# El acceso a vocabularios controlados en la elaboración de especificaciones técnicas de ingeniería: compartir y reutilizar conocimiento

***Access to controlled vocabularies in the elaboration of engineering technical specifications as a way to share and reuse knowledge***

**Ricardo Eito-Brun**

**Universidad Carlos III de Madrid**
reito@bib.uc3m.es

## Resumo

La elaboración de especificaciones técnicas de ingeniería exige codificar y transmitir con precisión las características que deben cumplir los productos atendiendo a aspectos funcionales y no-funcionales. Las especificaciones, elaboradas normalmente en lenguaje natural no controlado, constituyen la base para las actividades de diseño y fabricación, siendo el principal artefacto del proceso de ingeniería: cualquier error en su formulación puede traducirse en sobrecostes y trabajos adicionales a los planificados inicialmente.

El uso del lenguaje natural no controlado en la elaboración de especificaciones ha demostrado su utilidad frente a otros modelos más estrictos (lenguajes formales), al ser más flexible y favorecer la comunicación entre las partes implicadas en el proceso. Sin embargo, la ambigüedad intrínseca al lenguaje natural dar lugar a problemas e imprecisiones. Para evitarlos se han propuesto distintas aproximaciones basadas en el uso de vocabularios y sintaxis restringidas.

## *Abstract*

*The elaboration of technical specifications in engineering projects needs to encode and transfer with precision the functional and non-functional features of the products to be built. Specifications are usually written using non-controlled natural language, and are the basis for the subsequent design and manufacturing activities. It is possible to say that specifications are the main artifact in the engineering processes, as any error in their elaboration may translate into additional costs and work activities.*

*The use of non-controlled natural language has shown its usefulness with respect to other methods (e.g. formal languages). Natural language is more flexible and makes easier the communication between the involved parties. Anyway, the ambiguity that characterizes natural language may lead to problems and lack of precision. To avoid that, different approaches have been proposed in the professional bibliography based on the use of restricted syntax and controlled vocabularies.*

*This paper provides an overview of the main contributions to this area, and proposes a method to*

La comunicación revisa las principales aportaciones de la bibliografía reciente en esta área y propone un modelo de acceso a vocabularios controlados y ontologías codificados en OWL para la especificación de requisitos mediante plantillas basadas en sintaxis restringidas.

*Access controlled vocabularies and OWL ontologies for writing specifications and requirements using templates based on restricted syntax.*

**Palavras-chave:** Control del vocabulario; Especificaciones técnicas ; Interoperabilidad.

***Keywords:*** *Vocabulary control; Technical specifications; interoperability.*

## 1. Introducción

Uno de los principales problemas en el desarrollo de aplicaciones informáticas basadas en la orientación MDE (Model Driven Engineering) o MDA (Model Driven Architecture) es la obtención de un primer modelo de análisis a partir de una especificación funcional escrita en lenguaje natural. Las aproximaciones propuestas en la bibliografía profesional y académica para resolver este problema se basan en el uso de lenguajes restringidos para la redacción de los requisitos del software. En esta comunicación se describe un proyecto en el que se ha desarrollado un vocabulario controlado, concretamente una ontología, para apoyar a los equipos de ingeniería en la redacción de especificaciones técnicas y requisitos de software. Las ventajas de esta aproximación no sólo pueden aplicarse en la generación automática de modelos de análisis o PIM (Platform independent models), sino también en otros procesos relacionados con la gestión de requisitos, como su verificación o reutilización entre proyectos.

En la bibliografía académica encontramos una línea de investigación continua en torno al uso de lenguajes restringidos para la especificación de requisitos, si bien la mayoría de las aproximaciones se basan en aplicar restricciones sintácticas y evitar así los problemas derivados de la ambigüedad del lenguaje natural. Se establece así un rango de opciones para la redacción de requisitos y especificaciones, que irían desde el lenguaje natural, no controlado, hasta los lenguajes formales basados en la lógica como son el lenguaje Z o VDM

(Vienna Development Method). Entre estas dos opciones extremas se situarían otros métodos como los lenguajes semi-formales – normalmente asociados a los lenguajes de modelado o diagramación tipo UML (Unified Modeling Language) y los lenguajes restringidos.

Estos últimos utilizan el lenguaje natural pero limitando en cierta medida la sintaxis y el vocabulario que puede utilizarse. Se combina así la capacidad expresiva y la facilidad de uso del lenguaje natural con las ventajas que pueden obtenerse de un mínimo control en el vocabulario para dotar a las declaraciones de una mayor precisión. En ellos se definen: a) reglas que indican cómo se deben escribir las declaraciones, y b) reglas sobre cómo se deben organizar los documentos de requisitos. En el primer grupo incluye reglas de estilo de redacción y el uso de tablas de decisión para representar condiciones *if-then*. Entre los ejemplos de las restricciones citadas por Lamsweerde (2009, p. 121) se incluyen éstas:

- Asegurar que todos los conceptos se definen antes de usarse.
- No incluir más de un requisito, hipótesis o característica del dominio en cada frase.
- Mantener las frases cortas.
- Usar “shall” para frases que indican requisitos obligatorios y “should” para los deseables.
- Evitar acrónimos y términos difíciles de entender.
- Usar ejemplos para clarificar declaraciones abstractas.
- Incluir diagramas para representar relaciones complejas entre elementos.
- Usar tablas para recoger datos relacionados.
- Evitar combinaciones complejas de condiciones anidadas que puedan resultar ambiguas.

Junto a estas reglas de estilo Lamsweerde menciona el uso de plantillas predefinidas (*predefined statement templates*). Su utilidad es presentar las declaraciones de *forma normalizada* utilizando campos como:

- **Identificador único** de la declaración
- **Categoría** o **tipo**: requisito funcional, de calidad, hipótesis, característica de dominio, definición, escenario, etc.
- **Especificación** propiamente dicha, redactada siguiendo las pautas de estilo anteriores.
- **Criterios de cumplimiento**, que indican cómo comprobar si la declaración es satisfecha por el sistema en las distintas etapas de su desarrollo. El autor incluye ejemplos en los que se usa este campo para precisar aspectos medibles de cara a la verificación del requisito; por ejemplo, que el tiempo de respuesta de un sistema no sea superior a un periodo determinado.
- **Origen** o fuente de la que se ha tomado la declaración.
- **Razón** por la que se incluye la declaración.
- **Interacción** positiva o negativa con otras declaraciones.
- **Prioridad**
- **Estabilidad**.

Finalmente, la elaboración del RD se debe regir por unas reglas que dicten las secciones en las que debe organizarse. Son ejemplos de estas *reglas* la plantilla IEEE Std-830, La SMAP-DID-P200-SW de NASA, la ya obsoleta PSS-05 de ESA, o la plantilla VOLERE propuesta por Robertson en 1999 y cuya última versión se publicó en mayo de 2010[1].

Según Lamsweerde (2009, p. 126), la combinación de reglas sobre el uso del lenguaje y la organización del RD en secciones predeterminadas preservan la expresividad y accesibilidad del lenguaje natural y reducen el ruido y la ambigüedad. Lamsweerde no trata en este apartado las restricciones sintácticas o semánticas en la redacción de requisitos, aspecto que sí se trata ampliamente en la mayoría de los artículos analizados (Durán Toro, 1999; Majumdar, 2011; Tjong, 2006; Renault, 2009; Boyd, 2007; Ketabchi, 2011; Fantechi, 2002).

[1] Disponible en: http://www.volere.co.uk/template.htm. Consultado el 09/01/2012.

***Especificaciones en lenguaje natural con patrones***

Este apartado describe brevemente algunas iniciativas donde se ha propuesto la reutilización de requisitos especificados en lenguaje natural restringido. En este grupo se incluye el artículo asignado al alumno, "*A Requirements Elicitation Approach Based in Templates and Patterns"* de **Durán Toro** et al (1999). **Durán Toro** plantea una aproximación basada en el uso de "patrones de requisitos" o R-patterns y "patrones lingüísticos" o L-Patterns. Los R-patterns consisten en plantillas estructuradas en una serie de campos o elementos de datos que guiarán al ingeniero en la captura de los requisitos durante su interacción con los futuros usuarios del sistema. Los L-pattern son frases de uso frecuente en la redacción de requisitos, que pueden usarse para documentar escenarios o interacciones (por ejemplo, para dar de alta a un cliente), y en las que se identifican fragmentos textuales variables que deberán ser reemplazados por los términos adecuados según cada caso. Estos fragmentos textuales que deben reemplazarse se denominan *aspectos parametrizables del patrón lingüístico*, y se escriben entre los signos reservados < y >; también se propone el uso de los caracteres { y } para indicar la selección de una opción entre varias que aparecen separadas por comas. Durante la captura de requisitos el analista dispondrá de unas plantillas que describen interacciones compuestas por varios requisitos escritos con L-patterns; en este texto se tendrán que hacer las sustituciones oportunas. Los autores señalan que están trabajando en el desarrollo de una herramienta CASE para facilitar la redacción de los requisitos. También proponen organizar los patrones en distintos grupos:

- Patrones para requisitos de almacenamiento de datos e información.
- Patrones para requisitos funcionales, que especifican los servicios que presta el sistema con la información que gestiona.
- Patrones para requisitos no funcionales.

Los datos que recoge cada plantilla (R-pattern) y los patrones lingüísticos (L-patterns) definidos para cada una de ellas se detallan en el resumen detallado de este artículo disponible en el Anexo I del trabajo. Las plantillas (R-pattern) que proponen se basan en las plantillas para casos de uso propuestas por **Alistair Cockburn** (los autores indican que han ampliado la propuesta de **Cockburn** para incluir requisitos no funcionales y destacan la integración de los L-patterns como mejora respecto al trabajo de Cockburn). Disponer de

plantillas pre-escritas para documentar casos de uso y requisitos no funcionales recurrentes es una aproximación interesante para reutilizar el trabajo realizado anteriormente y agilizar la especificación de casos de uso. No obstante, en opinión del alumno, el conjunto de patrones lingüísticos que proponen puede considerarse extremadamente limitado. La evolución de las especificaciones en elementos de diseño no se trata en el artículo. El concepto de patrón que proponen difiere del utilizado por **Lam** y otros autores, al no interpretarlo como un conjunto de requisitos relacionados entre sí para implementar una funcionalidad determinada, sino como una plantilla (R-patterns) con frases-tipo o estructuras lingüísticas predefinidas (L-patterns)[2]

En este primer grupo también se incluye el trabajo de **Majumdar** (2011) titulado "*Automated Requirements Modelling with ADV-EARS*". Se propone una sintaxis formal para redactar requisitos usando un lenguaje natural restringido llamado *Adv-EARS.* Los requisitos escritos con estas reglas podrán ser analizados automáticamente para identificar actores y casos de uso y generar el diagrama de casos de uso. Normalmente, la generación del diagrama de casos de uso es resultado de una actividad manual realizada por los analistas. Estas *conversiones* pueden resultar en información incompleta o en una comprensión incorrecta de las necesidades de los usuarios. Para hacer una conversión automática del texto de los requisitos en casos de uso es necesario aplicar un mayor rigor en la redacción de los requisitos. **Majumdar** cita estudios previos como los de **Dijkamm** y **Joosten** para transformar diagramas de actividad UML en diagramas de casos de uso, o el de **Lamsweerde** para transformar diagramas de objetivos en diagramas de casos de uso, pero señalan que hay pocas experiencias en la generación de diagramas de casos de uso a partir de requisitos textuales. Otras técnicas propuestas en la bibliografía como las notaciones formales tipo Z, las notaciones gráficas como UML o SysML, las basadas en escenarios (*table-driven requirements* y *ConCERT*) o pseudo-código, exigen una traducción compleja de los requisitos iniciales a ese lenguaje de representación, por lo que ven preferible usar un conjunto simple de "*estructuras sintácticas para requisitos*" que mejore su redacción. Esto está en línea con iniciativas como *Simplified Technical English*, ACE (*Attempt to Controlled English*), ECA (*Event-Condition-Action*) o EARS (*Easy Approach to Requirements Syntax*). De hecho, ADV-

[2] Podría decirse que un conjunto de L-patterns instanciados y agrupados en un R-pattern constituyen el equivalente al patrón de requisitos tratado por Lam, aunque conceptualmente las dos aproximaciones no son equivalentes.

EARS es una extensión de EARS en la que se incorporan nuevos tipos de requisitos y se propone una *gramática independiente de contexto* o CFC (*Context Free Grammar*) que facilite la interpretación de los requisitos y la generación de los diagramas. En ADV-EARS los requisitos se deben escribir (manualmente) siguiendo esa sintaxis restringida; luego se hace el análisis automático para generar árboles sintácticos para cada requisito e identificar actores y casos de uso. Si a partir del árbol sintáctico de un requisito se identifica más de un caso de uso, se establecerá una relación de tipo *include* entre ellos.

EARS tomaba como punto de partida de una clasificación de los requisitos en distintos grupos: *nominales* (*ubiquous*), *comportamiento no deseado*, *dirigido por eventos* y *dirigido por estados*. Los autores han añadido a ADV-EARS un nuevo tipo, el *híbrido*, que resulta de la combinación de los tipos *dirigido por eventos* y *condicional*, y que corresponde a un comportamiento que se inicia en respuesta a un evento con una pre-condición. También se ha extendido la sintaxis (conjunto de frases-tipo) para las categorías existentes en EARS con el fin de "*acomodar modelos de frases adicionales*". En la tabla 7 se muestra, para cada tipo de requisitos, las sintaxis propuestas en EARS y en ADV-EARS (en la tercera columna figuran en negrita las frases-tipo añadidas en ADV-EARS).

Tabla 7: ***Ejemplos de patrones EARS y ADV-EARS (Majumdar, 2011, p. 60)***

| Req Type | Definition in EARS | Definition in ADV-EARS |
|---|---|---|
| UB | The <system name> shall <system response> | The <entity> shall <functionality> \|<br>The <entity> shall <functionality> the <entity> for <functionality> |
| EV | WHEN <optional preconditions> <trigger> the <system name> shall <system response> | When <optional preconditions> the <entity> shall <functionality> \|<br>When <optional preconditions> the <entity> shall perform <functionality> \| When <entity> <functionality> the <entity> shall <functionality> |

| Req Type | Definition in EARS | Definition in ADV-EARS |
|---|---|---|
| UW | IF <optional preconditions> <trigger>, THEN the <system name> shall <system response> | IF < preconditions> THEN the <entity> shall <functionality> \|<br>IF < preconditions> THEN the <functionality> of <functionality> shall <functionality> \|<br>IF < preconditions> THEN the <functionality> of <functionality> shall <functionality> to <functionality> \| IF < preconditions> THEN the <functionality> of <functionality> shall <functionality> to <functionality> and <functionality> |
| ST | WHILE <in a specific state> the <system name> shall <system response> | WHILE <in a specific state> the <entity> shall <functionality> \|<br>WHILE <in a specific state> the <functionality> shall <functionality> |
| OP | WHERE <feature is included> the <system name> shall <system response> | WHERE <feature is included> the <entity>shall <functionality> \|<br>WHERE < preconditions> the <functionality> shall <functionality> \| WHERE < preconditions> the <functionality> of <functionality> shall <functionality> to <functionality> |
| HY | Not defined | <While-in-a-specific-state> if necessary the <functionality> shall <functionality> \| <While-in-a-specificstate> if necessary the <entity> shall perform <functionality> \|<br><While-in-a-specific-state> if <preconditions> the <functionality> shall <functionality> |

La derivación de casos de uso y de actores se hará mediante estas reglas:

1. Los fragmentos <functionality> de ADV-EARS corresponderán a casos de uso.
2. Los fragmentos <entity> serán actores o clases. Aquí se necesita la intervención del analista para diferenciar unos y otros.
3. Los fragmentos <functionality> shall <functionality> se traducen como relaciones “include” entre dos casos de uso.
4. También se traducen como relaciones entre casos de uso los fragmentos del tipo <functionality> of <functionality>. Aquí se require la intervención del analista para ver si se trata de una relación include o extend.

Como ejemplo proponen este requisito:

```
if <age is less than 18> then
perform validation process shall invoke error-handler to
reject application".
```

En este caso el analizador identificaría tres casos de uso: *perform validation process*, *invoke error-handler* y *reject application*, y una relación *includes* entre *perform validation process* e *invoke error-handler*.

**Majumdar** concluye el artículo con un caso de estudio basado en un sistema de seguros (destacamos que los requisitos se han rescrito manualmente desde el lenguaje natural en el que se formularon inicialmente a la sintaxis ADV-EARS); el ejemplo recoge algunos árboles sintácticos generados por el analizador (uno para cada tipo de requisito). Concluyen señalando que ADV-EARS puede extenderse para incluir más restricciones o frases-tipo.

Otra aproximación en esta línea de trabajo la encontramos en **Tjong** (2006). Su artículo, titulado "*Improving the quality of natural language Requirements Specification through Natural Language Requirements Patterns"* pretende reducir el problema de la ambigüedad y la imprecisión en las especificaciones escritas en lenguaje natural mediante el uso de "*patrones de calidad del lenguaje*" y unas reglas-guía (*guiding rules*), aplicables en cualquier contexto.

Sobre los patrones lingüísticos, **Tjong** identificó que el uso frecuente de las conjunciones "*and*", "*or*", "*but*", "*and/or*" y "*both*" a menudo resulta en frases ambiguas. Por ejemplo, "*and"* se usa para representar varias combinaciones de requisitos en una única frase, y en ocasiones se usa una coma o "*but*" con esa misma intención. Algo similar sucede en el caso de "*or*". Para evitar estos problemas, proponen reducir estos casos a unos patrones llamados GAND (*Generic AND Pattern*), GOR (*Generic AND Pattern*), IFFP (*If and Only If Pattern*), CACP (*Compound AND Condition pattern*), GP *(Generic Pattern)*, GNP (*Generic Negative Pattern*), ECP (*Event Condition Patterns*) o y los TP (Time Patterns). Los dos últimos representan, respectivamente, operaciones causadas por eventos y/o condiciones y requisitos relacionados con el tiempo.

Sobre las *reglas-guía*, éstas se combinarán con los patrones lingüísticos para reducir al máximo la imprecisión y la ambigüedad. En el artículo se mencionan únicamente quince

reglas, aunque se señala que han definido un número mayor. Se trata de reglas de estilo, por ejemplo: usar frases afirmativas con un único verbo [Regla 1], evitar la voz pasiva [Regla 2], evitar términos como "*either*", "*whether*", "*otherwise*" [Regla 4], "*eventually*", "*at least*" [Regla 5], usar "*at most*" y "*at least*" en lugar de "*maximum*" y "*minimum*" [Regla 6], evitar "*both*", [Regla 7], "*but*" [Regla 8], "*and/or*" [Regla 10], "*not only*", "*but also*" [Regla 11], etc. Las reglas 13, 14 y 15 indican la necesidad de definir un glosario y listas con los acrónimos y abreviaturas que se empleen en la especificación de requisitos.

Según **Tjong** los patrones lingüísticos y las reglas-guía permiten simplificar las frases y reducir la imprecisión y ambigüedad del lenguaje natural. Para comprobar la aplicabilidad de su aproximación estudiaron documentos de requisitos de distintos dominios que posteriormente rescribieron aplicando sus reglas y patrones. Señalan que están desarrollando una herramienta llamada SREE (*Systemised Requirements Engineering Environment*) para ayudar al analista a rescribir las especificaciones aplicando estas reglas y patrones. Los requisitos así redactados podrán ser analizados y etiquetados por una herramienta automática con el fin de generar diagramas y modelos de análisis (no se ofrecen más detalles sobre esta herramienta ni la aproximación utilizada para generar esos diagramas).

**Renault** et al., en sus artículos "*PABRE: Pattern-based Requirements Elicitation*" (Renault, 2009a) y "*A Pattern-based method for Building Requirement Documents in Call-for-tender process"* (Renault, 2009b) propuso otra iniciativa encuadrable en este grupo. PABRE corresponde a *Pattern-Based Requirements Elicitation Method*, y es un método para facilitar la identificación y captura de requisitos a partir de patrones. Se aplica en la redacción de requisitos para proyectos basados en la selección e integración de COTS, si bien ninguno de los dos artículos describe la forma de evaluar la correspondencia o similitud entre las especificaciones basadas en PABRE y los COTS susceptibles de ser reutilizados. PABRE surge de la experiencia del departamento CITI del *Public Research Centre Henri Tudor* (CPRHT) de Luxemburgo, para capitalizar el conocimiento adquirido en proyectos anteriores y transferirlo con facilidad de un proyecto a otro. Hasta que se formuló PABRE, la reutilización de requisitos se hacía duplicando requisitos de proyectos anteriores. Esto no era efectivo al no estar los requisitos normalizados y al depender en gran medida del contexto en el que se

habían identificado. Además, los analistas debían estar al corriente de los requisitos existentes en proyectos anteriores.

La propuesta de PABRE consiste en elaborar un catálogo de patrones de requisitos del que se podrán derivar los requisitos específicos de cada proyecto, *instanciando* esos patrones en requisitos que constituirán el *Requirements Book* de cada proyecto. Se adoptó el concepto de patrón de requisitos como artefacto central para la reutilización, ya que los requisitos que aparecen una y otra vez, con ciertas variantes, en distintos proyectos, podrían equipararse a *soluciones a problemas particulares en un contexto dado*. Cada patrón contará con:

- **Metadatos del patrón**: nombre, descripción, autor, comentarios, objetivo, proyectos en los que se ha identificado y palabras clave para su recuperación. Es especialmente importante el objetivo o *goal*, ya que se utilizará como primer filtro para decidir si el patrón es aplicable a un proyecto particular.

- **Forma del patrón**: un patrón que se haya usado en distintos proyectos puede haberse escrito de forma diferente; se podrá tener así distintas formas para un mismo patrón, si bien se espera que las variaciones sean mínimas. Cada forma contará con metadatos específicos como su nombre, descripción, autor, comentarios, versión u origen. Una forma tendrá siempre una parte fija y podrá tener extensiones.

    - La **parte fija** es una frase breve en lenguaje natural que describe qué debe hacer el sistema, no cómo hacerlo. Por ejemplo, "*The solution shall give an alert in case of failure.*"

    - Las **extensiones** concretan la parte fija con información adicional y restricciones. Pueden especificarse rescribiendo la parte fija o restringiéndola mediante parámetros. En el ejemplo anterior, podrían añadirse extensiones para indicar qué tipo de errores se deben notificar y de qué manera deben enviarse las alertas.

La parte textual de las extensiones puede contar con parámetros o argumentos; se indicará qué valores pueden tomar esos argumentos (en la plantilla hablan de *métricas para los parámetros*, para hacer referencia a estos valores).

- **Dependencias**: permiten relacionar patrones de requisitos.

- **Esquemas de clasificación**: los patrones de requisitos se organizarán jerárquicamente a partir de la norma ISO 9126-1, la estructura de la plantilla *Volere* y un esquema de clasificación propio derivado de los experimentos completados por el *CITI*.

**Renault** señala que en el momento de redactar el artículo, el catálogo contaba con 48 patrones derivados del análisis de especificaciones existentes. Los patrones para requisitos no funcionales se ha obtenido a partir de siete especificaciones mediante un proceso de generalización, en el que se definió un patrón para varios requisitos que tenían objetivos similares. A partir del catálogo, la identificación de requisitos reutilizables consiste en la búsqueda y selección de patrones por parte del analista y del cliente. El analista seleccionará el esquema de clasificación que guiará la exploración del catálogo, y posteriormente comparará los objetivos de cada patrón con los del cliente para seleccionarlos o descartarlos. Para los patrones seleccionados, el analista explicará las formas de éste y elegirá la más adecuada[3]. Si no hay una forma que encaje con la necesidad del cliente, el analista elaborará un nuevo requisito que servirá como *feedback* para el catálogo de patrones.

La aproximación propuesta por **Renault** se ha validado mediante entrevistas con un equipo de ingenieros y en un proyecto para renovar un software de biblioteca digital. En este caso se exploraron en una reunión 21 de los 50 patrones del catálogo, y de los 22 requisitos, 20 se derivaron de patrones existentes (más de un 90%) en tan solo ocho minutos. Se calcula que el proceso de captura de requisitos podría reducirse a 4 ó 5 días frente a los 15-20 actuales. Como línea de trabajo futura señalan la necesidad de validar PABRE con más casos prácticos. En otro artículo de **Renault** (2009b), "*A Pattern-based method for Building Requirement Documents in Call-for-tender process*", se describe otro caso de estudio para requisitos no funcionales de un sistema CRM SaaS en el que los resultados también fueron satisfactorios. Sin embargo, en ninguno de los dos artículos se da información sobre la forma de relacionar las características de los COTS con los requisitos derivados del catálogo de patrones.

[3] Se puede obtener un único requisito concatenando la parte fija y las extensiones de un mismo patrón, por ejemplo: "*The Solution shall give an alert in case of failure. Alerts provided by the solution shall be: EMAIL, SMS. Failures to be alerted of shall be: ServerCrash, NetworkCrash*"

El uso del lenguaje natural restringido también es tratado por **Boyd** (2007) en su artículo "*Optimal-constraint Lexicons for Requirements Specification*". Se parte de los llamados CNL (*Constrained Natural Languages*) que se utilizan en la redacción de documentos técnicos y especificaciones con el fin de reducir la ambigüedad del lenguaje natural y mantener su comprensibilidad y expresividad, restringiendo el léxico (vocabulario) y las opciones sintácticas. Su fundamento es que la ambigüedad de un requisito se puede reducir si el léxico y/o la gramática usados para expresarlo se restringen. Restringir la gramática permite evitar frases complejas, y restringir el léxico permite eliminar variantes lingüísticas innecesarias y mantener únicamente los términos menos ambiguos. **Boyd** indica que los CNL existentes no se han centrado en la evaluación de la calidad de sus resultados, y que la selección de términos y frases-tipo (restricciones léxicas y sintácticas) se basa en las preferencias de un lexicógrafo más que en un análisis de los factores que pueden afectar a la calidad dependiendo de si se selecciona un término u otro. Los CNL acostumbran a ser lenguajes estáticos que no permiten añadir términos adicionales a los identificados en el análisis preliminar, y que se derivan de corpus textuales de un dominio particular combinando técnicas de análisis automático y participación de expertos. Esto hace que los CNL sean específicos de dominios particulares, con lo que están condicionados por las preferencias de los lexicógrafos que trabajan en su elaboración.

El texto de **Boyd** presenta una aproximación automatizable para restringir de forma óptima el léxico de un CNL a partir de las relaciones semánticas entre los términos procedentes de un conjunto de requisitos. El artículo no trata las restricciones de la gramática. Con el fin de lograr una selección automática de los términos de mayor calidad (es decir, menos ambiguos), el autor introduce el concepto de *reemplazabilidad*. La *reemplazabilidad* parte del concepto de *similitud léxica*. La *similitud léxica* cuantifica en qué medida los significados de dos términos son parecidos, con el fin de identificar términos redundantes. La *similitud* indica si dos palabras pueden intercambiarse, aunque esto no puede hacerse de forma desatendida ya que podría conducir a errores (por ejemplo con los hiperónimos o términos genéricos), por lo que se debe considerar la posición relativa de un término en una red semántica u ontología a la hora de decidir su capacidad de remplazar a otro. Dado este problema, **Boyd** plantea usar la *reemplazabilidad* en lugar de la *similitud léxica*, limitando su

estudio a los verbos (no se aplica a sustantivos ya que en las especificaciones normalmente se usan nombres propios).

La *reemplazabilidad* se define como la posibilidad de reemplazar un término X por otro término Y en un dominio particular. Es una propiedad asimétrica porque no se puede garantizar que dos términos puedan reemplazarse en ambas direcciones. Se calcula a partir de la *similitud léxica*, *convencionalidad* y *polisemia* mediante esta fórmula:

```
Reemplazabilidad (X, Y) = Similitud(X, Y) * ( Fy / Fx) * (Px / Py)
```

Donde:

- X e Y son dos términos, de forma que Y es un sinónimo o un hiperónimo de X
- Fx y Px son la frecuencia y la polisemia del término X en el documento de requisitos.
- Similitud(X,Y), es mayor o igual que un valor establecido entre 0.6 y 0.7 (guiados por su experiencia).

Si la *reemplazabilidad* es igual o mayor que 1, entonces se podrá sustituir un término por el otro. Un término sólo se reemplazará por un sinónimo o hiperónimo que sea usado con más frecuencia en la especificación o que tenga una menor polisemia (menos acepciones) en el contexto de la especificación. Si no hay un término que cumpla estas condiciones, se mantendrá el término original. Para evitar la problemática derivada de reemplazar un término por otro con un significado más genérico se propone considerar los nombres que actúan como objetos o sujetos de los verbos analizados (recordamos que las *reemplazabilidad* sólo se aplican a los verbos) y su proximidad semántica.

Para mantener y registrar los datos sobre *reemplazabilidad* entre pares de términos, se utilizan las llamadas *matrices de reemplazabilidad*. En realidad son n matrices, una para cada grupo de palabras entre las que se ha identificado cierta similitud (es decir, para cada conjunto de sinónimos e hiperónimos). Cada matriz cuenta con dos dimensiones y las filas y columnas corresponden a los términos. Para cada término se indica su categoría gramatical o PoS y su "significado" tras desambiguarlo. El significado se indica mediante un subíndice que indica a cuál de las posibles acepciones del término corresponde el uso que se hace de él en el documento.

El proceso que se propone para restringir el léxico es el siguiente:

1. Cada requisito se procesa individualmente para extraer su estructura gramatical, sus términos, la PoS de cada uno de ellos y su significado. Para el análisis gramatical aplican la técnica automática MBSP (*Memory Based Shallow Parser*) y una inspección visual de sus resultados. Para la desambiguación semántica o WSD intentaron aplicar *Word-Net::SenseRelate* y *Sense Learner 2.0*. Pero estas herramientas usan información contextual para hacer una determinación probabilística del significado de cada palabra; dado que en los requisitos suelen usarse nombres propios, estas herramientas no fueron aplicables, por lo que la WSD se hizo manualmente con *WordNet* como diccionario de referencia.

2. Cada término usado en el requisito se expresa en la forma word#pos#sense.

3. Luego se hace la "optimización", que consiste en usar los datos de cada término para actualizar los valores de la matriz.

4. A continuación se hace la "traducción", que hace las sustituciones oportunas en el texto del requisito.

En las conclusiones señalan que la *reemplazabilidad* puede completarse dando pesos distintos a la frecuencia y a la polisemia, y que este trabajo puede unirse a otras investigaciones sobre lenguaje natural restringido en los cuales los ingenieros deben definir los sustantivos, verbos, adverbios y adjetivos que conformarán el léxico del CNL. La reemplazabilidad podría guiar la selección de términos, y dejaría de ser necesario contar con un lexicógrafo que la haga manualmente al aplicarse técnicas automáticas. Según el alumno, esta conclusión es cuestionable, ya que los autores señalan que la WSD se hizo manualmente. La propuesta está pendiente de una validación empírica que demuestre de qué forma la selección de términos afecta a la comprensibilidad, expresividad y no ambigüedad de los requisitos.

**Ketabchi** (2011), en su artículo "*A Norm-based Approach towards Requirements Patterns*" también aplica el lenguaje natural restringido. Parte de la utilidad de los patrones para evitar el sesgo que introduce la experiencia del analista en la captura de requisitos. El concepto de patrón se toma de la definición de **Ambler** (2000): "*a solution to a common problem, taking*

*relevant forces into account and enabling the reuse of proven techniques and strategies*"[4] En la captura de requisitos, el uso de patrones puede ofrecer los siguientes beneficios:

- Ofrecer una guía sobre qué información se debe incluir y dirigir la atención del analista a las distintas alternativas.
- Ahorrar tiempo y esfuerzo al no comenzar la captura de requisitos desde cero.
- Dar consistencia al proceso.
- Usar las mejores prácticas y la experiencia de los expertos en situaciones similares.
- Reutilizar el conocimiento.
- Recordar requisitos o aspectos que el analista podrían obviar.
- Guiar las actividades de diseño y especificación de pruebas.

La propuesta de **Ketabchi** está basada en la semiótica, disciplina encargada del estudio de los signos y de cómo estos actúan en la sociedad. Se entiende por signo *cualquier cosa que conlleva un significado, como una palabra, sonido, objeto o imagen.* Se utiliza la representación clásica del triángulo de **S. J. Pierce** formado por un *signo*, el *objeto* al que se refiere el signo y un *intérprete* (*agente* que sigue unas *normas* para interpretar el *signo*), de forma que la relación entre el *signo* y el *objeto* se establece de forma subjetiva por el *intérprete*. Según el autor, la ingeniería de requisitos debería acordar el uso de los signos y la naturaleza de la información que los usuarios crean, analizan, guardan y aplican al comunicarse con otros usuarios. No es ésta la primera propuesta que relaciona semiótica y gestión de requisitos: **Ketabchi** cita MITAIS, técnica de análisis de dominio que distingue entre el "*espacio de decisión*" y el "*espacio de información*". En la introducción teórica, **Ketabchi** también señala que en la captura de requisitos hay dos procesos relacionados:

[4] "Una solución a un problema común, considerando las fuerzas relevantes que permite el reuso de técnicas y estrategias probadas".

También recogen en su artículo la definición del IEEE 610-1991 "*a meaningful regularity ahat can be used to classify objets or other itemrs of interest" o (28) "an approach to specifying a particular type of requirement*" (una regularidad signficativa que se puede usar para clasificar objetos u otros elementos de interés) [Traducción del alumno]

- Articular el espacio de decisión, donde el usuario es asistido por el analista para expresar el problema.

- Configurar el espacio de información, donde se representa la información requerida por el problema especificado.

Partiendo de esta teoría, **Ketabchi** describe la aproximación práctica basada en el uso de *patrones de problemas* enlazados a *patrones de requisitos*, de forma que para cada *patrón de problema* se identifican los requisitos necesarios para su resolución. El método consta de dos actividades:

- **Análisis y modelado del dominio**. Se identificarán los participantes clave o actores y sus funciones o roles. A continuación se hará un análisis de procesos donde se identificarán las relaciones entre procesos, actividades y participantes clave. El resultado de este análisis se volcará al repositorio de patrones de problemas.

Así se divide el dominio del problema global (plasmado en el análisis del proceso) en partes más pequeñas llamadas “patrones-problema” que relacionan cada actor con una actividad del proceso que debe completar. Se puede deducir que un patrón de problema es una actividad asignada a un actor.

Este análisis se completa con un análisis de *normas* y *reglas de negocio* para los procesos, creándose un repositorio de normas que facilitarán el análisis de requisitos.

- **Articulación del espacio-problema** y **configuración del espacio-requisitos**. El modelo de participantes clave, el repositorio de patrones y el de reglas de negocio sirven como entrada al proceso de *articulación*. En él el usuario expresa y especifica el problema de una forma más estructurada. La aplicación práctica de estas ideas se realizó con un ejemplo basado en el sistema de gestión de bibliotecas de la University of Reading Library. En primer lugar se identifican participantes clave y sus funciones. Los procesos de la organización se analizan y se representan mediante casos de uso UML. Para cada caso de uso se hace un diagrama de secuencia a partir del cual se crea una matriz de participantes y procesos. También se registran las reglas de negocio usando el formato prestablecido:

```
Whenever <condition>
if <state> then <agent> is <deonticoperator> to <action>.
```

**Ketabchi** indica que la mayoría de estudios previos no incluyen este tipo de recomendaciones para capturar reglas de negocio, motivo por el cual denominan a su aproximación "*basada en normas*" o "*norm-based*". Señalan que ésta es una de las fortalezas de su método. Finalmente, en su artículo señalan que se hace un análisis de requisitos usando SAM (*Semantic Analysis Method*), que tiene como resultado un diagrama de conceptos. No se incluye información sobre el objetivo de esta red semántica ni sobre cómo se obtiene.

Otra iniciativa que incluimos en este grupo es la de **Fantechi** (2002), "*Applications of linguistic techniques for use case analysis*", donde se aplican técnicas de procesamiento lingüístico para detectar defectos relacionados con la ambigüedad del lenguaje natural en documentos de requisitos expresados mediante casos de uso[5]. Los diagramas de casos de uso constituyen un excelente vehículo para la comunicación, si bien sirven únicamente como una vista-resumen de las capacidades del sistema; no permiten especificar el comportamiento del sistema y siguiendo a **Alistair Cockburn**, lo habitual es usar el lenguaje natural para especificar los escenarios y sus extensiones. El artículo no trata las técnicas para redactar especificaciones en lenguaje natural, sino técnicas para evaluar la calidad lingüística de las especificaciones escritas en lenguaje natural que alerten al autor de posibles problemas en la redacción.

**Fantechi** describe el proyecto CAFE y la experiencia de *Nokia* en el modelado de casos de uso con lenguaje natural para especificar los requisitos funcionales de una aplicación para

[5] Un caso de uso se describe como una interacción entre un sistema y su entorno iniciada por un actor externo para lograr un objetivo. Todo caso de uso consiste en un conjunto de interacciones orientadas a un objetivo entre actores externos y el sistema. El término actor se usa para describir cualquier persona o sistema que tiene un objetivo en el sistema o interactúa con él. Un actor primario inicia el comportamiento del sistema para lograr dicho objetivo. Un actor secundario interactúa con el sistema pero no inicia el caso de uso. La descripción del caso de uso incluye posibles extensiones a esa secuencia, como secuencias alternativas que conduzcan a un error controlado ante una excepción. Un escenario representa una secuencia única a través de la cual el caso de uso alcanza su objetivo (escenario principal o nominal). Los escenarios se suelen representar gráficamente mediante diagramas de secuencia.

móviles. En el proyecto se usó una aproximación lingüística para recoger métricas y analizar los requisitos desde los puntos de vista cualitativo y semántico, y para verificar su consistencia. En la sección 3 del artículo cita iniciativas previas para la evaluación de la calidad de los requisitos escritos en lenguaje natural[6]; la sección 4 describe las técnicas de procesamiento del lenguaje natural utilizadas en su trabajo.

El objetivo es que la especificación escrita en lenguaje natural mantenga estas características:

- **Expresividad**, distinguiendo entre **mitigar la ambigüedad** y **mejorar la comprensibilidad**.
- **Consistencia**, para evitar la presencia de contradicciones e incongruencias.
- **Completitud**, para evitar que falten partes necesarias en el documento de requisitos.

Los componentes del lenguaje natural asociado a los casos de uso (es decir, las frases) deben analizarse desde los puntos de vista léxico, sintáctico y semántico, ya que se puede hablar de ambigüedad léxica y de ambigüedad sintáctica. Por ejemplo, una frase puede no ser ambigua desde el punto de vista sintáctico (en el sentido de que sólo se puede obtener un árbol de derivación a partir de ella), pero puede serlo semánticamente al contar con palabras que tengan más de un significado.

[6] Entre estas iniciativas destacan las siguientes (remitimos a la bibliografía del artículo analizado para obtener más detalles):

- Macias y Pulman aplicaron técnicas NLP independientes de dominio basadas en un vocabulario controlado, fijando el estilo de redacción con unas reglas pre-determinadas que debían seguirse.
- Goldin y Berry implementaron una herramienta para extraer segmentos repetitivos que permitían identificar conceptos representativos en el campo de aplicación. La técnica propuesta se restringía al análisis léxico del texto.
- Hooks trató un conjunto de características de calidad de los requisitos escritos en lenguaje natural, y ofrece un amplio repertorio de orígenes de defectos en los requisitos en lenguaje natural y los riesgos asociados.
- Wilson examina la evaluación de la calidad de los requisitos en lenguaje natural y desarrolló una herramienta automática llamada ARM (Automated Requirements Measurement) para realizar el análisisi frente al modelo de calidad.
- Fuchs propone un modelo llamado ACE (Attempt Controlled English) que consiste en un lenguaje natural restringido, fácil de enterender por los participantes clave y lo suficientemente simple para evitar ambigüedades.
- Mich y Garigliano propusieron unas métricas para medir la ambigüedad semántica y sintáctica de los requisitos, usando una herramienta llamada LOLITA.
- Natt y Dag presentaron una aproximación basada en técnicas estadísticas para analizar la similitud entre requisitos en lenguaje natural con el fin de identificar duplicados. Esta técnica ha demostrado ser útil para revelar interdependencias.
- Ambriola y Gervasi definieron el sistema CIRCE, que construye modelos semi-formales de forma casi automática, extrayendo información del texto de los requisitos.

Para representar la calidad de los requisitos escritos en lenguaje natural, **Fantechi** presenta un espacio bidimensional donde el eje horizontal recoge las características que deben alcanzarse (expresividad, consistencia y completitud) y el eje vertical corresponde a las perspectivas léxica, sintáctica y semántica del lenguaje. El autor señala que las técnicas de procesamiento del lenguaje natural no son suficientes para satisfacer todos estos objetivos (concretamente la consistencia y la completitud), aunque sí se pueden aplicar para evaluar la expresividad **mitigando la ambigüedad léxica** y **sintáctica**. Con este fin se propone el uso de analizadores léxicos para detectar y corregir términos que puedan resultar ambiguos, y analizadores sintácticos para detectar frases que puedan tener diferentes interpretaciones. De cara a **mejorar la comprensión**, se hace: a) una evaluación léxica para detectar palabras poco comprensibles, y b) una evaluación sintáctica para detectar sentencias con estructura compleja, que sean difíciles de entender. Estas técnicas se aplican mediante las herramientas PLN *QuARS*, *ARM* y *SyTwo*.

- *QUARS (Quality Analyser for Reqirements Specifications)* se basa en un modelo de calidad compuesto por propiedades que se evalúan a partir de indicadores sintácticos y estructurales. Los indicadores se recogen en diccionarios específicos que contienen términos y construcciones lingüísticas características de algún defecto particular, y que podrían detectarse simplemente inspeccionando los requisitos. *QuARS* analiza las sentencias y señala aquellas palabras que hacen al documento ambiguo o poco claro desde el punto de vista léxico. Algunos ejemplos de los defectos que identificaría QUARS son los siguientes:

```
The C code shall be clearly commented
The system shall be as far as possible composed…
The system shall be such that… possibly without….
```

- *ARM (Automated Requirements Measurement)* ofrece métricas para evaluar especificaciones a partir de un conjunto de indicadores obtenidos a partir de una base de datos de términos y frases procedentes de requisitos de proyectos de NASA. La herramienta es similar a *QUARS*, en el sentido de que ayuda – en palabras del autor - a “escribir los requisitos correctamente”, no a “escribir los requisitos correctos”.

- *SyTwo* es una herramienta web que analiza el texto en inglés desde los puntos de vista léxico y sintáctico y comprueba el cumplimiento de las reglas del inglés simplificado. Es una evolución de *QUARS* y adopta el mismo modelo de calidad. El programa es capaz de hacer el árbol de derivación de cada frase, asociar a cada nodo sus datos morfosintácticos y calcular el índice de legibilidad de **Coleman-Liau**. También identifica frases ambiguas desde el punto de vista sintáctico, como por ejemplo: "*The system shall not remove faults and restore service*".

La aproximación se evaluó en la práctica con un documento de requisitos de NOKIA para un reproductor de radio para móviles (con unos cien casos de uso).

El artículo de **Fantechi** también recoge algunas frases problemáticas identificadas durante el análisis, por ejemplo:

- *This procedure is performed by the user to enter the frequency (Implicit sentence: indicator this).*
- *In addition, the user is naturally able to adjust the volume (Vague sentence: indicator naturally)*
- *The user can switch the radio on by selecting Radio from the menu (Under-specified sentence: indicator menu).*

Como parte de las conclusiones, **Fantechi** señala que se deben aplicar técnicas complementarias al análisis léxico y sintáctico para resolver problemas de consistencia y completitud, si bien esto obligaría a considerar los métodos formales. Dado que éstos son complejos y costosos, proponen una aproximación más simple basada en el estudio de relaciones entre actores. Así, han identificado que una especificación está formada por una serie de casos de uso, que cada caso de uso cuenta con un escenario que define el comportamiento del sistema, y que cada escenario tiene una estructura lingüística interna que define la relación entre los actores y las operaciones. Lo que proponen es determinar relaciones a partir de la estructura sintáctica de cada frase usada en los escenarios, generando una tupla del tipo (Actor_1, Verbo_i, Actor_2, Use_Case_id). La relación entre dos actores puede extenderse por transitividad a otros actores, creando cadenas. La identificación de todos los ítems derivados de esta forma la llaman *núcleo de relaciones*, y

muestra todas las relaciones funcionales entre actores; a partir de allí se podrían identificar inconsistencias o comportamientos dinámicos incompletos. Para detectar estas relaciones se propone usar técnicas de análisis lingüístico como FDG (*Functional Dependency Grammar*) y herramientas existentes como *CMAP* o *Connexor*. Como futuras líneas de trabajo incluyen la identificación de las herramientas adecuadas y experimentar con conjuntos de casos de uso más amplios. El alumno considera que esta aproximación puede resultar interesante pero no parece ofrecer una vía clara para detectar posibles inconsistencias, ya que sería necesario contar con información semántica adicional que dotase al método la capacidad de conocer entre qué cuádruplas se origina realmente un problema de inconsistencia.

Finalmente, **Videira** et al. (2006), en "*A linguistic patterns approach for requirements specification*" presentan el lenguaje de especificación de requisitos *ProjectIT-RSL*, basado en la identificación de los patrones lingüísticos más utilizados en la especificación de requisitos en lenguaje natural. *ProjectIT* parte del *Information Systems Group* del *INESC-ID,* cuyo objetivo era crear un entorno de desarrollo de software que diese soporte a la gestión del proyecto, ingeniería de requisitos, análisis, diseño y generación de código. En la arquitectura de *ProjectIT* se incluye el componente *ProjectIT-Requirements,* que a su vez incorpora el lenguaje RSL (*Requirements Specification Language*) y un editor para la validación de requisitos, *PIT-Studio/RSL* que ofrece funciones como autocompletar, auto-formatear, anotaciones, alertas y errores, resaltado de sintaxis incorrectas, etc. Para definir las **reglas del lenguaje** de especificación de requisitos (RSL), se analizó el formato y estructura de documentos. *ProjectIT-RSL* parte de que la mayoría de las frases de una especificación indican qué debería hacer el sistema y presentan una perspectiva operacional del mismo. Estas frases siguen un patrón común, sencillo, en el que un sujeto ejecuta una operación – expresada mediante un verbo – que afecta a un objeto. Otras sentencias, más elaboradas, incluyen una condición; otras especifican los atributos que definen una entidad. A partir de esta información se generó un metamodelo que identificó los siguientes conceptos:

a) Actores – recursos activos, por ejemplo un sistema externo o un usuario final, que ejecutan operaciones sobre una o más entidades.

b) Entidades – recursos estáticos afectados por las operaciones. Tienen propiedades que describen su estado.

c) Operaciones – se describen mediante secuencias de acciones sencillas que afectan a las entidades y a sus propiedades.

Respecto a las **reglas de organización**, recogen una estructura estándar para los documentos. Se separa la especificación o definición de entidades del dominio de la especificación de requisitos funcionales. Un tercer componente de ProjectIT-RSL los las **reglas de frases** *(sentence rules)*. Distinguen dos tipos de frases: declaraciones y definiciones. Las declaraciones nombran un concepto y lo asocian a un tipo de dato específico. Las definiciones expresan información sobre un concepto. Se establecen reglas para la especificación de los conceptos base (entidad, actor y operación). Un cuarto elemento son las **reglas de entidad** *(entity rules)* que especifican entidades mediante sentencias simples que especifican sus propiedades, especialización, equivalencia y la relación entre ellas. Un quinto elemento son las reglas para definir **actores** y **operaciones**. En términos de operaciones, necesitamos declarar una lista de todas las operaciones disponibles, para las que se dará una descripción detallada. Los autores señalan que el número de frases-tipo para la definición de operaciones es más grande que el número de patrones para la definición de entidades.

**La función del vocabulario controlado**

Las aproximaciones anteriores centran su atención en el control de aspectos sintácticos. Si se combina este nivel de control con el control del vocabulario, es posible obtener especificaciones con un mayor nivel de precisión, potencialmente verificables de forma automática a partir de las relaciones entre conceptos que se establecen en las ontologías.

Para demostrar la aplicabilidad práctica de esta aproximación se ha diseñado una ontología para el dominio de control de satélites que reúne 400 clases y facilitar la elaboración de especificaciones. La ontología se ha derivado del análisis manual de una serie de especificaciones y documentos de requisitos, y se ha codificado mediante el útil software Protégé. A partir de esta ontología, el autor de los requisitos puede completar su trabajo rellenando una serie de patrones basados en ADV-EARS y seleccionando los términos y conceptos de la ontología.

Durante la elaboración de la ontología, se ha identificado que el principal tipo de concepto es el de acción. Las acciones, clasificadas en datalógicas e infológicas, permiten realizar un

análisis de distintas facetas o aspectos que participan en su configuración: entradas, resultados, finalidad, verbo, elementos sobre el que se ejecuta el verbo, agente que la ejecuta y agente al cual se dirigen los resultados de las acciones. Este análisis, centrado en acciones y en su configuración en procesos, permite estructurar las relaciones entre los distintos conceptos y entidades de una forma más natural y más enfocada a la resolución al problema de la representación del conocimiento en contextos específicos como el modelado de aplicaciones informáticas y requisitos software. En la expresión de un requisito, el autor debe identificar en primer lugar el verbo o acción que se desea completar, y a partir de allí detallar sus características o aspectos mediante la selección de conceptos procedentes de la ontología. Esta se realimenta a su vez a partir de las relaciones entre conceptos que, de forma indirecta, se establece entre ellos al relacionarlas a través de las acciones o verbos.

## Bibliografía

AMBLER, Scott. Requirements Engineering Patterns: Three approaches to motivate your developers to invest time in taking care of first things first. *Dr. Dobb's ,* 2000. Disponible en línea: http://www.drdobbs.com/architect/184414612, Fecha acceso: 21/01/2012

BOYD, Stephen; ZOWGHI, Didar; GERVASI, Vincenzo. "Optimal-Constraint Lexicons for Requirements Specifications". IN: *REFSQ 2007*, Heidelberg: Springer-Verlag, 2007, p. 203–217 (LNCS 4542)

DURÁN TORO, A.; BERNÁRDEZ JIMÉNEZ, B.; RUÍZ CORTÉS, A.; TORO BONILLA, M. "A Requirements Elicitation Approach Based in Templates and Patterns". In: *Workshop em Engenharia de Requisitos*. Buenos Aires, 1999, p. 17-29

FANTECHI, A.; GNESI, S.; LAMI, G.; MACCARI, A. "Applications of linguistic techniques for use case analysis". IN: *Proceedings of Requirements Engineering 2002*. Society Press, 2002, p. 157-164

KETABCHI, Shokoofeh; SANI, Navid Karimi; LIU, Kecheng. "A Norm-Based Approach towards Requirements Pattern". IN: *35th IEEE Annual Computer Software and Applications Conference*, 2011. DOI 10.1109/COMPSAC.2011.82

LAM, W.; McDERMID, J. A.; VICKERS, A. J. "Ten Steps towards Systematic Requirements Reuse". *Requirements Engineering Journal*, No. 2, 1997, p. 102–113

LAMSWEERDE, Axel van. *Requirements Engineering: from System Goals to UML Models to Software Specifications.* New Jersey: Wiley, 2009

LAMSWEERDE, Axel van. "Requirements Engineering in the Year 00: A Research Perspective". IN: *Software Engineering, 2000. Proceedings of the 2000 International Conference on 2000*, ACM, 2000, p. 5-19

LÓPEZ VILLEGAS, Óscar; LAGUNA, Miguel Ángel. "Requirements Reuse for Software Development". In *Fifth IEEE International Symposium on Requirements Engineering.* 2001

MAJUMDAR, D.; SENGUPTA, S.; KANJILAL, A.; BHATTACHARYA, S. "Automated Requirements Modelling with Adv-EARS". *International Journal of Information Technology Convergence and Services (IJITCS)* Vol.1, No.4, 2011. DOI : 10.5121/ijitcs.2011.1406

NAISH, James; ZHAO, Liping. "Towards a Generalised Framework for Classifying and Retrieving Requirements. Patterns". IN: 2011 First International Workshop on Requirements Patterns (RePa). IEEE (2011), p. 42-51

PERIYASAMY, K.; CHIDAMBARAM, J. "Software Reuse Using Formal Specification of Requirements". IN: *Proceedings of the 1996 conference of the Centre for Advanced Studies on Collaborative research CASCON '96*. ACM (1996)