Cuando se habla de requisitos se hace referencia a elementos muy diversos:

- Requisitos de distinto grado de formalización: lenguaje natural, semiformal, notación formal
- Requisitos declarativos (reglas) o procedurales (escenarios)
- Objetivos y medios para alcanzarlos
- Requisitos de distinta granularidad: desde documento de requisitos, requisitos globales, requisitos funcionales, requisitos atómicos
- Requisitos funcionales (incluyendo requisitos de información) y no funcionales
- Requisitos de usuario y requisitos de desarrollador [4]

En esta línea, Pohl [31] recoge varias definiciones aceptadas de requisitos y establece una clasificación muy detallada de los distintos tipos. En primer lugar diferencia tres grandes bloques: requisitos funcionales, no funcionales y de información y hace una revisión del tratamiento de los requisitos recogidos en 21 documentos de estándares y guías metodológicas. Por otro lado, considera tres dimensiones en la Ingeniería de requisitos: especificación (de menos a más completa), representación (de menos a más formal) y acuerdo (*agreement*, de varias visiones personales a una visión común).

En nuestro contexto, los requisitos que nos interesan son los requisitos de usuario, de tipo funcional, de granularidad fina o intermedia, formato procedural o declarativo y bajo nivel de formalización. La razón de esta elección es que los usuarios finales plantean sus problemas de este modo y la búsqueda en el repositorio de uno o varios mecanos que solucionen esos problemas debe partir de esa base, aunque en fases posteriores se pueden tener en cuanta los requisitos no funcionales. Esta necesidad origina los problemas habituales de la utilización del lenguaje natural: ambigüedad, incertidumbre, parcialidad, etc., por lo que es necesario introducir un cierto grado de normalización en

el tratamiento de estos *assets* de modo que resulten más fácilmente identificables, comparables y relacionables entre sí.

Las aproximaciones más frecuentes desde el punto de vista del usuario final son los escenarios, en sus diversas variantes, y las reglas del negocio. Los escenarios más ampliamente utilizados son los casos de uso inicialmente introducidos por Jacobson [17], popularizados en [18] y actualizados en UML [3]. Sin embargo, es conveniente contemplar otras variantes, en particular a lo que se refiere a los procesos de negocio o *workflows*. Los escenarios están basados habitualmente en el lenguaje natural o en el mejor de los casos, lenguaje estructurado. Por lo tanto, desde el punto de vista de la reutilización es conveniente que este tipo de requisitos sigan algún tipo de norma que permita compararlos para su posterior incorporación a nuevos desarrollos.

Las reglas del negocio [15,30], por su naturaleza declarativa, se pueden describir de una manera bastante formal, mediante expresiones lógicas, fórmulas de cálculo o tablas de decisión. Por otro lado, una regla compleja se puede expresar como una composición de otras más sencillas (si *x*, entonces *y*, donde *x* e *y* a su vez son reglas) y por tanto es relativamente sencillo definir un formato para su introducción en el repositorio.

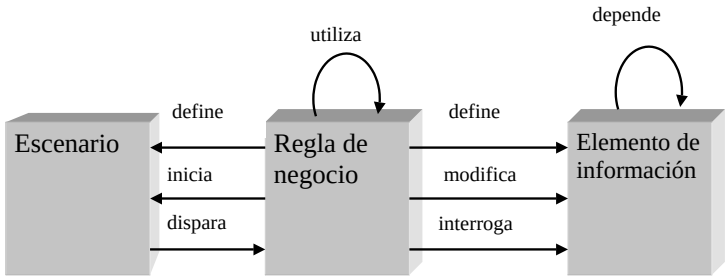


Figura3. Relación entre reglas de negocio y escenarios

Las reglas del negocio pueden presentar distintos niveles de abstracción. Desde el punto de vista de la reutilización, lo que más nos interesa son las reglas de gestión (el nivel de abstracción más alto). Desde el punto de vista formal, Loucopoulos [25] diferencia tres tipos de reglas:

- Restricciones estáticas y de transición: Ambos tipos de restricciones representan reglas que deben ser respetadas siempre (la aserción debe ser siempre cierta)
- Reglas de derivación estáticas y de transición: Sirven para definir información en términos de otras primitivas
- Reglas de acción: Invocan transacciones si se cumple su precondición

Las reglas estáticas pueden incorporarse en diagramas estructurales (por ejemplo, utilizando un lenguaje de restricciones como OCL [40] en un diagrama de clases). Por

su parte las reglas dinámicas y de acción se pueden encontrar en forma de disparadores en los diseños de bases de datos o en forma también de restricciones en modelos dinámicos. Aunque esta forma de expresar las reglas permite un alto grado de formalización, el hecho de estar incrustadas en otros requisitos dificulta su reutilización independiente.

En cualquier caso, se puede poner de manifiesto la relación entre reglas y escenarios entre sí y con los requisitos de información. La figura 3 representa interacciones entre el sistema y sus usuarios. Los escenarios representan una manera natural de representar esas interacciones. El sistema conecta entradas (estímulos) y salidas (respuestas). Las respuestas a una entrada válida se construyen en función de la entrada, del estado del sistema y de las reglas de negocio. Las reglas de negocio relacionan escenarios con el estado del sistema y se deben tener en cuenta al definir los *assets* de grano grueso y las relaciones entre los requisitos que los constituyen.

Desde el punto de vista estructural, Durán [9] descompone los escenarios (como casos de uso), además de los requisitos de información y no funcionales en sus partes elementales. Esta posibilidad de descomponer un requisito en sus partes atómicas (por ejemplo, un paso en un escenario, un elemento de información o una condición de una regla de negocio) es fundamental, no sólo para su almacenamiento en el repositorio de forma controlada sino también para intercambiar o incluso generar automáticamente requisitos en distintos formatos compatibles entre distintas herramientas.

En cuanto a la reutilización de requisitos, destacados autores [11,32] mantienen que su introducción el proceso de desarrollo lo mejora substancialmente. Sin embargo, ya que los requisitos representan la forma principal de comunicación con los clientes, su formato no es adecuado para su representación en un ordenador y mucho menos para su reutilización de forma simple y directa. Dado que las técnicas simples de clasificación y búsqueda no resultan demasiado efectivas, los métodos más prometedores incluyen entornos de desarrollo y herramientas CASE [32] o técnicas de representación del conocimiento, como las propuestas por Lowry [26]. Cybulski [7] ha revisado las distintas aproximaciones empleadas. En trabajos más recientes, Sutcliffe y Maiden [39] proponen el reconocimiento de patrones mediante analogía como solución al problema de comparación de escenarios.

Todas estas técnicas de reutilización hacen hincapié en la semántica de requisitos obtenidos de forma independiente. Como alternativa, la Ingeniería de Dominio propone la creación de requisitos reutilizables desde el principio. FODA [20] aparece como la técnica más representativa de esta tendencia. Los métodos más actuales, como ODM [38] o FeatureRSEB [16] se inspiran directa o indirectamente en esta técnica. FORM [23] representa la evolución de FODA hacia el paradigma de la orientación a objetos.

En las siguientes secciones se expone la experiencia en reutilización de requisitos y su problemática, siguiendo la aproximación propuesta en el modelo Mecano [13]. Este modelo incorpora aspectos de los esquemas clásicos de reutilización basada en repositorios, mejorados con la introducción del concepto de mecano como elemento reutilizable que incorpora *assets* de varios niveles de abstracción, y de la Ingeniería de Dominio, al permitir variaciones en los requisitos. Estas variaciones se registran mediante la definición de conjuntos de requisitos con distintos tipos de relaciones entre sí, también definidas en el modelo Mecano.

3 Normalización de los requisitos funcionales

Para las descripciones de requisitos funcionales introducidos el repositorio GIRO se ha optado por utilizar un formato basado en plantillas. La decisión para utilizar este formato de representación se justifica en los siguientes puntos:

- El lenguaje natural es la base de comunicación en la fase de elicitación de requisitos, lo cual lleva a que la forma más normal de documentar éstos sea utilizar algún mecanismo basado en lenguaje natural
- Estas plantillas se limitan a documentar requisitos, con independencia del paradigma de desarrollo que se haya seguido
- Este formato de plantilla se adecúa muy bien a su presentación en páginas HTML, lo cual facilita en gran medida la creación de sistemas de navegación de mecanos vía hipermedia
- Su estructura tabular también se presta a buscar formatos de intercambio y almacenamiento en el repositorio, de forma que no sólo sea fácil su presentación sino también su tratamiento. En este sentido XML (*eXtensible Markup Language*) parece el medio ideal para almacenar estas descripciones funcionales, siendo además extremadamente sencillo definir el DTD (*Documention Type Definition*) correspondiente a la gramática de las etiquetas para describir las entradas de la plantilla

Nuestra experiencia inicial se ha basado concretamente en las plantillas y patrones lingüísticos propuestos por Durán [9], que se pueden utilizar tanto durante las reuniones de elicitación con clientes y usuarios como para registrar y gestionar los requisitos almacenados en el repositorio. Como fruto de la experiencia de su utilización, para algunos campos de las plantillas se han identificado frases estándar que son habituales en las especificaciones de requisitos. Estas frases, denominadas por su autor *patrones lingüísticos*, deben completarse con la información oportuna. Los plantillas propuestas son las siguientes:

- Plantilla de requisitos de información
- Plantilla de requisitos funcionales
- Plantilla de requisitos no funcionales
- Plantilla de objetivos
- Plantilla de actores
- Plantilla de conflictos

La estructuración de la información en forma de plantilla y la propuesta de frases "estándar" facilitan la redacción de los requisitos, guiando a los desarrolladores para reutilización en la alimentación del repositorio. Para los propósitos de este artículo, las plantillas más interesantes son las relacionadas con los requisitos funcionales, requisitos de información y los objetivos. Los objetivos del sistema pueden considerarse como *requisitos de alto nivel* [34], de forma que los requisitos propiamente dichos serían la forma de alcanzar los objetivos.

Los dominios abordados han sido Gestión universitaria, Herramientas para discapacitados y Tratamiento digital de imágenes. En [13] se detallan la experiencia y las conclusiones obtenidas durante la introducción de los correspondientes *assets* en el Repositorio GIRO. Otros grupos han utilizado dicho repositorio para introducir sus propios requisitos (en concreto requisitos sobre seguridad de la información, recuperados y utilizados sin modificación en varios proyectos industriales). Inicialmente se utilizó una herramienta ya existente, EUROWARE [35], adaptada parcialmente al modelo Mecano

para la implementación del repositorio. En la actualidad, se utiliza un repositorio propio, desarrollado sobre un gestor de bases de datos convencional, que implementa el modelo completo (http://giro.infor.uva.es).

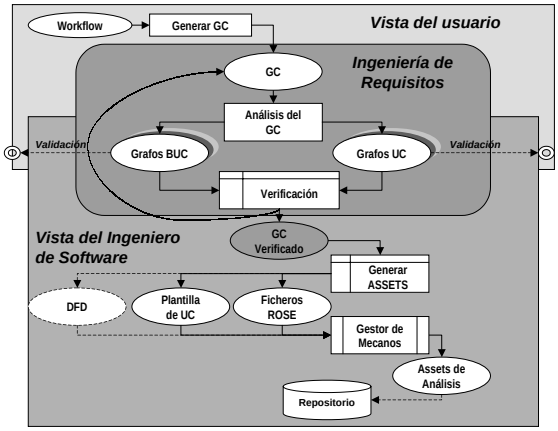


Figura4. Marco general de trabajo para la normalización de la captura de los requisitos de usuario

De estas experiencias iniciales, se ha hecho patente la necesidad de algún tipo de normalización adicional, más allá de la utilización de plantillas. En [21] se propone determinar la funcionalidad inicial del sistema software a partir de la descripción de la forma actual de trabajo del futuro usuario. Una de las líneas de investigación del grupo GIRO está orientada a desarrollar dicha propuesta mediante una herramienta para la determinación de los requisitos funcionales de un sistema de información y su almacenamiento en forma de *assets* de análisis. Como punto de partida se utilizan flujos de trabajo (emphworkflows) modelados con redes de Petri, lo que permitirá la generación automática de casos de uso y de otros elementos reutilizables de nivel de análisis (*assets* de análisis) que pueden ser incluidos en un repositorio para formar mecanos. De esta manera, las bondades de los casos de uso se refuerzan mediante el uso de *workflows* lo que permite la escalabilidad y la trazabilidad. Al mismo tiempo se corrige la informalidad de los casos de uso mediante un formalismo robusto proporcionado por las redes de Petri. En síntesis, se propone una alternativa para la normalización del proceso de captura de requisitos funcionales para generar casos de uso incluidos en mecanos.

La figura 4 muestra el marco de referencia utilizado para la normalización de requisitos. Se establecen dos niveles relacionados en el proceso de elicitación de los mismos: El nivel del usuario y el nivel del ingeniero del software. El primero tiene una visión externa del sistema como caja negra, y el segundo corresponde al interior de dicha ca-

ja. En el nivel del ingeniero del software, existe una vista del ingeniero de requisitos que actúa como una interfaz entre los dos niveles anteriores. El usuario proporciona la primera aproximación a los requisitos funcionales del sistema mediante la definición de *workflows* documentales. Se utiliza un tipo específico de *workflow*, el diagrama Documentos-Tareas [21] que, utilizado rigurosamente, cumple con los estándares definidos por la *Workflow Management Coalition (WfMC)* [41]. En esta etapa, la propuesta metodológica proporciona una definición preliminar de los requisitos de usuario y a partir de ella se modela la funcionalidad del sistema a través de un grafo de casos (GC). Mediante el análisis del grafo de casos se obtendrán familias de casos de uso del negocio (BUC) y de casos de uso (UC). Finalmente, los *assets* generados de este modo están listos para ser utilizados en el desarrollo en el contexto de la reutilización del software. En consecuencia, el marco general de la solución planteada conduce a *assets* de análisis aptos para ser asociados, mediante la interfaz de gestión del repositorio, con los *assets* de diseño e implementación correspondientes para formar los mecanos requeridos. En [27] de desarrolla esta propuesta con detalle utilizando un formalismo basado en redes de Petri y se aplica al caso real de una empresa eléctrica. La obtención automática de los *assets* de requisitos garantiza la normalización de los mismos en el repositorio.

Por otro lado, una segunda línea de trabajo aborda los problemas de transformación de requisitos de un tipo en otro. El objetivo es encontrar una representación común para todos los requisitos que implican interacción del usuario (escenarios y flujos de trabajo, fundamentalmente). Distintas visiones han llevado a una explosión de técnicas que tienen mucho en común:

- Los ingenieros informáticos manejan, sobre todo, casos de uso (y, en general, escenarios) para representar interacciones usuario-sistema
- Los técnicos en organización de empresas utilizan herramientas para la reingeniería de procesos de negocio (BPR)
- A su vez, partiendo de los casos de uso, se proponen extensiones (los casos de uso del negocio o BUC) para representar los procesos de negocio y permitir su estudio y mejora
- Los gestores de flujos de trabajo se imponen en las grandes empresas como soportes del trabajo cooperativo y de seguimiento y automatización de procesos complejos
- La *WfMC* [41] propone un marco estándar para la definición de modelos de flujo de trabajo y su implantación
- UML deriva un diagrama de estados especializado (el diagrama de actividades) para, precisamente, representar los procesos de negocio [3]

La situación actual nos lleva a destacar dos niveles extremos, el nivel de negocio global, donde encajarían la BPR, los flujos de trabajo o los BUC y el nivel de interacción usuario-computadora, donde se aplican los escenarios o casos de uso. Además, las actividades de un flujo de trabajo se pueden ver como casos de uso, de modo que se relacionen las dos técnicas. La hipótesis de trabajo es que todas estas técnicas son esencialmente idénticas y se pueden tratar de forma homogénea de modo que una actividad dentro de un flujo de trabajo se refine como otro *workflow*, de forma recursiva, hasta llegar a una actividad elemental que representa una interacción simple entre un actor y un sistema.

Lee [24] mantiene que un caso de uso puede convertirse en una red de Petri y en sentido contrario, un *workflow* se puede construir internamente como una red de Petri [1]. Más arriba se ha mostrado cómo utilizar un *workflow* para generar automáticamente casos de uso. En esta propuesta se cuestiona la necesidad de utilizar distintas herramientas para lo que, en el fondo, es esencialmente lo mismo: mostrar una interacción entre un sistema que solicita un servicio y otro sistema que lo proporciona. El desarrollo de un Sistema de Información se visualiza mediante una imagen de cajas que contienen otras cajas que se van abriendo a medida que se avanza en el proceso de elicitación y análisis de requisitos. En el trabajo pendiente se incluye la definición de un modelo que represente la relación entre los distintos tipos de requisitos funcionales a través de sus componentes elementales comunes, definiendo un lenguaje unificador que integre las distintas terminologías. Definido este modelo, se utilizará un formato común para representar internamente en el repositorio los distintos tipos de requisitos funcionales y facilitar su comparación.

4 Requisitos de grano grueso: descriptores funcionales

En la sección anterior se han mencionado los problemas de búsqueda de elementos reutilizables concretos en una biblioteca de reutilización. Sin embargo, para el modelo de reutilización basado en mecanos, no solamente es importante un requisito individual sino, sobre todo, el conjunto de requisitos para los cuales un mecano representa una solución completa desde el análisis hasta la implementación. Esta situación plantea el problema de cómo relacionar y agrupar los requisitos funcionales.

En trabajos de otros autores se pueden encontrar algunas aproximaciones al problema como las redes de representación de conceptos [6], bases de datos enlazadas mediante *plug-ins* a un procesador de textos con el que se modifica el documento de requisitos del software [36] o las fachadas que recogen la información considerada relevante de un elemento software reutilizable [19]. Los mecanismos de variabilidad relacionados con las fachadas utilizan las relaciones de *include* y *extend* entre casos de uso y pueden ser implementadas mediante las relaciones usa y extiende del modelo Mecano. En [29] se propone una clasificación de los requisitos en familias y considera un sistema de selección de requisitos basado en una estructura en forma de retículo. Por otro lado, como se ha mencionado en la sección 2 se pueden establecer relaciones entre las reglas de negocio y los escenarios: un usuario inicia un caso de uso que dispara una regla de negocio que utiliza elementos de información [8]. Esto implica que si se introducen requisitos de varios tipos en el repositorio, se pueden definir las relaciones entre los mismos en esos términos (dispara, inicia, etc., todas ellas definibles en términos de las relaciones intra-nivel del modelo Mecano).

En el modelo de Mecano, el problema se aborda mediante descriptores funcionales. Un descriptor funcional es un conjunto de enlaces a aquellos requisitos funcionales que el desarrollador para reutilización ha querido destacar, de forma que cada una de estos requisitos es un *asset* componente del mecano. No es obligatoria la presencia de un descriptor funcional en todo mecano, pero un mecano puede tener varios, a manera de múltiples vistas funcionales de un mismo mecano. A su vez, varios mecanos pueden compartir el mismo descriptor funcional [13].

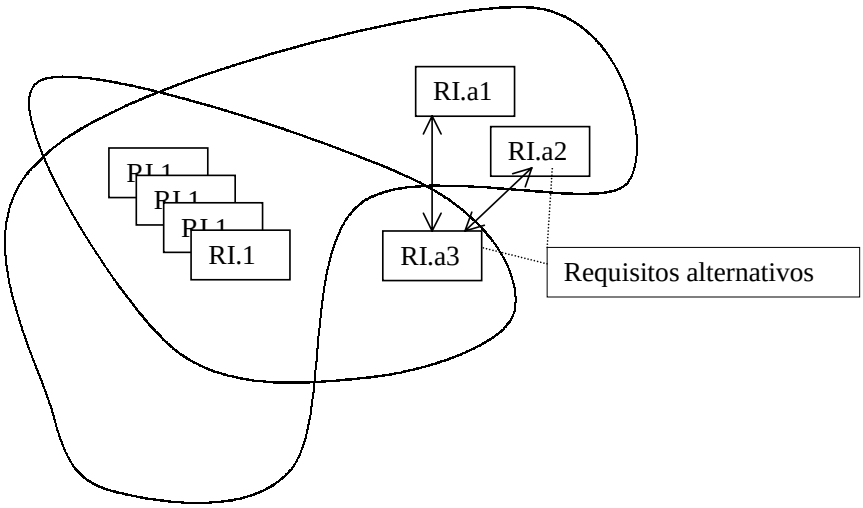


Figura5. Requisitos de grano grueso con alternativas

Ahora bien, además de la evidente relación de agregación entre un descriptor funcional y cada *asset* que lo compone, entre estos últimos deben establecerse explícitamente relaciones de otros tipos que permitan seleccionar entre requisitos alternativos pero incompatibles o añadir opcionalmente requisitos compatibles entre sí. Por otro lado, existirán relaciones de dependencia entre requisitos funcionales y diagramas de clases o entre requisitos de información y modelos de datos, etc. Este tipo de relaciones que se utilizan en el Repositorio GIRO tienen un antecedente en las técnicas de análisis de dominio. El método utilizado en FODA [20], ya mencionado anteriormente y FORM [23], que representa la evolución hacia el paradigma de Orientación a Objetos, propone tres tipos de características (*opcionales*, *obligatorias* y *alternativas*) y tres tipos de relaciones entre ellas (*compuesto-de*, *generalización* e *implementado-por*). El tipo de relación *implementado-por* es realmente una dependencia entre requisitos de modo que la implementación de un requisito implica la implementación necesaria del segundo.

En el modelo Mecano, las relaciones estructurales intra-nivel que podemos utilizar de forma equivalente son agregación (o composición), extensión y uso. Se dispone además de la asociación como tipo de relación bi-direccional.

Con estas relaciones pueden enlazarse los *assets* que describen requisitos atómicos a semejanza de los grafos de características (*feature graphs*) propios de FODA o FORM, agrupándose mediante agregación para predefinir distintos descriptores funcionales. Estos descriptores funcionales a su vez se enlazan con otros *assets* de análisis, diseño, etc. formando mecanos que suponen variaciones sobre un modelo básico (figura 5).

En el repositorio GIRO se pueden encontrar mecanos construidos mediante estas técnicas. La aplicación más inmediata del modelo Mecano con variantes se encuentra

en el desarrollo de líneas de producto. En este sentido, una vez experimentado en proyectos de utilidad académica, nos encontramos en la fase de aplicación a casos reales. El primero de ellos se está desarrollando en la Consejería de Agricultura y Ganadería de la Junta de Castilla y León para el control de subvenciones.

5 Requisitos genéricos

Sobre la reutilización de requisitos en familias de aplicaciones similares existen trabajos previos como los de Finkelstein [10], que se basa más en modelos que en requisitos de usuario. Lam [22] se basa en los trabajos de analogías y de análisis de dominios para desarrollar escenarios de dominio como una forma particular de abstracción del problema mediante compartimientos estancos. Plantea que la reutilización de escenarios es útil en aplicaciones que pertenecen a un mismo dominio. Por ejemplo, los escenarios de un sistema de bibliotecas (préstamo de libros, retornar libros, etc) son comparables mediante analogía con los encontrados en un sistema de alquiler de vídeos (préstamo de vídeos, retornar vídeos, etc). Ambas aplicaciones pertenecen al dominio de gestión de recursos y comparten escenarios similares. La adaptación de los escenarios se realiza mediante *walkthroughs* y discusiones con los clientes. Sutcliffe y otros [39] proponen modelos genéricos de aplicaciones y, relacionados con ellos, bibliotecas de casos de uso genéricos que sirven para comparar mediante generación de escenarios los casos de uso almacenados con los nuevos que se elicitan con los usuarios. Utilizan una estrategia basada en analogía para la reutilización de requisitos. La propuesta de comparación se basa en un proceso de *matching*, basado en lógica de primer orden y lógica temporal.

Como se ha mostrado en apartados anteriores, nuestras propuestas de reutilización a partir de los requisitos se basan en los escenarios como representación de la funcionalidad. En este apartado, siguiendo las ideas de los autores mencionados aunque con distinto enfoque, se presenta una aproximación basada en escenarios ganéricos como alternativa a los escenarios elicitados para cada sistema concreto. La figura 6 muestra el modelo para la reutilización a partir de los requisitos funcionales (posiblemente generados de forma automática a partir de *workflows*). Los escenarios deben ser generalizados, en el sentido de conseguir requisitos genéricos, para acceder a nivel de abstracción más alto. Los escenarios genéricos obtenidos pasan a la fase de comparación con los escenarios recuperados del repositorio. A partir de esta comparación se debe decidir si los escenarios recuperados (modelos genéricos) son adaptables al problema para generar los mecanos correspondientes en tiempo de reutilización. Estos mecanos instanciados (estructuras complejas formadas por elementos de análisis, diseño e implementación) serán la base para el desarrollo de la aplicación final. La adaptación puede dar lugar a nuevas versiones que retroalimentan al repositorio. Las tareas básicas para el proceso de reutilización de requisitos genéricos son cuatro:

- Representar los escenarios del problema
- Generalizar los escenarios del problema
- Comparar los escenarios generalizados con los existentes en el repositorio
- Adaptar los elementos seleccionados/generados para que satisfagan los requisitos

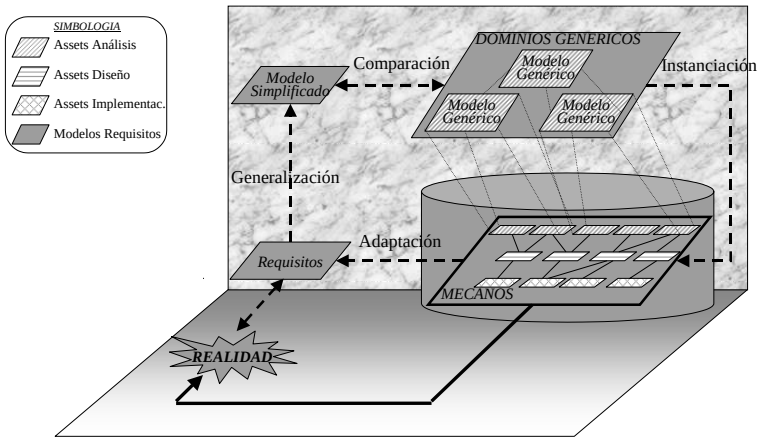


Figura6. Requisitos genéricos como clave de la reutilización de mecanos

El problema de la representación ha sido tratado en la sección 3. La adaptación de los elementos reutilizables relacionados será tema de trabajos futuros. La generalización y la comparación constituyen el tema central de este apartado, en el que proponemos la forma de abordar estas labores. El modelo del problema debe ser simplificado para recoger la funcionalidad esencial y los modelos genéricos de dominios se extraen del repositorio. El modelo simplificado del problema se debe comparar con el modelo genérico del dominio mediante algún tipo de analogía, basada en redes de Petri.

La generalización de escenarios se basa en el análisis por reducción de redes de Petri [37]. Las técnicas de reducción permiten eliminar o sustituir transiciones y/o lugares con base en propiedades específicas, como vivacidad o limitación, de modo que el sistema se simplifique, preservando las propiedades del comportamiento. El análisis por reducción se realiza mediante los siguientes pasos sucesivos: eliminar lugares implícitos, realizar fusión de lugares equivalentes y sustituir lugares por grupos de acciones. Desde el punto de vista conceptual, la reducción permite agrupar secuencias de operaciones en el sistema y sustituirlas por macro-acciones en un modelo reducido. La figura 7 muestra un ejemplo tomado de [28].

La comparación de escenarios se debe realizar con base en algún elemento que sea común y significativo entre los modelos. Las metas de los actores, incluidas de forma explícita en los modelos expresados como redes de Petri, pueden ser ese elemento de comparación. De esta manera se puede obtener alguna conclusión sobre la analogía entre dos modelos en términos de la satisfacción de metas. A partir del modelo generalizado del problema, la búsqueda de analogías entre escenarios puede realizarse

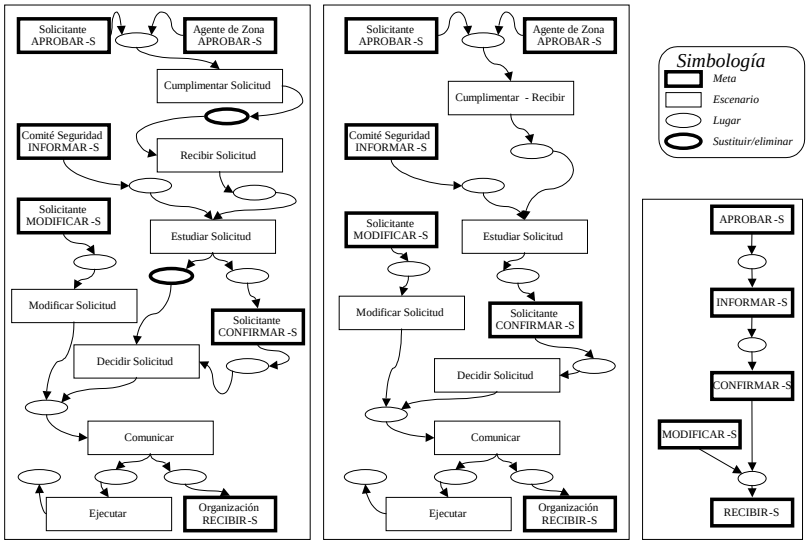


Figura7. Reducción de la red de Petri de un sistema de solicitudes

seleccionando uno o varios modelos genéricos del repositorio con un potencial básico de reutilización para el problema. Un modelo genérico es potencialmente reutilizable en un problema si se cumplen dos condiciones:

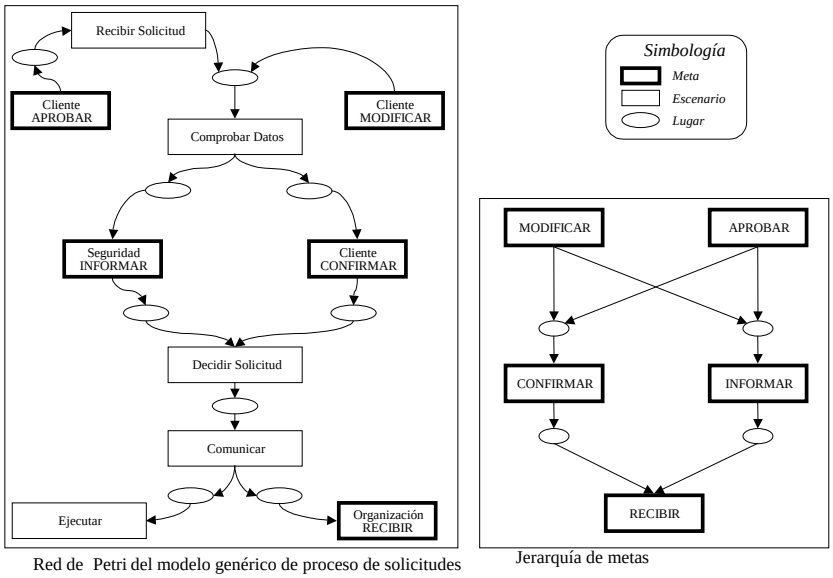
- Las metas raíz y las metas hoja de la jerarquía de metas del problema están incluidas (inclusión de nodos) en la jerarquía de metas del modelo genérico, lo que requiere la utilización de un lenguaje común
- La estructura interna de las metas en el modelo genérico (la jerarquía de metas sin considerar los nodos raíz) satisface el logro de las metas hoja del problema. Esta condición asegura que en el dominio existe al menos un camino lógico para satisfacer las metas finales del problema

Una vez encontrado un modelo genérico con potencial de reutilización, se deben contrastar los escenarios genéricos. Se deben buscar escenarios del modelo genérico que sean aplicables al problema. Un escenario genérico es aplicable al problema si:

- Parte de un nodo raíz del modelo genérico y satisface una meta hoja del modelo genérico
- Satisface una meta raíz del problema y una meta hoja del problema

Las secuencias de transiciones genéricas que satisfacen las metas raíz del problema y las metas hoja del problema se consideran escenarios análogos a los correspondientes del problema. Para probar esa analogía, se pueden expresar las secuencias de transiciones correspondientes a cada meta raíz, tanto en los escenarios análogos como en el modelo del problema, mediante disyunciones y conjunciones lógicas. Sin embargo, la

aplicabilidad real sólo es decidible en términos de la semántica del problema. Esto indica que el proceso general de reutilización desde los requisitos debe pasar por la etapa de la adaptación de escenarios. La determinación del potencial de reutilización del modelo genérico para el problema se puede realizar mediante una prueba automática de teoremas. La aplicabilidad de los escenarios genéricos en el problema es expresable en términos lógicos, aunque decidible en relación con la semántica del problema. En [28] se detalla el alcance de esta propuesta y se aplica a un caso práctico. La figura 8 muestra uno de los modelos genéricos utilizados para la comparación.



Red de Petri del modelo genérico de proceso de solicitudes

Jerarquía de metas

Figura8. Un modelo genérico para el dominio de solicitudes

El proceso de comparación toma como punto de partida las metas del sistema modelado, lo que requiere una sintaxis común para expresar las metas del problema y del dominio. No obstante esta debilidad, nuestro planteamiento presenta la ventaja de ser automatizable. A diferencia de otras aproximaciones de comparación de requisitos que utilizan procesos complejos e interactivos de reconocimiento de patrones, con heurísticas incorporadas, nosotros proponemos un proceso automatizable e incorporable al modelo Mecano. Por último, el tratamiento de los requisitos genéricos en el repositorio debe ser diferente del tratamiento de los requisitos simples. Si estos últimos se relacionaban con *assets* de diseño e implementación concretos para formar un mecano utilizable directamente en tiempo de reutilización, los requisitos genéricos formarán con los patrones arquitectónicos y de diseño una variedad especial de mecanos que no serán utilizables directamente. Lo que se obtiene realmente es un mecano genérico (o mecano patrón) que deberá instanciarse en cada caso concreto antes de poder utilizarlo en una situación de desarrollo con reutilización. La adaptación puede ser similar a los

mecanismos bien conocidos de adaptación de los patrones de diseño [12] o estar basada en parámetros, de modo que la sustitución de un parámetro en un requisito genérico provoque una sustitución paralela en los *assets* de diseño e implementación relacionados. El diseño de herramientas tipo *wizards* que faciliten la adaptación de los mecanos genéricos desde los requisitos al diseño e implementación es una tarea que debe abordarse en un futuro inmediato, una vez se definan el proceso a seguir y los mecanismos de transformación.

6 Conclusiones y trabajo futuro

En este trabajo se han presentado varias aproximaciones a la reutilización de requisitos. Aunque no se ha medido de forma experimental, y ello entra dentro del trabajo futuro, la reutilización de requisitos individuales *normalizados* se ha mostrado plausible. La obtención automática de los *assets* de requisitos garantiza la normalización de los mismos en el repositorio. El siguiente paso en esta línea consiste en encontrar una representación común para el mayor número posible de tipos de escenarios de modo que se puedan comparar requisitos obtenidos de distintas fuentes.

Desde el punto de vista de los sistemas complejos, resulta más interesante la reutilización de mecanos, en la que la selección de un descriptor funcional facilita, mediante las relaciones de reificación, la obtención de los *assets* de diseño e implementación correspondientes. Para ello, la definición explicita de las relaciones intra-nivel entre los requisitos facilita la construcción de los descriptores funcionales y representa una buena base para la utilización de mecanos en el desarrollo de líneas de producto.

La última propuesta presentada implica la utilización de requisitos genéricos. La comparación de escenarios genéricos, de los que se han eliminado los detalles, se puede llevar a cabo mediante la utilización de técnicas de reducción de redes de Petri y búsqueda de analogías entre ellas, basadas en las metas de los actores. El trabajo que se plantea como continuación incluye el diseño de herramientas tipo *wizards* que faciliten la fase de adaptación tanto de los requisitos como del resto de *assets* relacionados.

Referencias

1. W.M.P. van der Aalst. Three Good reasons for Using a Petri-net-based Workflow Management System. In *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, 1996.
2. Raquel Anaya. *Desarrollo y gestión de componentes reutilizables en el marco de Oasis*. PhD thesis, Universidad Politécnica de Valencia, 1999.
3. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison–Wesley, 1999.
4. J. W. Brackett. Software Requirements. Curriculum Module SEI–CM–19–1.2, Software Engineering Institute, Carnegie Mellon University, 1990. http://www.sei.cmu.edu.
5. Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 1996.
6. Peter Clark and Bruce Porter. Building Concept Representations from Reusable Components. In *Proceedings of the AAAI'97*, pages 369–376, 1997.

7. Jacob L. Cybulski. Patterns in Software Requirements Reuse. In *Proceedings of 3rd Australian Conference on Requirements Engineering ACRE'98, Deakin University, Geelong, Australia*, pages 135–153, 1998.
8. Oscar Díaz and Juan José Rodríguez. Change case analysis. *Journal of Object-Oriented Programming (JOOP)*, 12(9):9–15,48, February 2000.
9. 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.
10. A. Finkelstein. Re–use of Formatted Requirements Specifications. *Software Engineering-Journal*, 3(5):186–197, May 1998.
11. W. Frakes and S. Isoda. Success factors of systematic reuse. *IEEE Software*, 11(5):15–18, September 1994.
12. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
13. Francisco J. García. *Modelo de Reutilización Soportado por Estructuras Complejas de Reutilización Denominadas Mecanos*. PhD thesis, Universidad de Salamanca, 2000.
14. Francisco J. García, Juan M. Corchado, and Miguel A. Laguna. CBR Applied to Development with Reuse Based on mecanos. In *Proceedings of the 13th International Conference on Software Engineering and Knowledge Engineering*, june 13-15, 2001. Buenos Aires, 2001.
15. C. Goti. A business classification for business rules. Technical Report TR03.563, IBM, Santa Teresa Lab., San José, EEUU, 1994.
16. Martin L. Griss, John Favaro, and Massimo D'Alessandro. Integrating feature modeling with RSEB. In *Proceedings of the Fifth International Conference on Software Reuse (ICSR-5)*, pages 76–85, 2-5 June, 1998. Victoria, B. C., Canada, June 1998. IEEE Computer Society, IEEE Press.
17. I. Jacobson. Object Oriented Development in an Industrial Environment. In *OOPSLA - Conference proceedings on Object-Oriented Programming Systems, Languages and Applications*, pages 183–191, Orlando, FL USA, ACM, 1987.
18. I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object–Oriented Software Engineering: A Use Case Driven Approach*. Addison–Wesley, 4ª edition, 1993.
19. Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse. Architecture, Process and Organization for Bussiness Success*. ACM Press. Addison-Wesley, 1997.
20. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA). Feasibility study. Technical Report CMU/SEI-90-TR21 (ESD-90-TR-222), Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213, November 1990.
21. Miguel A. Laguna, José M. Marqués, and Francisco J. García. A user requirements elicitation tool. *ACM SIGSOFT Software Engineering Notes*, 26(2):35–37, March 2001.
22. W. Lam. Achieving Requirements Reuse: A Domain-Specific Approach from Avionics. *The Journal of Systems and Software*, 38(3):197–209, September 1997.
23. K. Lee, KC. Kang, W. Chae, and BW. Choi. Feature-based approach to object-oriented engineering of applications for reuse. *Software-Practice and Experience*, 30(9):1025–1046, July 2000.
24. W.J. Lee, S.D. Cha, and Y.R. Kwon. Integration and Analysis of Use Cases Using Modular Petri Nets in Requirement Engineering. *IEEE Transactions on Software Engineering*, 24(12):1115–1130, December 1998.
25. P. Loucopoulos. Bussiness Rules Modelling: Conceptual Modelling and Object-Oriented Specifications. In *Proceedings of the IFIP Conference, TC8/WG8.1*, 1991.
26. M. Lowry and R. Duran. *The Handbook of Artificial Intelligence*, chapter Knowledge-Based Software Engineering, pages 241–322. Addison-Wesley, 1989.

27. Oscar López, Miguel Angel Laguna, and José Manuel Marqués. Generación Automática de Casos de Uso para Desarrollo de Software Basado en Reutilización. In *Actas de las V Jornadas de Ingeniería del Software y Bases de Datos*, pages 89–101, Valladolid, November 2000.
28. Oscar López, Miguel Angel Laguna, and José Manuel Marqués. Reutilización del Software a partir de Requisitos Funcionales en el Modelo de Mecano: Comparación de Escenarios. In *Actas de IDEAS 2001*, San José, Costa Rica, April 2001.
29. 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.
30. T. Moriarty. Business rule analysis. *Database Programming & Design*, 6(4):66–69, 1993.
31. Klaus Pohl. Requirements engineering: An overview. Technical Report TR 96/2, CREWS, 1996.
32. Jeffrey S. Poulin. Integrated support for software reuse in computer-aided software engineering (CASE). *ACM SIGSOFT Software Engineering Notes*, 18(4):75–82, 1993.
33. Rubén Prieto-Díaz. Implementing faceted classification for software reuse. *Communications of the ACM*, 34(5):89–97, May 1991.
34. P. Sawyer and G. Kontoya. Software Requirements, in Guide to the Software Engineering Body of Knowledge, Version 0.95. Technical report, May 2001. http://www.swebok.org.
35. Sema Group. *Euroware User's Manual*. Sema Group, 1996.
36. A. Silva. Across Version/Variant requirement traceability in avionics software development and testing. In *Proceedings of the 2nd European Reuse Workshop (ERW'98)*, pages 179–198, November, 4-6, 1998. Madrid (Spain), November 1998. European Software Institute (ESI).
37. M. Silva. *Las Redes de Petri en la Informática y la Automática*. Editorial AC, Madrid, 1985.
38. Mark Simos, Dick Creps, Carol Klingler, Larry Levine, and Dean Allemang. Organization domain modeling (ODM) guidebook - version 2.0. Technical Report STARS-VC-A025/001/00, Lockheed Martin Tactical Defense Systems, 9255 Wellington Road Manassas, VA 22110-4121, June 1996.
39. Alistair G. Sutcliffe, Neil A. M. Maiden, Shailey Minocha, and Darrel Manuel. Supporting scenario-based requirements engineering. *IEEE Transactions on Software Engineering*, 24(12):1072–1088, December 1998.
40. J. B. Warmer and A. G. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison–Wesley, 1999.
41. WfMC. Workflow management coalition terminology and glosary. Technical Report WFMC-TC-1011, Workflow Management Coalition, Brussels, http://www.aiim.org/wfmc/standards/docs/glossy3.pdf, 1996.