TESIS DOCTORAL

DESARROLLO DE UN SERVICIO DE NOTIFICACIÓN DE CAMBIOS EN UNA BASE DE DATOS DE GESTIÓN DE LA CONFIGURACIÓN MEDIANTE PROGRAMACIÓN GENERATIVA

José Ramón Coz Fernández

Licenciado en Ciencias Físicas.

Escuela Técnica Superior de Ingeniería Informática.

Departamento de Ingeniería de Software y Sistemas Informáticos

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

2011

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA
Escuela Técnica Superior de Ingeniería Informática
Departamento de Ingeniería de Software y Sistemas Informáticos

---

DESARROLLO DE UN SERVICIO DE NOTIFICACIÓN DE CAMBIOS EN UNA BASE DE DATOS DE GESTIÓN DE LA CONFIGURACIÓN MEDIANTE PROGRAMACIÓN GENERATIVA

Memoria que presenta para obtener el grado de Doctor

José Ramón Coz Fernández

Licenciado en Ciencias Físicas por la Universidad de Cantabria.

Directores:

José Antonio Cerrada Somolinos

Catedrático de Universidad del Departamento de Ingeniería de Software y Sistemas Informáticos de la Universidad Nacional de Educación a Distancia.

Rubén Heradio Gil

Profesor del Departamento de Ingeniería de Software y Sistemas Informáticos de la Universidad Nacional de Educación a Distancia.

Agradecimientos

En primer lugar, quisiera agradecer al personal del Departamento de Ingeniería de Software y Sistemas Informáticos de la Universidad Nacional de Educación a Distancia por su profesionalidad y buen hacer.

Una mención especial y muy destacada requieren José Antonio Cerrada y Rubén Heradio Gil, directores de esta tesis, y el profesor David Fernández; sus estudios e investigaciones en la materia y su dedicación constituyen una parte esencial de este trabajo.

A mi mujer Julia Portilla por su apoyo incondicional, en todas sus facetas.

A mis padres José Ramón y Olga por su apoyo afectivo.

A mi hermano Alberto y a Diego Herranz que han sido siempre un ejemplo destacado de profesionalidad investigadora.

A mis compañeros de trabajo Alejandro Garrido, Enrique Díez Bejerano, Enrique Fojón y Alfonso Peña Mari, por su excelente compañerismo, y al Almirante Miguel A. Beltrán por todo lo que me ha enseñado profesionalmente.

Por último, quisiera agradecer a los Autores que menciono en esta Tesis por todo lo que han aportado a este trabajo.

Muchas gracias.

"La mayoría de las ideas fundamentales de la ciencia son esencialmente sencillas y, por regla general, pueden ser expresadas en un lenguaje comprensible para todos."

Albert Einstein.

Resumen

La importancia de una gestión rentable orientada a los usuarios y de la mejora continua de los servicios relacionados con las TIC (Tecnologías de la Información y las Comunicaciones) son cruciales para garantizar la supervivencia de las empresas y organizaciones.

En nuestros días la gestión TIC supone un gran desafío para todas las organizaciones, que hacen uso de varios marcos, normas y reglamentos para llevar a cabo su gestión. De todos ellos, ITIL (Infrastructure Technology Library) es el marco más aceptado. La gestión de la configuración, uno de los procesos más relevantes de ITIL, proporciona un servicio a través de la identificación, control, mantenimiento y verificación de los elementos de configuración (CI, Configuration Items).

Los CI representan los ejecutables, el código fuente, los modelos de datos y procesos, la documentación y cualquier elemento de la infraestructura que sea susceptible de ser gestionado. La CMDB (Configuration Management Database) reside y permite el acceso a todos los CI y proporciona visibilidad en las dependencias entre procesos de negocio, usuarios, aplicaciones e infraestructura de TIC.

En la actualidad, los servicios de comunicación entre la CMDB y el resto de los procesos de gestión de TIC no están normalizados.

Un servicio de comunicación destacado es la notificación de cambios (SNC, Service Notification Changes). Este servicio, que permite informar sobre los cambios que se producen en los CI, cobra una gran relevancia para ciertos procesos como la Gestión de Incidentes, la Gestión de Problemas, la Gestión de Activos o la Gestión de Gobierno TI, que interactúan con la CMDB.

Las bases de datos disponen de diversas librerías, lenguajes de programación extendidos y entornos de desarrollo que permiten dar soporte al desarrollo de este servicio, no obstante es necesario desarrollar productos software ad hoc que den respuesta a los requerimientos que se planteen en cada caso de estudio y el esfuerzo en su desarrollo es alto.

El principal objetivo de la Tesis es normalizar el desarrollo del SNC. Para cumplir este objetivo, se determina un proceso de desarrollo orientado a la construcción de líneas de productos software, haciendo uso de la programación generativa y se construye un entorno de desarrollo y pruebas que nos permite obtener todos los productos software que nos den soporte a este servicio.

Además, se define un modelo económico de costes, cuyo principal objetivo es obtener la rentabilidad de la solución ofrecida por nuestro proceso de desarrollo.

Este modelo económico y el proceso de desarrollo planteados pueden dar cobertura a proyectos con un mayor alcance, dentro de la programación generativa y el desarrollo de bases de datos y CMDBs.

Abstract

The importance of cost management and user-oriented continuous improvement of services related to the ICT (Information and Communications Technologies) are crucial to assure the survival of businesses and organizations.

Today, the ICT represents a huge management challenge for all organizations ~~that~~ which make use of multiple frameworks, rules and regulations to carry out their management. From all of them, ITIL (Infrastructure Technology Library) is the most widely accepted framework.

Configuration Management, one of the most important ITIL processes, provides services through the identification, control, maintenance and final checking of Configuration Items (CIs).

Any CI may represent the executables, source code, data models and processes, documentation items and any element of infrastructure that is capable of being managed.

The CMDB (Configuration Management Database) enables access to all CIs and provides visibility on the dependencies among business processes, users, applications and ICT infrastructure.

Currently, the communication services between the CMDB and the rest of the ICT management processes are not standardized.

An outstanding media service is the Service of Notification of Changes (SNC). This service, which enables any information about the changes produced on the CIs, is of great importance for certain processes such as Incident Management, Problem Management, Assets Management and IT Governance Management, all of which interact with the CMDB.

Today, the databases make use of a wide range of libraries, extended programming languages and development environments that enable to provide support to the development of this service, however it will be necessary to develop ad hoc software products to respond to the requirements related to each case study and the effort for their development is high as well.

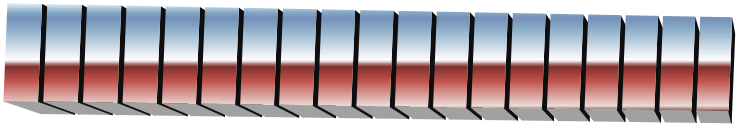
The main objective of this thesis is to standardize the development of this SNC.

To achieve this goal, a development process oriented to software product lines will be defined, making use of generative programming. A development and testing environment will also be built to get all the software products that will provide support to this service.

Furthermore, a cost economic model will be defined, whose main objective will be to obtain the ROI (Return On Investment) of the solution provided by our development process.

These proposed economic model and development process could provide a base for future wider scope projects within generative programming and database and CMDBs development.

Índice General



Contenido General

Índice Detallado

Contenido Detallado

Lista de Símbolos

ANTLR *ANother Tool for Language Recognition*

BC *Bussines Case*

BCC *Búsquedas sin criterio*

BSC *Búsquedas con criterio*

CAB *Core Actives Base*

CDM *Common Data Model*

CI *Configuration Item*

CIM *Common Information Model*

CMDB *Configuration Management Database*

CMS *Configuration Management System*

COBIT *Control Objectives for Information and related Technology*

DDM *Discovery and Dependency Mapping tool*

DECV *Desencolado con Esperas*

DEI *Desencolado con Esperas Indefinidas*

DSE *Desencolado Sin Esperas*

DSL *Domain Specific Language*

DVI *Desencolado con Visibilidad Inmediata*

DVT *Desencolado con Visibilidad Transaccional*

EDD *Exemplar Driven Development*

EFL *Exemplar Flexibilitation Language*

EODAM *Exemplar Oriented Database Methodology*

ERP *Enterprise Resource Planning*

EVI *Encolado con Visibilidad Inmediata*

EVT *Encolado con Visibilidad Transaccional*

FAP *Función Asignación de Permisos*

FBD *Función Borrado de Disparadores*

FBOM *Función Borrado de Operaciones Masivas*

FCC *Función Creación de Colas*

FCD *Función Creación de Disparadores*

FCFC *Función Configuración de la Cola*

FCMN *Función Creación de los Mensajes de Notificación*

FCS *Función Creación de Suscriptores*

FD *Función Desencolado*

FDD *Función Deshabilitación de Disparadores*

FDP *Función Denegación de Permisos*

FE *Función Encolado*

FEC *Función Eliminación de las Colas*

FES *Función Eliminación de Suscriptores*

FFOM *Función Fin de Operaciones Masivas*

FHD *Función Habilitación de Disparadores*

FIOM *Función Inicio de Operaciones Masivas*

FK *Foreign Key*

FN *Forma Normal*

FODA *Feature-Oriented Domain Analysis*

FOMDD *Feature Oriented Model Driven Development*

FOP *Feature Oriented Programming*

FORM *Feature-Oriented Reuse Method*

FRSEB *Feature Reuse-Driven Software Engineering Business*

GEMS *Generic Eclipse Modeling System*

GF *Granularidad de Grano Fino*

GG *Granularidad de Grano Grueso*

GM *Granularidad de Grano Medio*

GMF *Graphical Modeling Framework*

GP *Generative Programming*

GTCR *Gestión del Tiempo con Retención*

GTCRI *Gestión del Tiempo con Retención Infinita*

GTE *Gestión General del Tiempo con Expiración*

GTER *Gestión General del Tiempo con Expiración y Retraso*

GTR *Gestión General del Tiempo con Retraso*

GTS *Gestión General del Tiempo Sencilla*

GTSR *Gestión del Tiempo sin Retención*

ITIL *Information Technology Infrastructure Library*

JUDE *Java and UML Developers' Environment*

LGLP *GNU General Public License*

MCA *Mensajes Con agrupación*

MCP *Mensajes Con Prioridad*

MDD *Model Driven Development*

MNN *Navegación Modo Normal, sin afectar a los mensajes*

MSA *Mensajes Sin agrupación*

MSP *Mensajes Sin Prioridad*

NFR *Non Functional Requirements*

NMBL *Navegación Modo Bloqueo de mensajes*

NMBO *Navegación Modo Borrado de mensajes*

NPM *Navegación Primer Mensaje*

NSM *Navegación Siguiente Mensaje*

NST *Navegación Siguiente Transacción*

NTF *Neutral Tree Feature*

NTFE *Neutral Tree Feature Easy*

OODA *Object Oriented Domain Analysis*

OVM *Orthogonal Variability Method*

PK *Primary Key*

PRINCE2 *Project in Control Environments*

RAA *Ruby Application Archive*

RDBMS *Relation Database Management System*

RUP *Rational Unified Process*

SIMPLE *Structured Intuitive Model for Product Line Economics*

SLA *Service Level Agreement*

SM *Subscriptores Multiples*

SPD *Servicio con Persistencia y en Diferido*

SPL *Software Product Lines*

SQL *Structured Query Language*

SSP *Servicio Sin Persistencia*

SU *Subscriptor Único*

TADDM *Tivoli Application and Dependency Manager*

TDSL *Transformador de DSL*

TIC *Tecnologías de la Información y las Comunicaciones*

UML *Unified Modelling Language*

VFD *Varied Feature Diagram*

Índice de Figuras

Índice de Tablas

Índice de Fórmulas

1. Introducción

Nothing is really unprecedented. Faced with a new situation, people liken it to familiar ones and shape their response on the basis of the perceived similarities.
M.S. Mahoney. The Roots of Software Engineering.

Este capítulo ofrece un breve resumen teórico sobre la Gestión de las TIC (Tecnologías de la Información y las Comunicaciones), las buenas prácticas asociadas como ITIL (Infrastructure Technology Library) y el concepto de CMDB (Configuration Management Database).

Posteriormente, se plantean los retos más importantes que se marca la tesis, dentro de este ámbito.

A continuación, se desarrollan los conceptos más relevantes relacionados con la tesis como el Servicio de Notificación de Cambios, que permite transmitir los cambios que se producen en la infraestructura TIC en una organización a los diferentes procesos de negocio relacionados con la Gestión TIC, las líneas de productos software, como instrumento para el desarrollo de este servicio y los modelos de costes, como herramienta para obtener la rentabilidad y los costes asociados.

Los objetivos principales a cubrir por la tesis son planteados en la sección siguiente.

Finalmente, se exponen la estructura y organización completas de la tesis.

1.1.La Gestión TIC y la CMDB

Un aspecto clave para garantizar la rentabilidad y la prestación de servicios y productos de calidad por parte de las organizaciones y las empresas es disponer de unos servicios relacionados con las TIC orientados a los usuarios y asociados a una mejora continua.

Para llevar a cabo estos servicios, las empresas y organizaciones hacen uso de varios marcos, normas, estándares y reglamentos. De todos ellos, se desataca el marco de buenas prácticas **ITIL** (**Infrastructure Technology Library**), como el más aceptado en la actualidad [Masarat et al. 2009]. **ITIL** establece buenas prácticas para los procesos de negocio relacionados con la gestión de las TIC.

Dentro del marco de procesos de ITIL, la **gestión de la configuración** es uno de los procesos más relevantes, constituyéndose como un proceso clave que permite la identificación, control, mantenimiento y verificación de los **elementos de configuración** (CI, Configuration Items) [Sharifi et al. 2008]. Los CI representan los ejecutables, el código fuente, los modelos de datos y procesos, la documentación y cualquier elemento de la infraestructura que sea susceptible de ser gestionado.

Dentro de este proceso destaca el concepto de **CMDB** (**Configuration Management Database**), como la Base de Datos que reside y permite el acceso a todos los CI [MOF 2005] [Gartner 2006] y que proporciona visibilidad en las dependencias entre los procesos de negocio, los usuarios, las aplicaciones y la infraestructura de TIC [Sarah et al. 2009].

En la actualidad, los servicios de comunicación entre la CMDB y el resto de los procesos de gestión de TIC no están normalizados. Un servicio de comunicación destacado es la **notificación de cambios** (**SNC, Service for the Notification Changes**).

Este servicio permite comunicar los cambios que se producen en los CI, y cobra una gran relevancia para ciertos procesos como la Gestión de Incidentes [Gupta et al. 2008], la Gestión de Problemas [InfoWorld Oct. 2006], la Gestión de Activos [Sharifi et al. 2008] o la Gestión de Gobierno TI [Cobit 4.1 2007] [Sharifi et al. Cobit 2008]. Todos estos procesos interactúan con la CMDB.

Para implementar este servicio, en la actualidad los productos software de soporte a las bases de datos disponen de diversas librerías, lenguajes de programación extendidos y entornos de desarrollo integrados que permiten facilitar su desarrollo. No obstante, es necesario desarrollar productos software *"ad hoc"* que den respuesta a los requerimientos que se planteen en cada caso de estudio y el esfuerzo en su desarrollo puede llegar a ser muy alto, sobretodo en el caso de proyectos que cubran un gran alcance [Coz et al. 2008].

Uno de los principales **objetivos de la Tesis será normalizar el desarrollo de este servicio (SNC).** Para llevar a cabo este objetivo, se definirá un proceso de desarrollo normalizado, orientado a la construcción de líneas de productos software, y que hará uso de la **programación generativa** [Czarnecki 2000]. Una vez determinado el proceso, se construirá un entorno de desarrollo y pruebas que nos permitirá obtener los productos software que nos den soporte a este servicio.

Con el objetivo de obtener la rentabilidad de la solución ofrecida por nuestro proceso de desarrollo, se **define un modelo económico de costes**. Este modelo económico pone de manifiesto la gran rentabilidad de la propuesta planteada. Tanto el modelo económico como el proceso de desarrollo, que incluyen diferentes ámbitos como el estudio de la normalización en BBDD y en una CMDB, la presentación de nuevas herramientas y modelos para la gestión de la variabilidad en líneas de productos software, pueden dar cobertura a proyectos con un mayor alcance dentro de la programación generativa y la construcción de CMDBs.

1.2.Planteamiento de Retos

Para cumplir el objetivo principal de la tesis, y dada la diversidad de aplicaciones prácticas relacionadas con dicho objetivo, **se plantean como novedad una serie de retos:**

1) **Definir el dominio bajo estudio**: el desarrollo de un **servicio de comunicación de los cambios que se producen en una CMDB** hacia un conjunto de procesos relacionados con la gestión de las TIC, que actuarán como suscriptores de dicho servicio.

2) **Normalizar el proceso de diseño y desarrollo** que se va a llevar a cabo para la creación del dominio, que incluirá el diseño y la construcción de una CMDB y del servicio SNC (Service for the Notification Changes).

3) **Construir un entorno de desarrollo y pruebas** que nos facilite la obtención de todos los productos software necesarios para dar soporte al servicio de comunicación de los cambios que se producen en la CMDB. Acotaremos este dominio determinando, tras un análisis de requisitos detallado, un caso de estudio con suficiente alcance, que incluya los requerimientos más habituales y los procesos negocio con mayor impacto en su interacción con la CMDB. Este caso de estudio nos permitirá poner a prueba nuestro entorno de desarrollo y nuestro proceso normalizado.

4) Hacer un **estudio económico que estime la rentabilidad** de la solución propuesta y aplicarlo al caso de estudio planteado.

1.2.1. Servicio de Comunicación de Cambios

En el caso de una CMDB, disponer de un servicio de comunicación de los cambios (SNC) cobra una gran relevancia. A raíz de las mejores prácticas que ofrece el marco ITIL, la CMDB es una base de datos federada, lo que significa que no todos los datos de configuración deben residir en una misma base de datos física [Clark et al. 2007].

Los sistemas que soportan la gestión TIC y los repositorios que contienen los datos son las fuentes autorizadas de información de la organización, mientras que la CMDB se convierte en una referencia para la residencia y el acceso a los CI [Larry 2010].

ITIL, en su actual versión 3, reconoce la importancia de este enfoque, y **recomienda el uso de una CMDB federada** como una parte fundamental de la estructura de un CMS (Configuration Management System) [Jan Van Bon ST 2008]. Con la federación, los datos básicos se almacenan en la CMDB, que está vinculada a otros repositorios de datos, con información más detallada. La federación permite el acceso, a través de la CMDB, a todos los CI. A partir de esta fuente de información (la CMDB), **cuando se produce un cambio en un elemento de configuración, este cambio puede tener un impacto en la organización y otros procesos** relacionados con la CMDB.

Los procesos de negocio serán apoyados por un número de sistemas de información, como el sistema de gestión de activos o el sistema de gestión de incidentes y problemas. Estos sistemas pueden actuar como suscriptores del SNC. **Este servicio (SNC) permitiría que los cambios necesarios se notifiquen a los sistemas pertinentes.**

1.2.2. Normalización del proceso de desarrollo

El SNC tiene una gran dependencia con la CMDB. Como ya se ha mencionado en la sección anterior, la CMDB puede estar formada por un conjunto de bases de datos físicas, en forma de federación. Dependiendo de la arquitectura de datos disponible en la organización, el desarrollo del SNC podrá ser más o menos complejo. Esta variabilidad implica que el alcance de nuestro dominio puede ser bastante amplio.

Otro aspecto que influye notablemente en el alcance del SNC son los procesos de negocio de soporte a la gestión TIC. Cada uno de estos procesos podrá estar apoyado por un conjunto de sistemas de información, que podrán actuar como suscriptores de este servicio.

Por último, tenemos el conjunto de requisitos a cubrir por nuestro servicio. El SNC podría cubrir un conjunto muy reducido de requisitos si, por ejemplo, nuestro servicio no tiene requerimientos de prioridades en los mensajes, de gestión del tiempo, de agrupación de los mensajes, de múltiples suscriptores, etc. En otro extremo, podría nuestro SNC cubrir un gran número de requerimientos como dotar a los mensajes de notificación de diferentes prioridades, de retrasos, de retenciones, de agrupación de mensajes, gestionar múltiples suscriptores, ofrecer diferentes mensajes con mayor o menor detalle, etc.

Todos estos factores dotan a nuestro dominio de una gran variabilidad. Uno de los retos que nos planteamos es Normalizar el desarrollo de este servicio para ser capaces de dar una solución global que cubra cualquier arquitectura de datos, procesos de negocio y configuración de requisitos.

Las líneas de productos software (SPL, Software Products Line) ofrecen un marco tecnológico para el tipo de problemas que estamos tratando de resolver: la creación de productos similares pertenecientes a un mismo dominio. En nuestro caso, nuestro dominio estará constituido por los productos software que nos den soporte al servicio SNC de la CMDB.

Hay ejemplos bien documentados de reducción de costes y de la mejora de calidad que se consigue mediante la introducción del paradigma de SPL en la industria [Dinnus y Pohl 2005] [Sommerville 2006] [Pohl et al. 2005].

La reutilización del software dentro de un dominio de aplicación pasa por el descubrimiento de elementos comunes a los sistemas pertenecientes a dicho dominio. Se produce un cambio de un desarrollo orientado a un único producto software a un desarrollo centrado en varios productos que comparten unas características formando una familia.

Aparecen dos procesos distintos: la **Ingeniería de Dominio**, que se centra en el desarrollo de elementos reutilizables que formarán la familia de productos y la **Ingeniería de Aplicación**, que se orienta hacia la construcción o desarrollo de productos individuales, pertenecientes a la familia de productos, y que satisfacen un conjunto de requisitos y restricciones expresados por un usuario específico.

El objetivo fundamental de la ingeniería de dominio es la optimización del proceso de desarrollo del software en un espectro de múltiples aplicaciones que representan un negocio común o problema de dominio [Griss, M. L. 1996] [Simos, M. 1995], como es nuestro caso. Dentro del marco de la ingeniería de dominio, una propuesta destacada es la **Programación Generativa** [Czarnecki 2000].

En el paradigma del **desarrollo dirigido por modelos** (MDD, Model Driven Development), los modelos se utilizan para especificar los programas, y sobre éstos se efectúan transformaciones para sintetizar los ejecutables [Sami et al. 2005]. Por otra parte, la **programación orientada a características** (FOP, Feature Oriented

Programming) es un paradigma para líneas de productos software, donde los programas se sintetizan componiendo características (features). Feature Oriented Model Driven Development es una combinación de los dos anteriores, donde los programas de una SPL se pueden sintetizar componiendo modelos a partir de características y luego transformando estos modelos en ejecutables [FOP 2007].

La programación generativa y el desarrollo dirigido por modelos, ambos, consideran que el aumento de la productividad en la realización de software pasa por elevar el nivel de abstracción de los lenguajes de programación mediante el uso de especificaciones o modelos. Un factor clave para el éxito de estos paradigmas es la traducción automática de los modelos a código ejecutable. Para que la traducción automática sea viable, el dominio de aplicación de los modelos debe reducirse hasta que la variabilidad entre los productos que puedan generarse sea significativamente inferior a los aspectos comunes.

Uno de los enfoques utilizados es aplicar transformaciones sobre la especificación de los productos del dominio. Aunque existen lenguajes específicos para la expresión de transformaciones, exigen superar una curva de aprendizaje notable y carecen de algunas prestaciones típicas de los lenguajes de programación.

Existe una **iniciativa de investigación que**, con el propósito de aprovechar la proximidad entre los productos que pueden generarse dentro de un dominio, **propone que la aplicación de transformaciones no se realice sobre la especificación, sino sobre alguno de los productos**. Es decir, el traductor se define como un programa que toma un producto previamente desarrollado del dominio, al que se identifica como ejemplar, y lo transforma para adecuarlo a una especificación.

Esta iniciativa se define en [Heradio et al. 2005] y en la Tesis doctoral [Heradio 2007], donde se propone un **nuevo proceso de desarrollo de familias de productos, denominado EDD** (**Exemplar Driven Development**), **que aprovecha la similitud entre los productos de una familia para construirlos por analogía**.

Adaptar este proceso de desarrollo de líneas de productos a la construcción de nuestro servicio de comunicación entre la CMDB y los procesos de negocio es otro de los retos que se plantean.

Los productos de nuestro dominio tendrán una dependencia directa con la tecnología y con la CMDB. Trabajar con un producto concreto de base de datos, un lenguaje de programación, una CMDB específica y un conjunto de procesos de negocio susceptibles de recibir las comunicaciones de los cambios que se produzcan en la CMDB implicará un desarrollo ad hoc que variaría en función de todas estas casuísticas. Por ejemplo, si nuestra CMDB cambia, nuestros productos variarán. Uno de los objetivos de nuestro proceso de desarrollo será **permitir que estos cambios no tengan un impacto en el desarrollo de los productos**.

1.2.3. Entorno de desarrollo y pruebas

Durante las primeras fases de nuestro proceso se realizarán una serie de actividades enmarcadas dentro de la **Ingeniería de Requisitos**. Se definirá un proceso propio de análisis de los requisitos del dominio bajo estudio. En este **proceso se utilizarán escenarios** que incluirán los requisitos funcionales de nuestro **dominio y modelos orientados a características** (features) **que permitirán especificar la variabilidad**.

Se realizará un **estudio de las herramientas disponibles y los métodos y técnicas de soporte a la gestión de la variabilidad** que nos faciliten la obtención de las métricas para el desarrollo de un modelo completo económico que nos proporcione la rentabilidad y los costes.

Además, se realizará una **propuesta para normalizar los modelos de análisis de nuestro dominio**.

Una vez definido nuestro proceso de desarrollo, **será necesario construir un entorno de desarrollo y pruebas que nos permita obtener y testear todos los productos software de nuestro dominio**.

Nuestro dominio incluirá, además del desarrollo de una base de datos de gestión de la configuración (CMDB), el desarrollo de una línea de productos software para dar soporte al servicio de notificación de cambios.

1.2.4. Modelo de Costes

El enfoque del desarrollo de líneas de productos no es siempre es la mejor opción económica para el desarrollo de una familia de sistemas relacionados. Los sistemas pueden ser extremadamente diferentes entre sí, o la familia de productos puede contener muy pocos productos para recuperar el coste de desarrollo del "*core asset base*" (la infraestructura común a todos los productos).

Los decisores deben ser capaces de predecir los costes y beneficios de la construcción y la evolución hacia una ingeniería de líneas de productos, en comparación con los enfoques tradicionales de desarrollo software [Clements et al. 2005].

Existen varias **líneas de investigación sobre los aspectos económicos de las líneas de productos** que han conducido al desarrollo de diversos modelos económicos de costes: **COCOMO II** [Boehm et al. 2000], **COPLIMO** [Boehm et al. 2004], **qCOPLIMO** [In et al. 2006] y **SIMPLE** [Clements et al. 2005], entre los más destacados.

Un aspecto esencial de estos modelos, además del cálculo para obtener el coste de la construcción de la línea de productos, es la **obtención de la rentabilidad**. Esto es, no tan solo centrarse en determinar cuánto cuesta construir la línea de productos, sino lo rentable o beneficioso que puede ser su construcción.

Ofrecer un Modelo Económico para el estudio de los costes de la línea de productos que se construya es otro de los retos planteados. Este modelo debería permitir obtener la rentabilidad alcanzada por la solución ofrecida.

1.3.Resumen de Objetivos

Se plantean una serie de objetivos para el desarrollo de la Tesis, agrupados por los retos inicialmente planteados:

- **Estudiar y Definir el Dominio de la notificación de cambios en bases de datos y su aplicación a una CMDB**, **siguiendo las buenas prácticas de ITIL**. Otro objetivo complementario será analizar con detalle el dominio bajo estudio, determinando sus objetivos (goals) y todos los elementos de análisis necesarios.

- **Diseñar una metodología específica de construcción de SPL**, **orientada a la programación generativa y enmarcada en la ingeniería de dominio**, que nos permita la construcción de nuestra SPL.

- **Integrar dicha metodología de desarrollo con las buenas prácticas internacionales de gestión de proyectos** como PRINCE2 [PRINCE2 2009] nos permitirá dar un soporte necesario de gestión y control a la metodología planteada.

- **Adaptar el proceso metodológico para la construcción de bases de datos de gestión de la configuración** (CMDB), estudiando las diferentes iniciativas actuales de diseño y desarrollo de CMDBs y proponer una normativa para el diseño de nuestra CMDB.

- **Definir un entorno de desarrollo y pruebas para la construcción de SPL, orientado a la programación generativa**. Además, el otro objetivo será **construir este entorno y aplicarlo a un caso de estudio específico**, **que nos permita cubrir un gran alcance.**

- **Desarrollar un modelo de costes para medir la rentabilidad** de la solución propuesta.

1.4.Metodología

Para abordar el proyecto de la construcción de nuestro servicio, se definirá, en primer lugar, una **metodología como extensión de EDD** (Exemplar Driven Development) [Heradio et al. 2005] [Heradio 2007], **un nuevo proceso de desarrollo de familias de productos, que trate de aprovechar la similitud entre los productos software de soporte al servicio SNC de una CMDB, para construirlos por analogía**.

El **proceso se descompondrá en tres actividades básicas**. En primer lugar, **la construcción de un producto concreto de la familia**, que se denomina **ejemplar**; en segundo lugar, **la realización de una infraestructura** con ocultación de caja negra, que facilite la obtención posterior de los productos (esta infraestructura se desarrollará flexibilizando el ejemplar, es decir, definiendo la relación de analogía que permitirá derivar automáticamente del ejemplar todos los productos de la familia) y, en tercer lugar, **la obtención de cada producto parametrizando la infraestructura.**

Durante las primeras fases del proceso, toda la **información de análisis del dominio** se recopilará en los modelos del dominio, y siguiendo las recomendaciones de [Czarnecki et al. 2000] y [Greenfield and Short 2004], **se utilizará un modelado de características.** Además, se incluirá un modelo de objetivos (*goals*) del dominio bajo estudio, donde los objetivos a cubrir serán de tipo *hard*, entendidos como objetivos funcionales.

En lo que se refiere al modelado de la CMDB, **se definirá un estándar que nos permita normalizar los modelos conceptuales, lógicos y físicos de la CMDB**. Para ello, se estudiarán las normas sobre el uso de objetos en una CMDB, la documentación obligatoria, las siglas, los acrónimos, los dominios, las reglas de validación, las relaciones y las transformaciones entre los modelos lógicos, físicos y conceptuales, siguiendo las directrices marcadas por [Kleppe et al. 2003] para la transformación de modelos.

Para construir el servicio de notificación de cambios**, se definirá un alcance determinado, focalizándose en aquellos procesos de negocio con mayor impacto en la CMDB**, como la **Gestión de Incidentes** [Gupta et al. 2008], la **Gestión de Problemas** [InfoWorld Oct. 2006], la **Gestión de Activos** [Sharifi et al. 2008] o la **Gestión de Gobierno TI** [Cobit 4.1 2007] [Sharifi et al. Cobit 2008].

La última actividad del proceso será el **desarrollo de un entorno de desarrollo y pruebas** que incluya unos generadores que, a partir del ejemplar y de un lenguaje específico del dominio (DSL), nos permitan obtener el resto de los productos del dominio.

Para la implementación de estos transformadores se utilizará el lenguaje EFL (**Exemplar Flexibilitation Language**) tal y como se propone en [Coz et al. 2008] [López et al. 2008] [Heradio et al. 2008]. La implementación actual de EFL corresponde a un lenguaje específico del dominio embebido en Ruby [Thomas et al. 2004].

La implementación de este lenguaje se distribuye a través de una licencia de tipo LGLP (GNU General Public License) [LGPL 2007] y está disponible en diferentes repositorios como Ruby Forge o RAA [EFL 2007].

Posteriormente a la obtención de los productos del dominio, se considera relevante la realización de un **análisis detallado de los costes**, incluyendo un **estudio de la rentabilidad ofrecida por nuestra propuesta**. Para ello, y sobre la base de los modelos económicos COCOMO II [Boehm et al. 2000], COPLIMO [Boehm et al. 2004] y qCOPLIMO [In et al. 2006], **se definirá el modelo económico** que de soporte a nuestro proceso de desarrollo.

1.5.Organización de la Tesis

La tesis está organizada en once capítulos. **El presente capítulo introductorio ha expuesto el marco general de la tesis**: las buenas prácticas ITIL y la gestión de servicios TIC, la Base de Datos de Gestión de la configuración (CMDB), el servicio de notificación de cambios en bases de datos, las líneas de productos software (SPL) y sus modelos de costes, y la programación generativa.

El **capítulo 2 es un estado del arte** que resume:

- Los logros, problemas y áreas de investigación más relevantes en el entorno en el que se enmarca la tesis:
  - ITIL y la Gestión de Servicios TI,
  - la Base de Datos de Gestión de la Configuración (CMDB) y
  - la Notificación de Cambios en Bases de Datos.
- Las recientes y más destacadas propuestas para el desarrollo de familias de productos: la programación generativa, las metodologías orientadas a líneas de productos y los modelos de costes.

- Las principales aproximaciones para modelar líneas de productos mediante la gestión de la variabilidad, incluyendo un estudio detallado de las técnicas, métodos y herramientas de soporte.

En el capítulo tres se expone una metodología original, denominada EODAM (Exemplar Oriented Database Methodology), **que ofrece un proceso de desarrollo software de líneas de productos para bases de datos, orientado al desarrollo de productos a partir de un ejemplar del dominio.** Además, en este capítulo se propone una adaptación de este proceso para construir desarrollos de bases de datos de gestión de la configuración (CMDB).

En el capítulo cuarto se expone una propuesta de estandarización para una CMDB, incluyendo normas para el diseño conceptual, lógico y físico. Esta propuesta puede ser extendida a cualquier otra tipología de base de datos, realizando la correspondiente adaptación.

A continuación, el resto de capítulos, siguen de forma secuencial la propuesta metodológica de EODAM para construir un dominio específico.

La ingeniería de requisitos y el análisis del dominio bajo estudio, que consiste en el desarrollo de un servicio de notificación de cambios en una CMDB, **se exponen en el capítulo quinto.**

El capítulo sexto realiza una propuesta para la documentación de la variabilidad de un dominio y su aplicación al dominio bajo estudio.

Esta propuesta incluye un conjunto de técnicas y herramientas novedosas que ofrecen una serie de mejoras sustanciales, permitiendo obtener métricas y funcionalidades de gran relevancia para el desarrollo de líneas de productos software.

En el **capítulo séptimo se lleva a cabo el desarrollo de la CMDB**, según la propuesta metodológica ofrecida en el capítulo dos, y se **obtienen las principales métricas del dominio**. En este capítulo **se analiza**, también, **la integración de EODAM con las buenas prácticas de gestión de proyectos**, como PRINCE2. Además, se realiza un estudio de las diferentes propuestas existentes en el mercado de diseño y construcción de las CMDB.

El **desarrollo del dominio bajo estudio se describe en el capítulo octavo**, donde se expone un **Framework de desarrollo**, basado en programación generativa, para la construcción del dominio.

La realización de las pruebas de nuestro dominio se describen en el capítulo nueve. Además, en este capítulo se realiza un pequeño análisis de las diferentes pruebas existentes en la ingeniería de software y su relación con la metodología propuesta.

En el **capítulo diez, se ofrece un modelo económico** para nuestra línea de productos. Este modelo tiene como objetivo realizar un estudio de la rentabilidad de la solución ofrecida por el desarrollo realizado. Este modelo puede ser extendido a otros modelos de programación generativa y/o de desarrollo de líneas de productos.

Finalmente, en el **capítulo once, se exponen las principales conclusiones y los trabajos futuros** que pueden desarrollarse a raíz de esta Tesis.

A continuación, se muestra en forma gráfica la estructura de la Tesis.

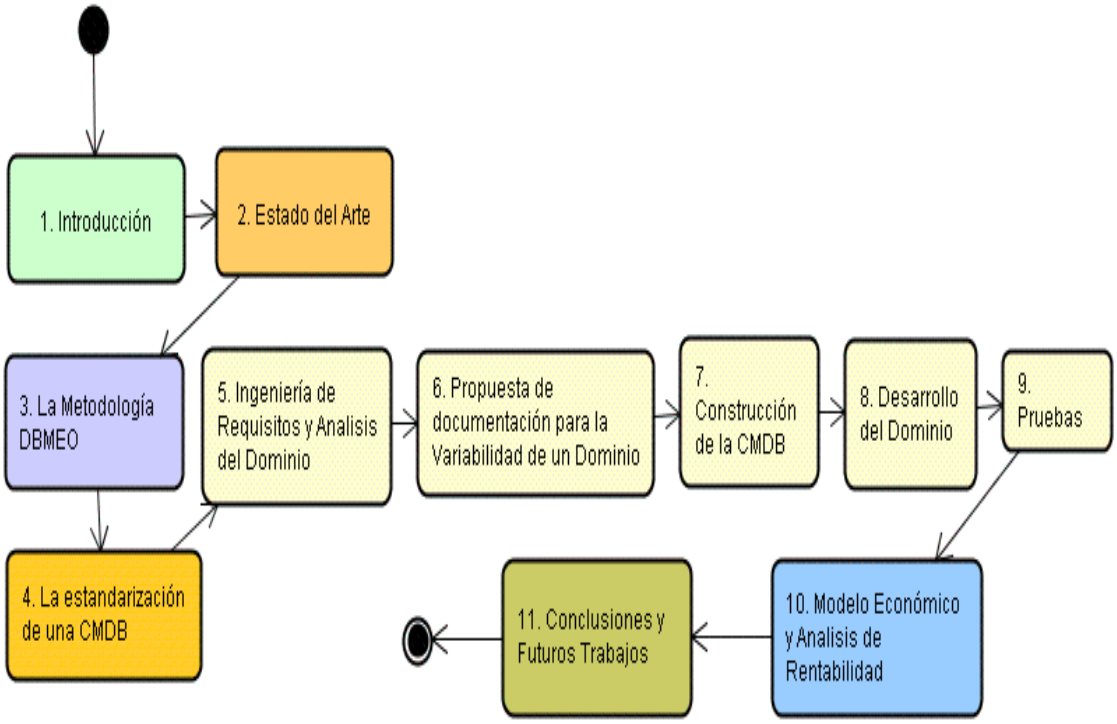


Figura 1-1 Organización de la Tesis.

2. Estado del Arte

"Por norma, los sistemas software
no funcionan bien hasta que han sido utilizados y
han fallado repetidamente en entornos reales"
Dave Parnas.

En el presente capítulo se describe un estado del arte que resume los logros, problemas y áreas de investigación más relevantes en el entorno en el que se enmarca la tesis.

En las primeras secciones se incluyen:

- Los conceptos más relevantes sobre Ingeniería del Software.
- Las buenas prácticas ITIL y la Gestión de Servicios TI.
- La Base de Datos de Gestión de la Configuración (CMDB).
- La Notificación de Cambios en Bases de Datos.

Posteriormente, se hace un breve resumen sobre las recientes y más destacadas propuestas para el desarrollo de familias de productos: la ingeniería de dominio, la programación generativa, las metodologías orientadas a líneas de productos, el proceso de desarrollo EDD (*Exemplar Driven Development*) y los modelos de costes.

Por último, se ofrece un resumen sobre las principales aproximaciones para modelar líneas de productos mediante la gestión de la variabilidad, incluyendo un estudio detallado de las técnicas, métodos y herramientas de soporte.

2.1.Conceptos de Ingeniería del Software

Según la definición del IEEE, citada por [Lewis 1994] *"**software** es la suma total de los programas de computadora, procedimientos, reglas, la documentación asociada y los datos que pertenecen a un sistema de cómputo". Según el mismo autor, "un **producto de software** es un producto diseñado para un usuario".* En este contexto, la Ingeniería de Software es un *"enfoque sistemático del desarrollo, operación, mantenimiento y baja del software".*

Se considera que *"la **Ingeniería de Software** es la rama de la ingeniería que aplica los principios de la ciencia de la computación y las matemáticas para lograr soluciones costo-efectivas (eficaces en costo o económicas) a los problemas de desarrollo de software"*, es decir, "*permite elaborar consistentemente productos correctos, utilizables y costo-efectivos*" [Cota 1994].

El **proceso de ingeniería de software** se define *como "un conjunto de etapas parcialmente ordenadas con la intención de lograr un objetivo, en este caso, la obtención de un producto de software de calidad"* [Jacobson 1998].

El **proceso de desarrollo de software** "*es aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en diseño y el diseño implementado en código, el código es probado, documentado y certificado para su uso operativo"*. Concretamente *"define quién está haciendo qué, cuándo hacerlo y cómo alcanzar un cierto objetivo"* [Jacobson 1998].

El proceso de desarrollo de software requiere, por un lado, un conjunto de conceptos y, por otro, una metodología y un lenguaje propio. A este proceso se le llama **el ciclo de vida del software. En la presente tesis se propone el uso de una metodología específica para la construcción de una serie de productos software, de un lenguaje para la especificación de la variabilidad de los productos y de un lenguaje para la construcción de la línea completa de productos.**

2.2.ITIL y la Gestión de Servicios de TI

Information Technology Infrastructure Library, en su versión 3.0 (en lo sucesivo, **ITIL**) [Simon 2009] y los nuevos conceptos que apoyan la operación de los sistemas de información en las organizaciones como la **Base de Datos de Gestión de la Configuración** (Configuration Management Database, en adelante, **CMDB**) constituyen la columna vertebral de la gestión de las TIC [Gerard 2008].

ITIL constituye hoy día el estándar *"de facto"* en la mayoría de las organizaciones de todos los procesos relacionados con la gestión de servicios de TIC.

La figura 2-1 muestra, en forma gráfica, algunos de los principales procesos. Para todos estos procesos, ITIL proporciona las mejores prácticas que pueden servir como punto de referencia para las organizaciones para mejorar sus servicios.

Figura 2-1 Algunos de los principales procesos ITIL.

En primer lugar, citamos los **servicios de diseño** [Jan Van Bon SD 2008] incluyendo el proceso de gestión de la disponibilidad, entrega, diseño y mantenimiento de los acuerdos de nivel de servicio (SLA's) y el mantenimiento del catálogo de servicios TIC prestados por la organización.

En segundo lugar, los **servicios operativos** [Jan Van Bon SO 2008], que incluyen los procesos de gestión de eventos, problemas e incidentes. Por su interés, mencionaremos que *incidente*, de acuerdo a ITIL, es cualquier evento que cause o pueda causar la interrupción del servicio y el *problema* es la causa subyacente de uno o más incidentes. *Evento* incluye cualquier situación que causa la interrupción del servicio.

Por último, existen los **servicios de transición** [Jan Van Bon ST 2008] que cubren el proceso de gestión del cambio, prueba y validación de sistemas, desarrollo, gestión del conocimiento y gestión de la configuración.

La tesis se enmarca en las buenas prácticas ITIL. En el trabajo presentado se realiza un análisis detallado de algunos de los procesos más importantes y se construye un servicio que permite el intercambio de información entre estos procesos.

2.3.La CMDB

La gestión de la configuración, uno de los principales procesos en los servicios de transición, proporciona un modelo lógico de la infraestructura o servicio a través de la identificación, control, mantenimiento y verificación de elementos de configuración.

Los **Elementos de Configuración** (Configuration Items, **CI**) son los componentes de una infraestructura que estén o vayan a estar bajo el control de la configuración. Los CI son únicos e identificables, están sujetos a cambios y se pueden controlar.

Los CI tienen asociados un conjunto de atributos estándar como categoría, relaciones, atributos, estado e historial.

Para decidir si un elemento es susceptible de constituir un CI es necesario determinar si la organización tiene que gestionarlo para garantizar la entrega adecuada de un servicio de TI. Si ese elemento tiene que ser administrado, entonces debe ser un CI.

Otra forma de identificar si un elemento es un CI es el **test USMC:**

- ¿Es único? (U, unique)
- ¿Es determinante para la entrega de un servicio de TI? (S, service)
- ¿Se puede gestionar? (M, management)
- ¿Tiene, por lo menos, algunas características que pueden cambiar? (C, change)

Si todas las respuestas son afirmativas, se trata de un CI. **La base de datos que contiene todos los datos relevantes de cada uno de los CI de la organización y los detalles de la relación entre ellos, se llama CMDB** (base de datos de gestión de la configuración).

Esta base de datos puede constituirse en el punto de referencia para todas las decisiones de TI y operaciones en la organización y proporcionar visibilidad en las dependencias entre procesos de negocio, usuarios, aplicaciones e infraestructura de TI. La CMDB reside y permite el acceso a todos los CI. Los CI se suelen especificar a un nivel muy detallado dentro de la CMBD [MOF 2005] [Gartner 2006].

Uno de los objetivos de la tesis es la estandarización del desarrollo de una CMDB. En la tesis se realiza una propuesta de estándar para el diseño conceptual, lógico y físico de una CMDB. Además, revisaremos las propuestas más actuales para el desarrollo y diseño de una CMDB.

2.4.La notificación de cambios en Bases de Datos

La **notificación de cambios en bases de datos** es, de forma muy breve, la implementación de un **observador de los cambios** que se producen en la base de datos, de un servicio que **notifica de los cambios** y de una serie de **entidades que se suscriben a dicho servicio.** El servicio de notificación de cambios, en base a una serie de requerimientos, notifica a los suscriptores de los cambios producidos en la base de datos. Nos podemos encontrar con **numerosos escenarios**.

A continuación se muestran algunos posibles ejemplos obtenidos de la bibliografía [Coz et al. 2008] y otros de elaboración propia:

- Una agencia de bolsa ofrece al público su nuevo servicio, que dejará a los clientes estipular el tiempo así como el precio como un parámetro. Por ejemplo, una petición para comprar o vender no es ejecutada a no ser que esto ocurra dentro de un período de tiempo específico (por ejemplo, dentro de 15 minutos). El sistema de información que gestiona las transacciones está soportado por una base de datos y se precisa un control sobre los procesos que superen dichos marcos temporales establecidos por los clientes. Para su desarrollo, ciertas transacciones serán notificadas cuando no cumplan los criterios establecidos, de tal manera que posteriormente puedan ser anuladas.

- Una universidad estatal decide automatizar su proceso de inscripción de cursos. Los estudiantes serán capaces de registrarse utilizando plantillas Web, tanto desde casa como desde los propios recintos universitarios. La administración de la universidad anuncia que los parámetros siguientes se aplicarán: el registro seguirá una prioridad del “primero vendido”, con las siguientes excepciones: el personal de la propia universidad tiene mayor prioridad, seguidos de estudiantes de tercer ciclo. El sistema será soportado

por una base de datos y los registros de compra de los cursos serán notificados al sistema de gestión. Estas notificaciones deberán de seguir las prioridades establecidas por la administración de la universidad.

- Un sistema de control de navegación aérea dispone de unos controles electrónicos que informan a los pilotos de cambios producidos en diferentes aspectos de interés en la navegación. El sistema de control está soportado por una base de datos que recibe información de diferentes ámbitos: meteorológica, de bases de aéreas, de órdenes de vuelo, etc. Cuando se produce un cambio en ciertos parámetros es necesario que el sistema de control avise a los pilotos para facilitar la toma de decisiones durante el vuelo y, de la misma forma, permitan a los sistemas electrónicos del avión ejecutar ciertos procesos.

- Un sistema de información soportado por una base de datos está activo con un horario de lunes a viernes, no festivos. Los sábados no festivos se realizan tareas de mantenimiento. Por las características de la información soportada por el sistema se considera necesario tener un control sobre cualquier cambio producido fuera del horario de uso o de mantenimiento. Por ejemplo, si se realiza una conexión a la base de datos un día festivo se precisa que un responsable delegado sea informado.

- Un grupo empresarial dispone de un sistema de información común soportado por una base de datos con información sobre vacantes. Se desea que los departamentos de recursos humanos reciban notificaciones inmediatas cuando ciertas vacantes queden disponibles. Además, se considera necesario recibir las notificaciones con cierto retraso para algunas empresas del grupo. Se podría tratar de un proyecto de implantación de un ERP de recursos humanos de un grupo multinacional.

Como se puede observar, los elementos básicos de estos escenarios son muy comunes. Cada uno de estos escenarios describe una situación en la cual los mensajes se intercambian entre múltiples clientes (nodos) en un ambiente distribuido. Los mensajes no solamente se intercomunicarán entre el cliente y el servidor, sino también entre procesos del propio servidor.

Si enfocamos estos escenarios en términos de mensajes, los usos pueden ser vistos como procesos en los cuales cada paso es provocado según uno o varios mensajes, y da lugar a uno o varios mensajes. Otro modo de decir esto consiste en que los mensajes son los acontecimientos que provocan otros acontecimientos de mensaje. En estos escenarios se considera fundamental conservar los requisitos de seguridad adicionales como la disponibilidad, integridad, etc.

Además, la persistencia de los mensajes, y los requisitos adicionales de prioridad de los mensajes, retrasos, expiración, etc. son un complemento que permiten dar una solución robusta a las problemáticas planteadas.

La tesis propone el desarrollo de una línea de productos que de soporte a este dominio específico: **el servicio de notificación de cambios en bases de datos.**

2.5.La notificación de cambios en una CMDB

Como ya se ha mencionado con anterioridad, en el caso de una CMDB, la notificación de cambios tiene una gran relevancia. A raíz de las mejores prácticas que ofrece el marco ITIL, los procesos como la Gestión de Incidentes, la Gestión de Problemas, la Gestión de Versiones, la Gestión de Continuidad, la Gestión de la

Disponibilidad, el Control de Cambios, etc. estarán soportados por una serie de sistemas de información.

Los cambios que se produzcan en los elementos de configuración (CI) pueden llegar a tener un gran impacto en estos procesos y, por tanto, la comunicación de estos cambios a estos sistemas de información se convierte en una necesidad. Estos sistemas podrán actuar como suscriptores de este servicio, que será el responsable de gestionar estas comunicaciones.

Por tanto, la implementación de un servicio que permita observar estos cambios y notificar a los diferentes sistemas constituye un reto importante. La tesis propone el desarrollo de una línea de productos que de soporte a este servicio. Además, en la tesis se analizarán los posibles suscriptores y los diferentes procesos involucrados.

2.6.Las Líneas de Productos SW

La ingeniería de Líneas de Productos Software (SPL) permite la transición desde el desarrollo de una serie de soluciones ad hoc hacia la construcción automatizada de una cartera completa de productos de software. La idea fundamental del enfoque es proceder al desarrollo de un conjunto de productos como una sola tarea. Los productos se construyen a partir de un **"core" base de activos (CAB)** y una colección de artefactos que se han diseñado específicamente para su uso en toda la cartera de productos.

Clements y Northrop [Clements y Northrop 2001] definen las *líneas de productos software* de la siguiente forma: *"A Product Line is a set of software intensive systems, sharing common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way".*

Las líneas de productos software ofrecen un marco tecnológico para el problema que estamos tratando de resolver: la creación de productos muy similares pertenecientes a un mismo dominio. En nuestro caso, el servicio de comunicación de los cambios que se producen en una CMDB.

La tesis propone un Proceso de Desarrollo orientado a la construcción de líneas de productos SW que puede cubrir proyectos de mayor alcance y con otros requerimientos.

2.7.Las metodologías para líneas de productos SW

Existe un gran variedad de **metodologías software para construir líneas de productos**: PLP [Clements and Northrop 2002], PULSE [Bayer et al. 1999], FORM [FORM 1998], FAST [Weiss 1999] o KOBRA [Atkinson et al. 2002].

Los procesos software [Pohl et al. 2001] [Linden 2002] derivados de los proyectos europeos ESAPS y CAFÉ [ESPAPS-CAFE] coinciden en diferenciar **tres macro-procesos dentro de las metodologías** para crear *líneas de productos*: la ***ingeniería del dominio, la ingeniería de la aplicación y las actividades de gestión del proceso***. ESAPS sienta los conceptos sobre la familia de productos, mientras que CAFÉ enfatiza la puesta en práctica de estos conceptos.

Estas metodologías no nombran de la misma forma a estos macro-procesos, pero sí que coinciden en el propósito de cada uno de ellos.

Incluimos una breve descripción del propósito de cada macro-proceso:

- El proceso de ***ingeniería del dominio*** reúne las actividades de creación de la *plataforma,* y comprende el desarrollo del conjunto de activos reutilizables (requisitos, diseño, implementación, pruebas…). En este proceso se define qué aspectos comunes y variables presenta la *línea de productos*.

- El proceso de ***ingeniería de la aplicación*** reúne actividades que tienen el objetivo de derivar los productos a partir de la infraestructura de la *línea de productos*, usando la *plataforma* y recursos elaborados en la *ingeniería del dominio*.

- El ***proceso de gestión*** es responsable de gestionar la *ingeniería del dominio* y la *ingeniería de la aplicación*, pero las actividades en este proceso enfatizan el papel que juega la organización y el entorno, en la creación de las *líneas de productos*.

En estos procesos se separa la construcción de la *plataforma* de la *derivación de los productos* individuales. No obstante, para que sean efectivos, debe de existir una sinergia entre ambos y la *plataforma* debe de ser lo suficientemente flexible como para *derivar los productos* de la *línea de productos*.

En la *ingeniería de la aplicación* se utiliza la plataforma para derivar los productos. Por otra parte, la *ingeniería del dominio* recibe realimentación de la *ingeniería de la aplicación* con el propósito de incorporar cambios en la *plataforma.* El propósito de las *líneas de productos* no es crear un único producto, sino disponer de la capacidad potencial de crear un conjunto o familia de productos.

Los clientes que solicitan un producto necesitan una forma de describir y distinguir los elementos que debe de tener el producto. En inglés a estas unidades se les llama

*"**features**"*. Las *líneas de productos* se gestionan en términos de un conjunto de *"features"*. Algunas de las "features" serán comunes para todos los productos potenciales de la línea y otras "features" sólo se encontrarán en algunos productos. Los productos que una *línea* permite construir corresponden a un mismo *dominio*, o *segmento de mercado común*. Un **dominio** es un cuerpo de conocimiento especializado, un área de experiencia o una colección de funcionalidad relacionada [McGregor 2004] [Clements and Northrop 2002].

Se utiliza el término **activo reutilizable (core asset)** para referirse a los artefactos o recursos que se utilizan en más de un producto de los generados por la *línea de productos*. Los *activos reutilizables* pueden ser requisitos, documentación, planes, componentes, arquitecturas, etc.

La creación de un miembro de una *línea* no se realiza de una forma ad hoc, sino que se realiza de una forma planificada a través de un **plan de producción**. Un *plan de producción* es una descripción de actividades necesarias para generar un producto de una *línea* [Clements and Northrop 2002]. Un *plan de producción* captura la estrategia para desarrollar productos a partir de los *activos reutilizables* [Chastek et al. 2002].

El primero de los objetivos de la presente tesis, y como ya ha sido expuesto en el capítulo primero, **es el establecimiento de una Metodología para la elaboración de líneas de productos aplicadas al desarrollo de funcionalidades de bases de datos.**

Pese a que la metodología expuesta se utiliza en un contexto determinado, una Base de Datos de Gestión de la Configuración (CMDB), puede extenderse su uso a otras bases de datos. No obstante, el alcance del contexto de aplicación de la metodología, es lo suficientemente ambicioso.

2.8.La Ingeniería de Dominio

Como ya se ha mencionado en el primer capítulo, existen dos enfoques distintos de ingeniería del software: la **ingeniería de dominio**, que se centra en el desarrollo de elementos reutilizables que formarán una familia de productos y la **ingeniería de aplicación**, que se orienta hacia la construcción o desarrollo de productos individuales, pertenecientes a la familia de productos, y que satisfacen un conjunto de requisitos y restricciones expresados por un usuario específico.

Podemos encontrar diferentes definiciones de ingeniería de dominio. Destacamos las siguiente: *"La ingeniería de dominio se puede definir como el proceso clave que se necesita para el diseño sistemático de una arquitectura y de un conjunto de elementos software reutilizables que pueden ser usados en la construcción de una familia de aplicaciones relacionadas o subsistemas".* [Griss, M. L. 1996]. El objetivo fundamental de la ingeniería de dominio es la optimización del proceso de desarrollo del software en un espectro de múltiples aplicaciones que representan un negocio común o problema de dominio [Simos, M. 1995].

Dentro del contexto de la ingeniería de dominio, cada una de las características que define un sistema es una ***feature***. Tanto los usuarios, como los analistas y los desarrolladores están involucrados en el desarrollo, pero con diferentes perspectivas. Los usuarios están más preocupados por los servicios o funciones que debe ofrecer el sistema; los analistas y diseñadores se centran en las tecnologías del dominio y los desarrolladores se interesan, especialmente, por las técnicas de implementación.

Existen diferentes **propuestas para llevar a cabo el proceso de análisis en la Ingeniería de Dominio**, como: **FODA** (Feature-Oriented Domain Analysis). [FODA 1990], **OODA** (Object Oriented Domain Analysis) [OODA 1995], **ODM** (Organization Domain Modeling) [Simos, M. 1995], **FORM** (Feature-Oriented Reuse Method) [FORM 1998] o **Feature RSEB** (Feature Reuse-Driven Software Engineering Business) [RSEB 1998], entre algunas de las más destacadas.

En la tesis se expondrá una propuesta específica para el proceso de análisis del dominio, mediante el uso de una notación específica para documentar la variabilidad de los productos. Esta notación permitirá obtener métricas esenciales para el cálculo de los costes de la línea de productos construida.

2.9.Programación Generativa y EDD

Czarnecki define la **programación generativa** de la siguiente manera: *"Generative Programming is a software engineering paradigm based on modeling software system families such that, given a particular requirements specification, a highly customized and optimized intermediate or end-product can be automatically manufactured on demand from elementary, reusable implementation components by means of configuration knowledge".* [Czarnecki 2000].

Como ya se ha mencionado en el primer capítulo, la **programación generativa** considera que el aumento de la productividad pasa por elevar el nivel de abstracción de los lenguajes de programación, haciendo el uso de especificaciones, que son representadas a través de modelos. Un aspecto clave de este paradigma es la traducción automática de los modelos a código ejecutable, y uno de los enfoques utilizados es aplicar las transformaciones sobre la especificación de los productos. La implementación de estas transformaciones es un proceso bastante complejo y suelen estar soportadas por lenguajes que carecen las prestaciones típicas de los lenguajes de programación.

Para superar este aspecto, en la Tesis se hará uso de un nuevo proceso de desarrollo de familias de productos, denominado **EDD (Exemplar Driven Development),** que aprovecha la similitud entre los productos de una familia para construirlos por analogía. EDD propone que la aplicación de transformaciones no se realice sobre la especificación de los productos, sino sobre alguno de los productos.

El traductor es un programa que toma un producto previamente desarrollado del dominio, al que se denomina ejemplar, y lo transforma para adecuarlo a la especificación del caso de estudio [Heradio et al. 2005] [Heradio 2007].

Este proceso, EDD, será utilizado en todo su alcance en esta tesis, y se realizará una propuesta de extensión, incluyendo nuevas fases y técnicas que nos permitirán adaptarlo a la construcción de bases de datos y CMDBs.

2.10. La Ingeniería de Requisitos

La **ingeniería de requisitos** es la disciplina que constituye la primera etapa en el ciclo de vida del proceso de desarrollo software. Las últimas tendencias en ingeniería de requisitos proponen trabajar con los *goals* (objetivos) [Raian et al 2010] [Yu et al. 2008] [Yu et al. 2008 2] [Kousik and Raman 2007] [Oliver et al. 2005].

Los *goals* se estructuran en *hard y soft*, entendidos como objetivos funcionales y no funcionales. Los objetivos no funcionales no se pueden demostrar o solo parcialmente, como son: el rendimiento, la facilidad de uso, etc. El **NFR** (Non Functional Requirements) **Framework** [NFR 1999] propone la utilización de una serie de modelos para representar los *soft goals*. Otras propuestas, incluyen otros elementos a ser considerados en la ingeniería de requisitos, como son:

- Las **Tareas**, como maneras de alcanzar los objetivos [Daniel et al. 2010] [Bertran et al. 2010].

- Los **Agentes**, como entidades que utilizan tareas para alcanzar objetivos [Adriana et al. 2000] [Wood et al. 2001] [Bresciani et al. 2004] [Henderson 2005] [A. García et al. 2006].

- Los **Recursos**, como los recursos necesarios para alcanzar las tareas.

En cuanto a la representación de requisitos con metodologías orientadas a objetos como Rational Unified Process (RUP) [Kruchten 2004], se propone la utilización de Modelos de **Casos de Uso** basados en Unified Modeling Language (UML) [Booch et al. 2005].

La ingeniería de requisitos, en lo que respecta al estudio de **familias de productos e ingeniería de dominio, propone el trabajo con modelos orientados a características** (features) como FODA, FORM o Feature RSEB. Estos modelos permiten representar la variabilidad [FODA 1990] [FORM 1998] [RSEB 1998].

Otras novedosas iniciativas proponen la utilización de **escenarios**, tareas y características para construir modelos de ingeniería de requisitos, donde el conjunto de objetivos se une para formar escenarios (mediante diagramas *and / or*) y cuyas variantes están enfocadas hacia factores de calidad (*soft goals*), y las tareas implementan las acciones de los escenarios con diferentes contribuciones a los *soft goals.* Al final, el escenario es un intermedio entre los objetivos y las tareas [R. Díaz 2002] [B. González et al. 2004]. Estas propuestas permiten la relación entre las tareas y los diagramas FODA, con lo que las características pueden ser modeladas como tareas. En estas propuestas se consideran tres vistas: la de alto nivel (*modelo de objetivos*), la de cualidades de sistema (*soft goals*) y la de bajo nivel (*tareas*).

Otras técnicas derivadas de la ingeniería de requisitos que se pueden utilizar para representar la variabilidad en familias de productos son los **casos de uso variables** [John and Muthig 2002] [Thomas and Horst 2002] [Eriksson 2006], **los requisitos textuales variables** [Pohl et al. 2005], los **modelos de características** [B. González et al. 2004] [López 2006] y los **lenguajes específicos del dominio textuales y visuales** [Pérez y Lara 2006].

Dentro de la metodología propuesta para el desarrollo de nuestros productos, que como ya se ha mencionado será una extensión del proceso EDD, **se definirá un proceso propio de análisis de los requisitos del dominio bajo estudio**. En este proceso se utilizarán los escenarios y los modelos orientados a características**. Se utilizarán nuevas notaciones para los modelos orientados a características** (Features) **y un lenguaje específico del dominio**, que serán analizados en profundidad en las próximas secciones y capítulos.

2.11. Los modelos de características y las Líneas de Productos SW

Nuestro objetivo es desarrollar los productos con la máxima calidad y eficacia. Las principales ventajas que esperamos obtener utilizando las líneas de productos software son la reducción del coste de desarrollo, la mejora de la calidad de los productos y la reducción en el tiempo de desarrollo.

Las SPL se pueden ver desde dos perspectivas: el **espacio del problema y el espacio de la solución**. El Espacio del problema define un contexto que reúne un conjunto de abstracciones específicas del dominio. Estas abstracciones se utilizan para especificar el miembro de la familia de productos que se desea construir.

El Espacio de la solución define un contexto que reúne las abstracciones de la implementación. Estas abstracciones se pueden utilizar para crear implementaciones de las especificaciones, guiadas por las abstracciones específicas del dominio que residen en el espacio del problema.

La distinción entre Espacio solución y Espacio problema es clave, porque nos permite disminuir el salto conceptual que tiene que efectuar el definidor del producto al expresar los conceptos del producto deseado. El definidor del producto se mueve en el espacio del problema, utilizando conceptos muy próximos al dominio, y sin detalles acerca de la implementación.

Al mismo tiempo, esta distinción permite la evolución paralela de ambos espacios. Por ejemplo, se puede mejorar la forma de especificar el producto (*espacio del problema*) y, por otro lado, se pueden mejorar las implementaciones de estos productos (*en el espacio de la solución*).

Las líneas de productos software, como ya mencionamos con anterioridad, se gestionan en términos de un conjunto de "**features**" (características). Algunas de las "*features*" serán comunes para todos los productos potenciales de la línea y otras "*features*" sólo se encontrarán en algunos productos. En la literatura encontramos diferentes definiciones de lo que significa una "*feature*":

- ✓ *"Una característica es cualquier cosa que los usuarios o programas clientes querrían controlar sobre un concepto"* [Czarnecki 2000].

- ✓ *"Una característica es un aspecto, cualidad o característica de un sistema software o sistemas software que para un usuario es visible y distintivo"* [Lee et all 2002].

- ✓ *"Una característica es un incremento en la funcionalidad"* [Zave 2004].

- ✓ *"Una característica es un elemento distinguible de un sistema que es relevante para un participante del sistema"* [Simos et al. 1996].

- ✓ *"Una característica describe un aspecto del sistema desde un punto de vista del usuario o cliente, que esencialmente consiste en un conjunto cohesivo de requisitos individuales"* [Chen et al. 2005].

Además de la gestión de features, también encontramos en la bibliografía el uso de **puntos de variación o variantes** (variation point / variant). De acuerdo con [Pohl et al. 2005] las definiciones de estos elementos son: "*a variation point is a representation of a variability subject within domain artifacts enriched by contextual information" y "variant is a representation of a variability object within domain artifacts."*

Las features (características) y los puntos de variación / variantes se utilizan tanto en los espacios del problema como de la solución. No hay un consenso en este sentido. Por ejemplo, en [Biglever 2009] [Gomaa 2004] las características se utilizan en el espacio del problema y los puntos de variación/variantes en el espacio de la solución. Sin embargo en [Metzger et al. 2007] es al contrario, las características se utilizan en el espacio de la solución y los puntos de variación/variantes en el espacio del problema. En [Czarnecki and Antkiewicz 2005] las características se utilizan en el espacio de la solución y del problema. En [Pohl et al. 2005] los puntos de variación/variantes se utilizan en el espacio de la solución y del problema.

En el presente trabajo **haremos uso de la gestión de características** y utilizaremos la siguiente definición de *feature*, en términos generales: *"**una feature es una propiedad de un concepto, dentro del dominio, que, tanto desde la perspectiva del espacio de la solución como del espacio del problema representa un aspecto relevante para cualquier stakeholder**".*

Para llevar a cabo una representación de las features se utilizan los **diagramas de características**, que visualmente simplifican la comprensión del sistema. No obstante, con el fin de permitir el trabajo computacional con estos modelos, es necesario formalizarlos [Batory et al. 2006] [Benavides et al. 2007].

Batory propone un mecanismo de formalización de los modelos de características en términos de gramáticas de atributos y fórmulas lógicas proposicionales [Batory 2005]. Con la formalización de estos modelos podemos, por ejemplo, comprobar que las diferentes configuraciones establecidas sean correctas.

Hay otras propuestas como la programación orientada a características (FOP, Feature Oriented Programming), que es un paradigma para líneas de productos software donde los programas se sintetizan componiendo características (features) [FOP 2007].

También tenemos Feature Oriented Model Driven Development (FOMDD), que es una combinación de FOP y del desarrollo dirigido por modelos (MDD, Model Driven Development) [Sami et al. 2005], donde los modelos se utilizan para especificar los programas, y sobre éstos se efectúan transformaciones para sintetizar los ejecutables [FOMDD 2007].

Con el fin de incrementar de forma progresiva la especialización de los modelos de características se propone la configuración en etapas (*staged configuration*). La configuración se divide en diferentes etapas que se llevan a cabo por diferentes roles (configuradores o desarrolladores) y en diferentes momentos. En cada etapa se lleva a cabo un proceso de especialización que consiste en seleccionar o deseleccionar un conjunto de características y añadir restricciones. [Czarnecki et al. 2005] [Czarnecki et al. 2005, 2].

Un modelo de características describe todas las posibles configuraciones de un sistema software, centrándose en aquellas características que puedan diferir para cada una de las configuraciones [Deursen and Klint 2001]. A través de un diagrama de características se puede representar, con una notación gráfica, las dependencias entre las características.

Los modelos de características se utilizan con dos propósitos principales.

1) El primero de ellos es representar las variabilidades (requisitos comunes y variables de los productos), organizar, clasificar y representar las dependencias entre las características.

2) Un segundo propósito de estos modelos es servir de base como árbol de decisión para configurar (seleccionar las características) y derivar la configuración correspondiente a un producto.

Un **diagrama de características** corresponde a una vista gráfica de un modelo de características. Las características del sistema suelen estar estructuradas de forma jerárquica. Las características padre se pueden descomponer en otras características (características hija), y éstas pueden pertenecer a las categorías:

- **y**: todas las subcaracterísticas tienen que ser seleccionadas.

- **Alternativa**: sólo una característica puede ser seleccionada.

- **O**: una o más características pueden ser seleccionadas. Esta relación puede llevar asociada una cardinalidad del tipo n: m, que indica que un mínimo de n características deben de seleccionarse, y como mucho se pueden seleccionar m.

- **Obligatoria**: la característica debe de estar en el producto.

- **Opcional**: características que pueden o no estar en el producto final.

Con las características opcionales y alternativas se representa parte de la variabilidad del sistema. Las características opcionales pueden ser seleccionadas o no, y las características alternativas permiten elegir una característica de un conjunto. Podemos encontrar variaciones en cuanto a la capacidad expresiva de los modelos de características, como analizaremos a continuación.

2.12. Concepto de Variabilidad de un Dominio

Una vez hemos presentado los modelos y diagramas de características y su impacto en el desarrollo de líneas de productos, ampliaremos el concepto de gestión de la variabilidad.

La variabilidad es la capacidad de un sistema, un activo reutilizable o un entorno de desarrollo, de producir un conjunto de artefactos que difieren entre ellos en la forma que previamente se ha planificado [Bachmann and Clements 2005].

También por variabilidad entendemos la capacidad o tendencia a cambiar de un artefacto, para ser extendido, personalizado o configurado, con el objetivo de ser usado en un contexto particular [Bosch 2005].

Uno de los puntos clave en el desarrollo de familias de productos es identificar cuáles son los aspectos comunes y variables de una misma familia de productos. Cada producto es un miembro de una familia, y esto significa que comparte elementos con todos los otros miembros de la familia, pero también se trata de un producto diferente y, por tanto, tiene elementos distintos.

Es necesario gestionar la variabilidad para permitir la creación de diferentes productos de una forma flexible. La especificación del nuevo producto tendrá sus propios mecanismos para expresar la variabilidad, pero el diseño, la arquitectura, la implementación, las pruebas y el mantenimiento también incluirán sus propios mecanismos.

Existen varias técnicas y métodos para representar la variabilidad que serán analizados a continuación.

2.13. Técnicas y Métodos para documentar la Variabilidad

Durante los últimos veinte años, los investigadores y la industria han desarrollado varias técnicas y métodos para documentar la variabilidad en los productos de la ingeniería del software.

Una primera propuesta se introdujo como parte del método **FODA** (Feature-Oriented Domain Analysis) en 1990 [FODA 1990]. A partir de esta propuesta inicial se han construido diferentes métodos como **FORM** (Feature-Oriented Reuse Method) [FORM 1998] o **Feature RSEB** (Feature Reuse-Driven Software Engineering Business) [RSEB 1998], además se han instrumentalizado otros enfoques como la **programación generativa** [Czarnecki 2000], **la ingeniería de líneas de productos software** [Pohl et al. 2005] o **PLUSS** [Eriksson et al. 2005].

Otras técnicas y metodologías se pueden basar en las investigaciones de algunos autores como M. Riebisch [Riebisch et al. 2002], J. van Gurp [Gurp et al. 2001], A. van Deursen [Deursen et al. 2001], K. Czarnecki [Czarnecki et al. 2005] [Czarnecki et al. 2005, 2] o H. Gomaa [Gomaa 2004]. Además, otras propuestas han incluido el **desarrollo de herramientas de soporte a la documentación de la variabilidad** que serán expuestas en la siguiente sección como Gears [Biglever 2009] y pure::variants [Variant 2010].

Las diferencias entre los diferentes enfoques incluyen, pero no se limitan a, los siguientes aspectos: sintaxis, semántica, expresividad, propósito y soporte de automatismo. En lo que se refiere a los **aspectos de sintaxis**, la mayoría de las propuestas son notaciones gráficas, pero también hay lenguajes textuales, como el *Lenguaje Declaraciones de Características* proporcionado por Gears [Biglever 2009].

En lo que respecta a los **aspectos de semántica**, las notaciones gráficas suelen representar diagramas de características por medio de nodos y conexiones. En estos diagramas, en algunas notaciones, la función opcional está vinculada a los nodos (por ejemplo, en el caso de PLUSS [Eriksson et al. 2005]), y en otras está vinculada a las conexiones (por ejemplo, en la notación propuesta por M. Riebisch et al.).

Sobre los **aspectos de expresividad**, mientras que la mayoría de los lenguajes desarrollados para la documentación de la variabilidad son expresivamente completos, es decir, capaces de representar cualquier serie de prestaciones de combinaciones del mundo real, algunos de ellos no lo son, como por ejemplo, la notación original FODA.

En lo que al **propósito** se refiere también encontramos diferencias; mientras que el *Modelo de Variabilidad Ortogonal* (OVM) [Pohl et al. 2005] está orientado al espacio del problema y de la solución, el *Lenguaje Declaraciones de Características* proporcionado por Gears está orientado exclusivamente al espacio de problema.

Esta profusión de lenguajes y notaciones dificulta la comunicación entre los desarrolladores, diseñadores y usuarios y, además, provoca problemas de portabilidad de los diagramas de características entre las herramientas de soporte. Por último, precisamente, está el aspecto de las herramientas de soporte.

La automatización y las herramientas es un aspecto clave para el soporte de estas técnicas, métodos y lenguajes para la gestión de la variabilidad y, como analizaremos en la próxima sección, algunas de estas técnicas y métodos tienen un soporte detrás con mayor o menor funcionalidad y otras ni siquiera lo tienen.

Finalmente, mencionaremos que **la propuesta de varios autores es utilizar Varied Feature Diagram+ (VFD+) como lenguaje de representación de la variabilidad** [Schobbens et al. 2006] [Metzger et al. 2007] [Schobbens et al. 2007] [Heymans et al. 2008]. VFD+ es un lenguaje de representación gráfica en forma de árbol, donde las características son los nodos del árbol.

Según estos autores, VFD+ es el lenguaje formal idóneo para la representación de la variabilidad. Este lenguaje es completo desde el punto de vista de la expresividad, tal y como se demuestra en [Schobbens et al. 2007].

VFD+ soporta la traducción entre sus representaciones, sin pérdida de información sobre la estructura original. Además, los principales métodos ya mencionados con anterioridad como FODA, FORM, FeatureRSEB, la programación generativa y PLUSS están embebidos en VFD+ y las diferentes propuestas de los autores M. et Riebisch et al., J. van Gurp et al. y A. van Deursen et al, pueden ser integradas en VFD+ [Schobbens et al. 2007]. Por ejemplo, en [Metzger et al. 2007] se demuestra como traducir desde OVM a VFD+.

2.14. Herramientas de soporte para la Variabilidad

Una vez analizados los diferentes métodos, técnicas y lenguajes para la gestión de la variabilidad, analizamos las posibles herramientas de soporte a los mismos. **Podemos distinguir hasta cuatro tipologías diferentes de herramientas**.

En la **primera tipología incluimos las herramientas que podemos utilizar para crear, gestionar y configurar modelos de *características*.** Estas herramientas, a su vez, las podemos dividir en dos grandes grupos:

- Herramientas que ofrecen las funciones de visualización y organización de las *características* y

- herramientas que ofrecen las funciones relacionadas con la validez y dependencias entre *características*.

De esta primera tipología destacamos las siguientes herramientas:

- FeatureModeling [Feature Modeling 2010], herramienta complementaria a la plataforma ECLIPSE,
- Pure::Variants [Variant 2010],
- CaptainFeature [Captain Feature 2010],
- Xfeature [XFeature 2010] y
- VARMod-PRIME Tool [Varmod-Prime 2010].

En una **segunda tipología incluimos herramientas que nos permiten crear lenguajes específicos del dominio textuales**. En este grupo tenemos como ejemplo las siguientes herramientas:

- Xtext *del Proyecto Open Architectureware* [OARW 2010],
- IBM SAFARI [SAFARI 2010] y
- ANTLR, *ANother Tool for Language Recognition* [ANTLR 2010].

En **una tercera tipología incluimos las herramientas que nos permiten crear lenguajes específicos del dominio visuales**. Como ejemplo de este tipo de herramientas tenemos:

- Metaedit++ Modeler [METAEDITM 2010],
- GEMS (Generic Eclipse Modeling System) [GEMS 2010],
- GMF (Graphical Modeling Framework) [GMF 2010] y
- DSL (Domain Specific Language) Tools [DSLT 2010].

En la última tipología incluimos las herramientas CASE convencionales. Podemos usar los diagramas de UML y los mecanismos de extensión de UML [UML 2003] [Booch et al. 1998] (por ejemplo, los perfiles) para crear los modelos que expresan la variabilidad. Consideramos que es necesario que la salida de estas herramientas sea un estándar, como en el caso de XMI [MOF 2005] para permitir la introducción de estos modelos en otras herramientas.

Para finalizar nuestro análisis incluiremos un **resumen de las capacidades o funcionalidades que son deseables en este tipo de herramientas** para la automatización de ciertas funciones en los diagramas de características, siguiendo la recomendación de algunos autores [Benavides 2007] [Benavides et al. 2006]:

I. **La validación del Modelo.** Un Modelo de características es válido si, al menos, un producto está representado por el modelo. La mayoría de las herramientas y propuestas dan soporte a esta operación.

II. **Detección de características muertas.** Un Modelo de características válido puede tener contradicciones internas que conllevan a la aparición de características muertas. Una característica muerta es una característica que no aparece nunca en ningún tipo de configuración real de un modelo de características. La mayoría de las herramientas y propuestas dan soporte a esta operación.

III. **Soporte Interactivo.** Un ingeniero de aplicación podrá especificar determinados productos de una SPL mediante la configuración de un diagrama de características. Cuando el número de características es grande y

las dependencias entre ellas son complejas, manipular estos diagramas no es una tarea trivial. La asistencia interactiva durante el proceso de configuración, como por ejemplo proporcionar sugerencias o ayudar a la toma de decisiones durante todo el proceso, supone una ayuda importantísima en este proceso. Varios autores [Batory 2005] [Janota 2008] [Sun et al. 2008] [Wang et al. 2005] [White et al. 2008] y herramientas [Feature Modeling 2010] [Biglever 2009] [Variant 2010] proporcionan implementaciones para esta funcionalidad.

IV. **Optimización**. Esta funcionalidad tiene como entrada un modelo de características y como salida el mejor producto, de acuerdo con el criterio establecido por la función. Esta funcionalidad puede ser útil para, por ejemplo, proporcionar el producto que satisface un conjunto de requisitos o determinar el conjunto de productos más baratos si algunos atributos de coste fueran añadidos a los diagramas de características. Ejemplos de este tipo de funcionalidades los tenemos en [White et al. 2008, 2] y [Berre and Parrain 2008].

V. **Cálculo del número de productos.** Esta funcionalidad obtiene el número de productos de nuestro dominio. Aplicando modelos económicos de costes como el *Structured Intuitive Model for Product Line Economics* (SIMPLE) [Clements et al. 2005] es posible calcular el coste del desarrollo de la línea de productos. El número de productos es un dato clave en todos los modelos de costes para líneas de productos. Otra información clave de esta métrica es la satisfacibilidad, ya que un modelo de características es satisfacible si el número de productos es mayor de cero. Por ejemplo, la herramienta FAMA Tool Suite [Benavides 2010] obtiene esta medida con una limitación importante de escalabilidad (300 características). Las herramientas Gears, pure::variants y FMP [Biglever 2009] [Variant 2010] [Feature Modeling 2010] también obtienen esta medida, pero no admiten *constraints* en su funcionalidad y la escalabilidad es muy baja.

VI. **Porcentaje de uso de una característica.** Desde la perspectiva del espacio de la solución, las características se pueden ver como componentes de desarrollo. Dado que los esfuerzos tanto en el desarrollo como en el mantenimiento de los sistemas deberían centrarse, principalmente, en aquellos componentes que aparecen con más frecuencia en los productos, será conveniente conocer el número de productos que hacen uso de una determinada característica en los diagramas del espacio de la solución. No conocemos herramientas ni propuestas que den soporte a esta funcionalidad.

VII. **Escalabilidad**. La escalabilidad de las herramientas de soporte a la variabilidad es un aspecto clave. Hasta la fecha, aunque existen numerosas propuestas, la mayor parte de ellas sufren problemas de escalabilidad ya que solo permiten trabajar con pequeños diagramas. Por ejemplo, en [Hemakumar 2008] los experimentos trabajan con modelos de 64 características como máximo, en [Janota 2008] con un tope de 10 y en [Segura 2008] [Benavides 2007] el máximo es de 300 características. Sin embargo, en líneas de productos del mundo real las empresas han informado que los modelos ofrecen miles de características, como en [Steger et al. 2004] con un ejemplo de un modelo con 5.200 características. Además, si no solo estamos interesados en manipular modelos aislados sino modelos interrelacionados, vamos a trabajar en el mundo real con diagramas mucho más grandes y este aspecto será de vital importancia. Con el fin de aumentar la escalabilidad disponible algunos autores como [Segura 2008] proponen fraccionar los modelos y otros autores como [Broek et al. 2008] presentan soluciones que permiten eliminar las restricciones o constraints de los modelos transformando éstos en otros modelos más complejos sin las restricciones, aunque eso no disminuye el problema global de la escalabilidad.

VIII. **Control de errores.** Esta funcionalidad nos permite obtener información de un diagrama de características en tiempo de diseño, proporcionado datos sobre la satisfacibilidad del diagrama, las características muertas detectadas,

los errores de configuración, etc. Algunos autores como [Batory 2005] [Janota 2008] [Sun et al. 2008] [Wang et al. 2005] [White et al. 2008] y herramientas como [Feature Modeling 2010] [Biglever 2009] [Variant 2010] proporcionan explicaciones sobre los errores encontrados durante la configuración y el diseño de los diagramas. Otros autores como [Sun et al. 2008] proponen formalizar los diagramas utilizando el lenguaje Z [Davies and Woodcock 1996] y el uso de Alloy Analyzer [Alloy 2010] para el soporte de errores y otros como [Osman et al. 2008] utilizan el control de errores mediante reglas de conocimiento implementadas en Prolog [Colmerauer and Roussel 1992].

IX. **Refactorización**. Durante el desarrollo y el mantenimiento de una SPL su alcance suele variar. Para gestionar este tipo de modificaciones las herramientas deben incluir una serie de funcionalidades como determinar si dos modelos son equivalentes, es decir si ambos representan el mismo conjunto de productos (la herramienta FAMA [Benavides 2007] dispone de esta funcionalidad y en [Sun et al. 2008] se propone una implementación en Alloy para soportarlo), si un modelo está incluido en otros (el modelo B contiene el A significaría que tiene, además de los productos de B, otros productos) y permitir operaciones entre modelos como la unión o la intersección (en [Schobbens et al. 2007] se identifican diversas operaciones de este tipo y en [Segura 2008] se propone realizar la combinación de estos modelos mediante transformaciones gráficas).

X. **Control sobre los productos generados.** Una herramienta de soporte a la gestión de la variabilidad debe garantizar que cualquier diagrama generado es válido y que los productos que se generan de forma automática a partir de su especificación son productos reales. Diversos autores trabajan en esta área de investigación [Clements et al. 2002] [Czarnecki et al. 2006], proponiendo la descomposición de los modelos, pero presentan diversas limitaciones como el control de los bucles en los diagramas y la restricción de la existencia de los módulos descompuestos en diferentes modelos.

XI. **Generación automática de modelos.** Ante la ausencia de modelos reales de características con la suficiente escalabilidad [Heymans et al. 2008] algunos autores proponen esta funcionalidad como el caso de [Segura 2008], que ofrece un algoritmo para la generación automática de diagramas de características.

XII. **Portabilidad e interoperabilidad**. La portabilidad e interoperabilidad de las herramientas de soporte es otro factor clave. Trabajar con estándares abiertos y permitir interoperabilidad entre las herramientas y los entornos de desarrollo, para el espacio de la solución, y de análisis y diseño, para el espacio del problema, puede ahorrar costes y automatizar multitud de fases del desarrollo y el mantenimiento de los sistemas. En este caso destacamos las herramientas FM, GEMS y GMF [Feature Modeling 2010] [GEMS 2010] [GMF 2010], compatibles con el entorno de desarrollo ECLIPSE, un entorno ampliamente difundido o DSL Tools [DSLT 2010] compatible con el entorno .NET de Microsoft. No obstante, la mayor parte de estas herramientas presentan formatos propietarios para la representación de los diagramas, como el caso de las DSL Tools, que dificultan sobremanera su portabilidad e interoperabilidad.

XIII. **Facilidad de uso.** Por último, mencionaremos la amigabilidad de las herramientas, su facilidad de uso. Los roles que participan en el espacio del problema pueden estar muy alejados del campo de la ingeniería del software y la facilidad de uso se convierte un aspecto clave en este tipo de herramientas. La mayor parte de las herramientas analizadas presentan una dificultad en la configuración y uso bastante considerables.

Como complemento al proceso de desarrollo planteado se realizará una propuesta de herramientas y métodos que complementarán las iniciativas existentes y cubrirán gran parte de las funcionalidades analizadas.

3. La Metodología EODAM

"Unas buenas especificaciones incrementarán
la productividad del programador
mucho más de lo que puede hacerlo
cualquier herramienta o técnica"
-- Milt Bryce

En este capítulo se expone Metodología original EODAM (Exemplar Oriented DataBase Methodology), una metodología para la construcción de bases de datos orientada a ejemplares. La Metodología expuesta es un proceso extensible a cualquier desarrollo de base de datos y sus servicios asociados.

El presente capítulo contiene dos grandes secciones. La primera de ellas ofrece una visión general de la metodología EODAM y posteriormente detalla las fases principales que incluye la metodología. La segunda sección presenta una adaptación de la metodología EODAM al desarrollo de una base de datos de gestión de la configuración (CMDB).

3.1.EODAM. Visión General.

La Metodología propuesta se ha denominado, con sus siglas en inglés, EODAM (Exemplar Oriented Database Methodology); se trata de una metodología para bases de datos orientada a ejemplares. La Metodología expuesta es un proceso extensible a cualquier desarrollo de base de datos y sus servicios asociados. **EODAM es una extensión de EDD (Exemplar Driven Development)** [Heradio et al. 2005] [Heradio 2007], un nuevo proceso de desarrollo de familias de productos, que plantea la idea original de aprovechar la similitud entre los productos de bases de datos para construirlos por analogía.

La figura 3.1 resume el **proceso EDD**, que se descompone en **tres actividades básicas.**

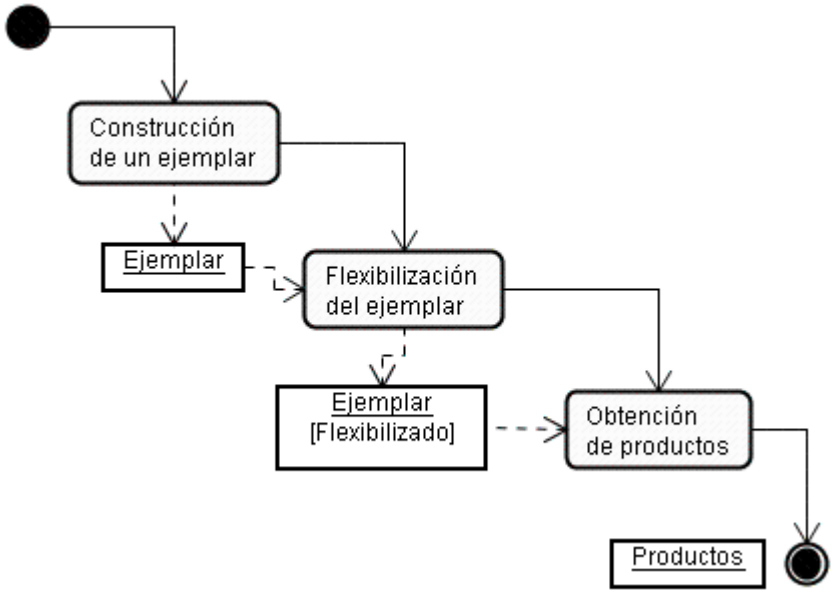


Figura 3-1 El proceso EDD.

En primer lugar, la **construcción de un producto concreto de la familia**, que se denomina **ejemplar**, en segundo lugar, la **realización de una infraestructura con ocultación de caja negra**, que facilita la obtención posterior de los productos (esta infraestructura se desarrolla flexibilizando el ejemplar, es decir, definiendo la

relación de analogía que permitirá derivar automáticamente del ejemplar todos los productos de la familia) y, en tercer lugar, la **obtención de cada producto parametrizando la infraestructura.**

La figura siguiente da una **visión panorámica de EODAM**, cuya primera actividad es construir la Base de Datos, posteriormente se desarrolla un ejemplar, esto es, el dominio específico a construir para la base de datos, que será un producto de la familia. A continuación, se busca cómo flexibilizar este ejemplar para que satisfaga los requisitos del resto de los productos.

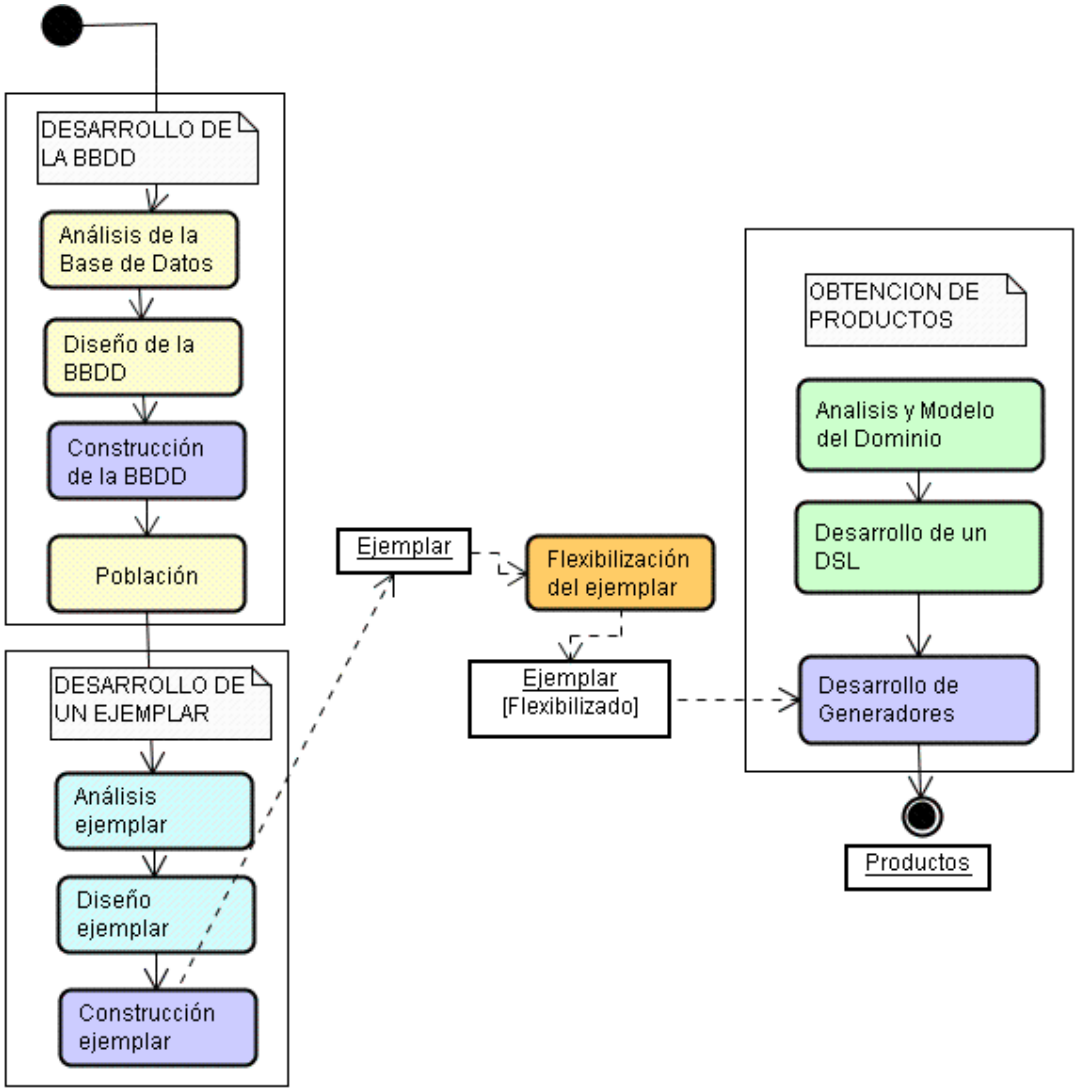


Figura 3-2 Visión panorámica de EODAM.

NOTA: La modelización de todos los procesos ha sido realizada con la herramienta JUDE (Java and UML Developers' Environment), una herramienta de Modelización. [JUDE 2007].

Es decir, se trata de definir una relación de analogía que permita derivar los demás productos del ejemplar. Por último, se obtienen todos los productos aprovechando la flexibilización del ejemplar.

A continuación, se desarrollan todas las actividades identificadas en la metodología.

3.1.1. Desarrollo de la BBDD

La primera actividad propuesta por EODAM es el desarrollo de la Base de Datos. Para ello, se proponen cuatro fases: comenzar por el análisis de la base de datos y posteriormente diseñar los modelos conceptual, lógico y físico de la misma y, por último, construir y poblar con los datos iniciales la base de datos. Posteriormente, analizaremos cada una de estas actividades para el caso concreto de una CMDB.

Como ya mencionamos con anterioridad, en principio, EODAM es una metodología extensible a cualquier base de datos y de ahí la no especificidad de estas fases iniciales. No obstante, se realizará un análisis muy detallado en el presente capítulo de la aplicación de esta metodología a la construcción de una base de datos de gestión de la configuración (CMDB).

3.1.2. Desarrollo de un ejemplar del dominio

Al comienzo del desarrollo de una familia de productos suele disponerse de algún ejemplar, pues la decisión de elaborar una familia, a menudo, se toma al detectar trabajo repetitivo en el desarrollo aislado de varios productos de un dominio o al identificar oportunidades de negocio en la ampliación de las prestaciones de un producto de éxito [Morisio et al. 2002].

Precisamente, una de las ventajas que tiene EODAM frente a otros procesos de desarrollo es que reconoce esta situación y trata de aprovecharla mediante la reutilización íntegra de un ejemplar. En los casos excepcionales en los que no se cuente con ningún ejemplar se desarrollará uno, aplicando el proceso de ingeniería del software *"para productos aislados"* que se estime más conveniente, que incluirá las etapas de análisis, diseño y desarrollo del ejemplar, tal y como se detalla en la visión panorámica.

3.1.3. Flexibilización del Ejemplar

La flexibilización del ejemplar, según el proceso EDD y tal como indica la figura 3-3, se descompone en varias fases. En primer lugar, el análisis de los requisitos de la familia de productos, en segundo lugar, la definición de una interfaz que facilite la especificación abstracta de los productos de la familia y, en tercer lugar, la implementación de la flexibilización, que a partir de las especificaciones abstractas obtenga los productos finales correspondientes.

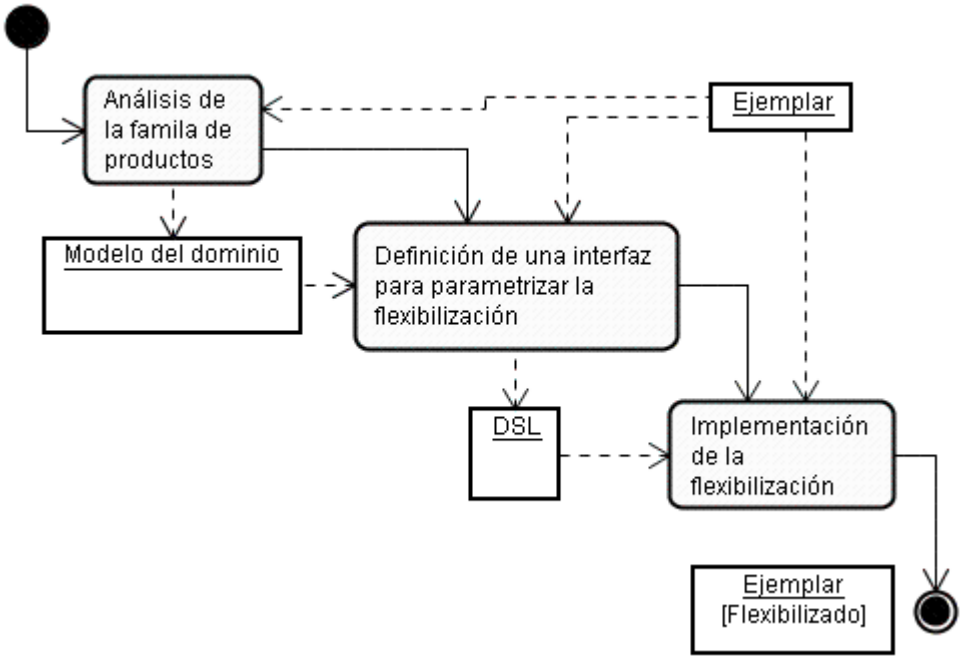


Figura 3-3 Flexibilización según EDD.

La flexibilización del ejemplar, según la metodología EODAM, es un proceso de mayor complejidad, aplicado al desarrollo de servicios para bases de datos, e incluye una serie de actividades que son expuestas en detalle en la figura siguiente.

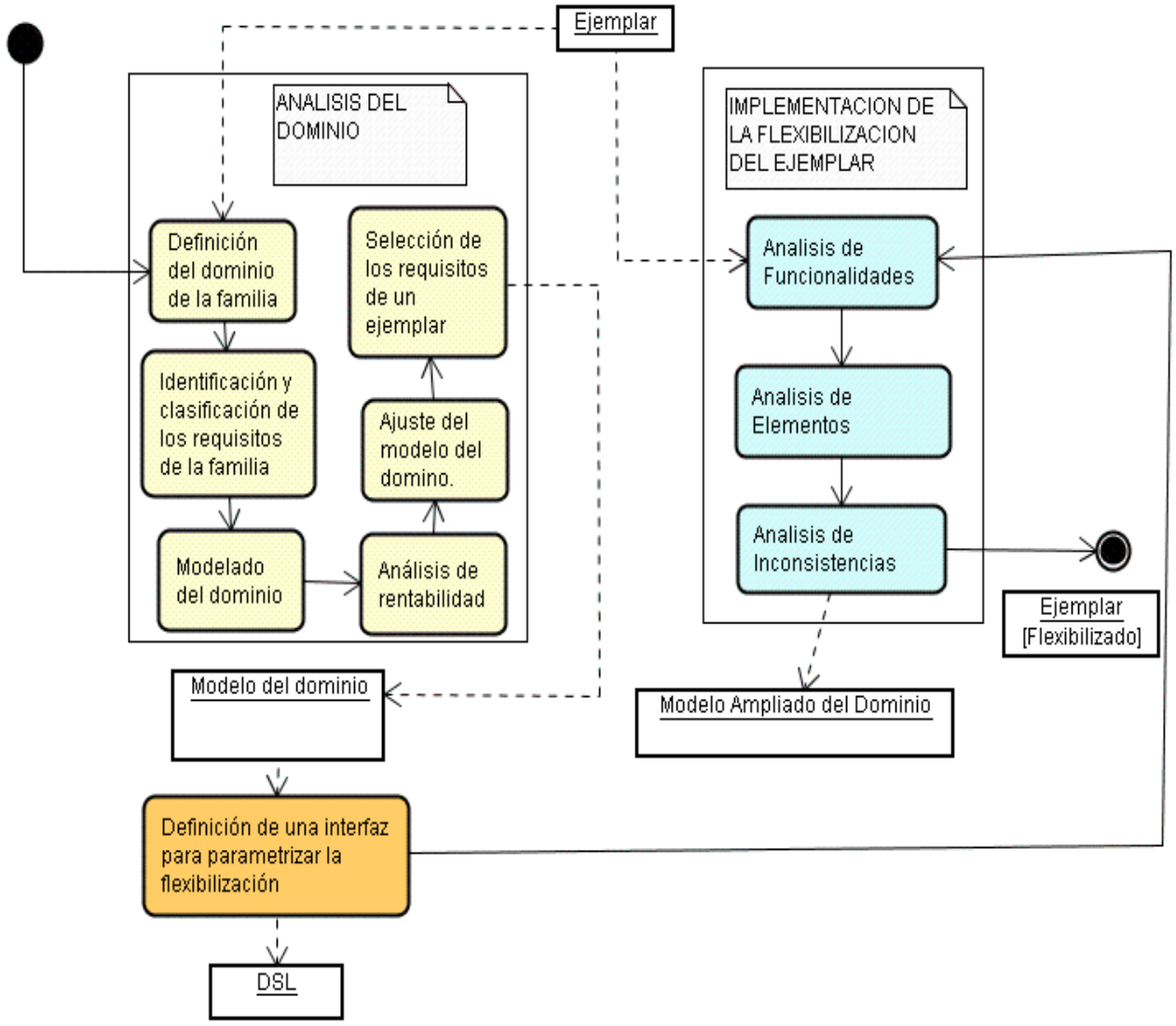


Figura 3-4 Flexibilización según EODAM.

Las tres fases principales de la flexibilización del ejemplar, según EODAM, son el análisis del dominio, la definición de una interfaz para parametrizar la flexibilización y la implementación de la flexibilización del ejemplar. A continuación, se detalla cada una de ellas.

3.1.3.1.Análisis del Dominio

En lo que se refiere a la primera fase de flexibilización del ejemplar, el análisis de la familia de productos, los objetivos del análisis de una familia de productos software son definir y acotar la familia, recabar toda la información relevante del dominio e integrarla de forma coherente en un modelo.

El análisis de una familia de productos se descompone en: 1) **la definición del dominio de la familia**, 2) la **identificación y clasificación de los requisitos** de la familia, 3) el **modelado del dominio**, 4) el **análisis de la rentabilidad** de la flexibilización del ejemplar, 5) el **ajuste del modelo** del domino y 6) la **selección de los requisitos del ejemplar.**

Aprovechando la **especificación de requisitos del ejemplar**, se procederá a:

- Identificar a los clientes potenciales de la familia de productos, sus necesidades y expectativas.
- Describir, de forma aproximada, el tipo de productos que se desea obtener.
- Describir el contexto donde se emplearán los productos. Esta actividad es especialmente importante en el caso de que se quieran producir componentes de otros sistemas y puedan darse problemas de incompatibilidad.
- Recoger el vocabulario propio del dominio en un glosario de términos.

Con respecto a la **identificación y clasificación de los requisitos** de la familia de productos, utilizando la definición del dominio, se identificarán los requisitos de los productos de la familia, anotando su nivel de importancia según la demanda de los clientes potenciales.

A continuación, los requisitos se clasificarán en fijos y variables. Un requisito es fijo si es común a todos los productos de la familia y tiene un valor único. Un requisito es variable si no es fijo.

Si se dispone de más de un ejemplar, la búsqueda de los requisitos fijos comenzará examinando la intersección entre los requisitos de los ejemplares.

Interesa encontrar un equilibrio entre la cantidad de requisitos fijos y variables. Los requisitos fijos son clave para mejorar la economía de alcance y la abstracción de la interfaz con que se especificarán los productos, que se limitará al ajuste de los requisitos variables. Por otro lado, el aumento de los requisitos variables amplía el dominio y, por tanto, los clientes potenciales de la familia.

Conviene determinar una serie de propiedades de cada requisito variable como su rango de variación o si tiene dependencia con otros requisitos. Por ejemplo, si su valor se deriva del valor de otros requisitos, si ciertas combinaciones de valores están prohibidas, el momento en que se concretará su valor (*binding time*) o quién especifica el valor de cada requisito (*binding role*). [Cleaveland 2001].

Posteriormente, se modela el dominio. Toda la información anterior se recopilará en un **modelo del dominio**. Siguiendo las recomendaciones de [Czarnecki et al. 2000] y [Greenfield and Short 2004], sugerimos utilizar un modelado de características. Como se analizará posteriormente, **en el capítulo 5, se realizará una propuesta novedosa para la construcción de este modelo**, que permitirá mejorar las propuestas actuales. **Este es otro de los objetivos de la presente tesis.**

El siguiente paso es el **análisis de la rentabilidad de la flexibilización del ejemplar**. Para decidir si vale la pena flexibilizar el ejemplar o, por el contrario, es más conveniente construir cada producto por separado, se estimarán los costes de desarrollar y mantener, por un lado, cada producto por separado y, por otro, la flexibilización del ejemplar y los productos obtenidos por ésta.

Como se podrá **observar en el capítulo dedicado al estudio económico y al análisis de la rentabilidad, se ofrece un modelo de económico para la familia de productos construida**. **Obtener este modelo es otro de los objetivos de la tesis.**

El modelo presentado podrá ser utilizado en otro ámbito de desarrollo software para líneas de productos.

Teniendo en cuenta el análisis de rentabilidad anterior, se reajustará el dominio para suprimir requisitos excesivamente costosos y pasar a considerar como fijos algunos requisitos variables.

La fase de análisis del dominio, como primera fase de la flexibilización de un ejemplar, concluye con la **selección de los requisitos de un ejemplar.** Como se indicó anteriormente, cuando no exista ningún ejemplar será necesario desarrollar uno. Convendrá que el nuevo ejemplar, para facilitar su posterior flexibilización, satisfaga un conjunto representativo de los requisitos de la familia de productos.

3.1.3.2.Interfaz para parametrizar la flexibilización

En lo que se refiere a la segunda fase de flexibilización del ejemplar**, se construye un interfaz con el objeto de parametrizar la flexibilización**. Para facilitar la última actividad dentro de la flexibilización del ejemplar, la obtención de productos parametrizando la flexibilización del ejemplar, será conveniente que dicha flexibilización ofrezca una interfaz que oculte los detalles de implementación.

Hemos seguido, para EODAM, la terminología utilizada en el proceso original EDD, no obstante, esta fase puede provocar cierta confusión por el uso de la palabra interfaz. Lo que realmente queremos en esta fase es una forma de especificar los productos a nivel de ingeniería de la aplicación, una meta-especificación o forma de expresar la variabilidad de los productos.

No se trata de la especificación de un producto concreto, sino de cómo se especifican los productos que se incorporan a la línea de productos. En este apartado podemos ofrecer un abanico amplio de posibilidades de especificar los productos: lenguajes específicos del dominio textuales o gráficos, asistentes, modelos de características, etc.

EODAM y EDD proponen modelar esta interfaz como un Lenguaje Específico del Dominio (DSL). La especificación de un DSL, como la de cualquier lenguaje formal, comprende la definición de su sintaxis y de su semántica.

En lo que respecta a la sintaxis, generalmente, la sintaxis de un DSL se definirá con una gramática independiente del contexto [Aho et al. 1990] o un metamodelo [MOF 2006]. EODAM propone un proceso para la obtención sistemática de la sintaxis abstracta de un DSL, especificada con una notación específica, como se detallará ampliamente en el capítulo sobre documentación de la variabilidad. Para definir la semántica del DSL algunos autores recomiendan dos alternativas [Ghosh et al. 2006]:

- Expresar una traducción del DSL a otro lenguaje cuya semántica esté bien definida.
- Expresar la semántica del DSL en función del hilo de ejecución de las sentencias del DSL.

Para el caso de la metodología EODAM, como los productos de nuestro dominio y la flexibilización del ejemplar son formales, la semántica del DSL vendrá implícita en dicha flexibilización, que describirá cómo obtener un producto final a partir de su especificación DSL (opción primera).

3.1.3.3.Implementación de la flexibilización del ejemplar

La última fase de la flexibilización del ejemplar es la implementación. Para elaborar correctamente la flexibilización de un ejemplar, deberá disponerse de algún mecanismo formal que cumpla las siguientes propiedades: que sea modular, no invasivo, aplicable a la flexibilización de cualquier producto software, capaz de producir productos eficientes y con algún medio para detectar automáticamente errores en la flexibilización.

El código es el producto del ciclo de vida con mayor tasa de reutilización. Por esta razón, las técnicas que comúnmente se emplean para generalizar código y posibilitar su reutilización parecen buenas candidatas para la flexibilización de un ejemplar. Dentro de la metodología EODAM, en esta fase, se distinguen varias actividades: Análisis de Funcionalidades, Análisis de Elementos y Análisis de Inconsistencias.

La primera de estas actividades consiste en el **análisis de las funcionalidades** que es necesario flexibilizar en el ejemplar para la obtención de los productos. Este análisis, principalmente, consiste en:

- Obtener aquellas funcionalidades que son opcionales y obligatorias. Existirán algunas funcionalidades que siempre son requeridas y otras que serán opcionales.

- Determinar las funcionalidades que deben ser flexibilizadas y cuales no. Algunas de las funcionalidades serán comunes a todos los productos del dominio y otras deberán ser desarrolladas a medida y, por tanto, deberán ser flexibilizadas.

Para comprender la segunda de las fases, el **Análisis de Elementos**, es necesario poner de manifiesto que el objetivo de la metodología EODAM es el desarrollo de líneas de productos software para dar soporte a un conjutno de servicios asociados a bases de datos. En la mayor parte de los casos, existe una dependencia entre el desarrollo de los productos de la línea y la Base de Datos objetivo.

En este sentido, **se considerarán Elementos** *a aquellas entidades que, teniendo una dependencia interna con la Base de Datos (como por ejemplo son: las tablas de la base de datos, las columnas, las claves, los usuarios de la base de datos, etc.), tienen algún tipo de incidencia en el desarrollo de los productos del dominio*. Es necesario realizar un análisis de los elementos del dominio objetivo. Esta fase incluye esta actividad.

Una vez determinados los requisitos de nuestro dominio, en la fase de análisis del dominio, y las funcionalidades y los elementos en esta última fase, es necesario el desarrollo de una actividad que permita la **detección de todas las inconsistencias**. Entre todos estos objetos, y entre los propios requisitos del dominio, podrá haber dependencias y combinaciones no permitidas. En esta actividad se propone la realización de un análisis de la implicación de esas inconsistencias a la hora de flexibilizar los ejemplares del dominio.

Puede ser el caso que este análisis nos muestre que no es necesario realizar ninguna flexibilización de los ejemplares específica para tratar esas incompatibilidades, y que únicamente sea necesario comprobar dichas inconsistencias en nuestro DSL, de tal forma que si son detectadas no se realicen los desarrollos correspondientes para obtener los productos. Hay que insistir en que continuar con el proceso de obtención de los productos sin considerar estas inconsistencias podría dar lugar al desarrollo de productos erróneos. De ahí la importancia de esta fase.

3.1.4. Obtención de los Productos

La última fase de la metodología EODAM es la obtención de productos, a partir de la flexibilización del ejemplar. Aprovechando la interfaz abstracta de la flexibilización del ejemplar, se especificarán las particularidades de cada producto. Para ello, ya se ha realizado un análisis y un modelado del dominio en las anteriores fases**.**

La fase de obtención de los productos incluye como primera actividad este análisis y modelado que, aunque en parte ya se ha realizado, **es necesario integrar en un solo modelo**, que incluirá el modelo obtenido durante la primera fase de flexibilización del ejemplar, y que contiene los requisitos, el modelo de las funcionalidades y el de los elementos, obtenidos ambos en la última fase de la flexibilización del ejemplar. De esta manera, contamos con un único modelo que permite especificar todo nuestro dominio. **Este modelo se especifica mendiante un Lenguaje Específico del Dominio (DSL)**, de la misma forma que ocurría durante la fase de flexibilización del ejemplar, pero únicamente con los requisitos.

La última actividad es el desarrollo de los generadores que, a partir del ejemplar y del DSL, permiten obtener el resto de los productos. Para la implementación de estos transformadores se utiliza el lenguaje EFL (Exemplar Flexibilitaton Lenguage) tal y como se propone en [Coz et al. 2008] [Heradio 2007]. La implementación actual de EFL corresponde a un lenguaje específico del dominio embebido en Ruby [Thomas et al. 2004]. La implementación de este lenguaje se distribuye a través de una licencia de tipo LGLP (GNU General Public License) [LGPL 2007] y está disponible en diferentes repositorios como Ruby Forge o RAA (Ruby Application Archive) [EFL 2007]. El uso de EFL y el desarrollo de los generadores serán abordados en detalle en el capítulo octavo, correspondiente a la implementación del servicio de notificación de cambios para una CMDB.

3.2.Adaptación del Proceso a una CMDB

Como ya hemos mencionado, EODAM es una metodología general para la implementación de servicios de soporte a bases de datos, mediante el uso de un desarrollo orientado a líneas de productos software.

EODAM puede ser utilizada en cualquier ámbito de bases de datos. En nuestro caso, haremos uso de EODAM en el contexto de una Base de Datos de Gestión de la Configuración (CMDB) y, por tanto, será necesario realizar una adaptación a la metodología para tratar esta tipología muy específica de base de datos. Como ya vimos, las fases que inciden directamente sobre el desarrollo de la Base de Datos, dentro de EODAM son las que se exponen a continuación.

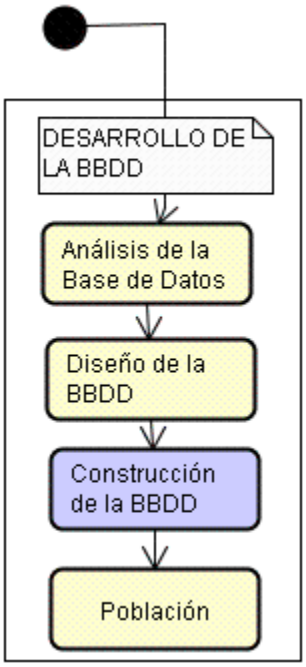


Figura 3-5 Desarrollo de la BBDD en EODAM.

Para el caso de una CMDB, nuestra propuesta de adaptación se describe en la figura siguiente, donde se detallan una serie de actividades adicionales a las fases ya identificadas en EODAM.

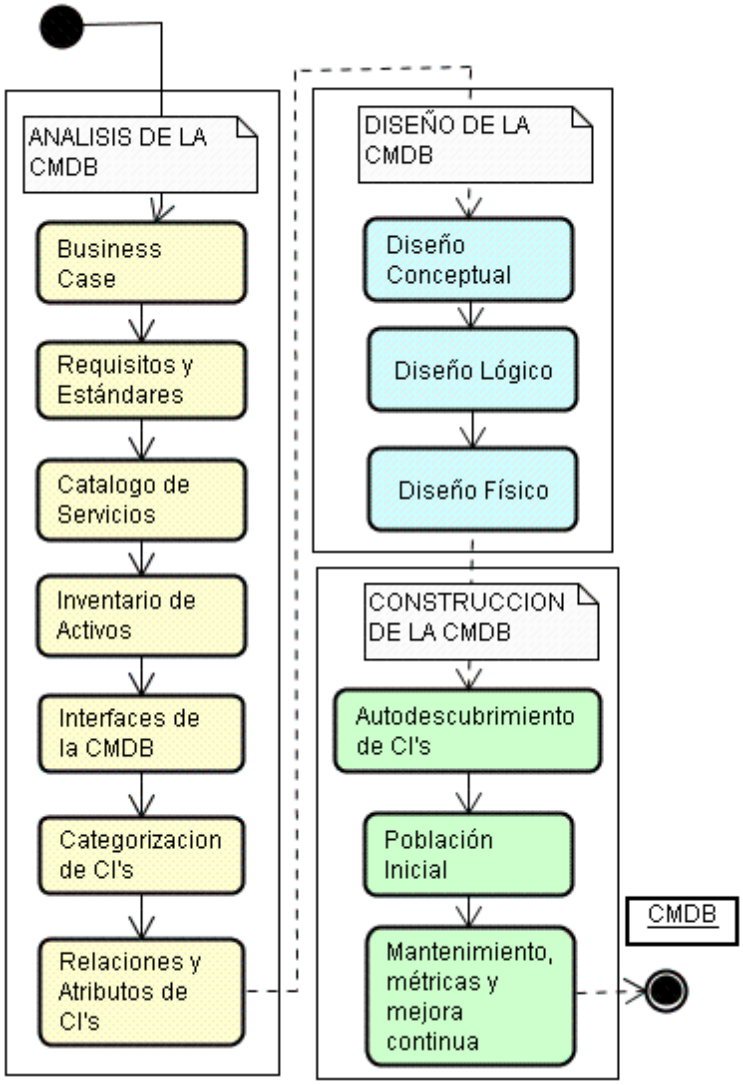


Figura 3-6 Desarrollo de la CMDB en EODAM.

3.2.1. Análisis de la CMDB

1. La **primera fase** propuesta para la construcción de la CMDB es la **elaboración de un Bussines Case** de la CMDB. Esta actividad se detallará según establecen las buenas prácticas PRINCE2 [PRINCE2 BC 2010]. Este *Bussines Case* incluirá la siguiente información: las razones para el desarrollo de la CMDB y los servicios de soporte, las opciones para el desarrollo, los beneficios esperados, los riesgos, una planificación del trabajo y el proceso de evaluación del desarrollo.

2. La **segunda de las actividades** es la **identificación de los requisitos de la CMDB**, que incluirá los requisitos legales, de seguridad y de negocio y los estándares a ser aplicados en su desarrollo. En nuestro caso, el uso de las buenas prácticas recomendadas por ITIL

3. La **tercera actividad** es la **identificación del Catálogo de Servicios TIC** de la organización y, en concreto, **de aquellos servicios que tienen un impacto en el desarrollo de la CMDB**. En el capítulo correspondiente al análisis del dominio haremos un análisis de los servicios que impactan en el desarrollo de una CMDB. **Uno de los objetivos de la tesis es analizar en detalle este dominio y realizar esta tarea de evaluación de impacto de la CMDB en los servicios TIC de una organización.**

4. A continuación, es necesario realizar un **inventario de los activos TIC** de la organización **que van a estar soportados por la CMDB**. Es la cuarta actividad.

5. Como consecuencia de la tercera actividad, será necesario **analizar todos los interfaces con los que va a interactuar la CMDB**. En el capítulo correspondiente a la construcción de la CMDB haremos un análisis de los sistemas de información que interactúan con el servicio de notificación de cambios de la CMDB. Es la **quinta actividad de este proceso.**

6. Por último, se **analizarán los ítems de configuración, CI's**, a los que dará soporte la CMDB, determinando su alcance, su identificación, dotando de una categorización a los mismos y obteniendo las relaciones y los atributos de los mismos. Toda esta información será diseñada en la segunda etapa del proceso.

3.2.2. Diseño de la CMDB

Con toda la información obtenida en la fase anterior, se construirá el modelo conceptual de la CMDB y posteriormente los modelos lógico y físico.

El Modelo Conceptual es utilizado en las primeras fases del ciclo de vida, en él se identifican las entidades y atributos principales, claves candidatas, dominios y relaciones (que pueden ser múltiples, recursivas, generalizaciones, agregaciones, etc.). El principal objetivo de este tipo de modelos es establecer el ámbito de información que gestionará el sistema. Suele ir acompañado de la descripción de las *"reglas del negocio".* No se suelen tener en consideración las necesidades de tecnología, ni otras restricciones.

El Modelo Lógico de datos se obtiene a partir del modelo conceptual resolviendo las relaciones complejas, eliminando redundancias y ambigüedades, identificando relaciones de dependencia, completando entidades y atributos, identificando las claves de cada entidad y detallando la cardinalidad. Asociado al modelo lógico se debe estimar el crecimiento de las entidades, su tipo y frecuencia de accesos, así como aquellas características relativas a la seguridad, confidencialidad, disponibilidad, etc., consideradas relevantes. Este tipo de modelos está ligado a la tecnología de bases de datos relacionales.

El Modelo Físico se obtiene a partir del modelo lógico de datos normalizado, analizando las peculiaridades técnicas del gestor de bases de datos a utilizar, estimando volúmenes y definiendo índices y otros elementos dependientes del gestor como, por ejemplo, el bloqueo y la compresión de datos, los clústeres, etc. En función del rendimiento exigido y las capacidades del gestor se pueden realizar desnormalizaciones y particiones en el modelo [METRICA3 2010].

3.2.3. Construcción de la CMDB

Una vez construido el modelo físico e implantado sobre una base de datos, es necesario realizar la inicialización de la base de datos con la información existente en la organización.

Para ello, **será necesario realizar un proceso de Autodescubrimiento de los ítems de configuración** (CIs). Este proceso, suele estar soportado por una serie de herramientas software especializadas en este cometido. La automatización de este proceso es imprescindible en empresas medianas y grandes. Incluso en pequeñas empresas, este proceso puede automatizarse haciendo uso de sencillas herramientas de inventario y de autodescubrimiento de CIs y sus relaciones.

Hoy día la práctica totalidad de los CIs están conectados a la red IP y son, por tanto, susceptibles de ser inventariados de forma automática. Existen dos tipologías de herramientas para este propósito: las basadas en la instalación de agentes en servidores y terminales y otras que interrogan a éstos mediantes APIs y comandos nativos disponibles en los dispositivos [J. GARCIA R. 2008].

Una vez obtenida la información de salida del proceso de autodescubrimiento y considerando el alcance de la CMDB, se procederá a poblar la base de datos con esta información [MODELWARE 2006]. Por último, y para concluir la construcción de la CMDB, será necesario diseñar un programa de mantenimiento y mejora continua de la CMDB.

4. La Estandarización de Bases de Datos. Aplicación a una CMDB

"Para crear un nuevo estándar, se necesita algo que
no solo sea un poco diferente,
se requiere de algo que sea realmente nuevo y
realmente capture la imaginación de la gente y la Macintosh,
de todas las máquinas que he visto,
es la única que cumple con ese estándar." Bill Gates.

Una vez presentada la metodología EODAM en el capítulo anterior, en este capítulo se detalla una propuesta para estandarizar el desarrollo de una base de datos y su aplicación a una base de datos de gestión de la configuración (CMDB). Esta estandarización complementa la metodología EODAM.

Por tratarse de una metodología orientada al desarrollo de líneas de productos, EODAM requiere la automatización de ciertos procesos. La estandarización nos permitirá reducir la complejidad de esta automatización, garantizando unas normas comunes en el diseño.

En las dos primeras secciones se trata el problema de la estandarización de los modelos de datos y de la CMDB. Posteriormente, se incluye una propuesta de estándar para modelar una CMDB. En la sección siguiente, se tratan las transformaciones entre los modelos. Finalmente, el capítulo incluye una propuesta de estándar para el diseño lógico y físico de la CMDB.

4.1.La importancia de los modelos y su estandarización

La complejidad de las plataformas crece de forma espectacular, mucho más rápido que la capacidad para tratar esta complejidad. Los lenguajes actuales, componentes, marcos de trabajo, middleware, servicios, etc. no nos permiten tratar la complejidad de forma satisfactoria y necesitamos de técnicas de abstracción más potentes, y que esencialmente se apliquen a las etapas iniciales del desarrollo.

Con el propósito de reducir la complejidad en este entorno, utilizamos los modelos y una aproximación de desarrollo basada en los mismos.

Un modelo es *una imagen simplificada de un sistema, y un sistema es un conjunto de elementos que interactúan* [MODELWARE 2006].

Una definición más completa de modelo y modelar la tenemos en [William et al. 1989]. *"Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality".*

Una propiedad importante que debe de cumplir un modelo es el principio limitado de sustitución [MODELWARE 2006]: "*Un modelo M se dice que es una representación de un sistema S para un conjunto de preguntas Q, si para cada pregunta del conjunto Q, el modelo proporciona la misma respuesta que el sistema S hubiera proporcionado respondiendo a la misma pregunta".*

Los modelos son claves en las primeras etapas en el desarrollo software y, en el caso del desarrollo de líneas de productos (SPL), su importancia es capital. En el caso de la metodología EODAM se plantea la construcción de varios modelos: el modelado del dominio, mediante un Lenguaje Específico del Dominio, que será tratado en próximos capítulos, y los modelos conceptual, lógico y físico de la base de datos. El desarrollo de estos modelos constituye una fase clave dentro de la construcción de bases de datos y dentro de la metodología EODAM.

Estandarizar las etapas de desarrollo dentro del ámbito de la Ingeniería de Dominio, el desarrollo de SPL y la programación generativa, que constituyen los fundamentos de la metodología EODAM, es una cuestión de gran importancia.

La primera tarea clave que se plantea es la estandarización de los modelos. A partir de los modelos, se obtiene información y al automatizar ciertos procesos, es vital contar con una normativa en la construcción de estos modelos.

Con ese propósito**, uno de los objetivos de la tesis es la estandarización de los modelos de construcción de bases de datos.**

En nuestro caso, aplicaremos la estandarización a una Base de Datos de Gestión de la Configuración (CMDB), pues nuestro objetivo es el desarrollo de servicios que den soporte a este elemento clave en las organización de gestión TIC. No obstante, la estandarización propuesta puede ser adaptada, tras un proceso de transformación, a cualquier tipología de base de datos.

4.2.La estandarización de la CMDB

La adaptación de la metodología EODAM a la construcción de líneas de productos para bases de datos de gestión de la configuración (CMDB) incluye una fase general de diseño de la CMDB que tiene tres actividades: el diseño conceptual, lógico y físico de la CMDB. Cada uno de estos diseños da lugar a un modelo que representa toda la información obtenida en las actividades de diseño.

En nuestra propuesta de estandarización de una CMDB, incluiremos un estándar general para su modelado, tres estándares específicos para cada uno de estos modelos y algún aspecto adicional, como el tratamiento de las claves subrogadas, **que complementará el estándar.**

4.3.Estándar General de Modelado de una CMDB

Cada norma o estándar está compuesto por una serie de normas de carácter más específico a implementar en los modelos. El Estándar General aplica a cualquier modelo de una CMDB, incluyendo los modelos conceptual, lógico y físico. **Estas normas están estructuradas en la siguiente tipología:**

- **Documentación obligatoria de objetos de la CMDB.**
- **Uso de caracteres en una CMDB.**
- **Nomenclatura de los objetos de una CMDB.**
- **Principio de unicidad de nombres.**
- **Siglas y acrónimos en una CMDB.**
- **Organización de modelos en una CMDB y Control de Cambios.**
- **Dominios en una CMDB.**
- **Reglas de Validación en una CMDB.**
- **Relaciones en una CMDB.**

4.3.1. Documentación obligatoria de objetos de la CMDB

En primer lugar, **todos los objetos de los modelos** de la CMDB (entidades o CIs, atributos, relaciones, etc.) **deben tener una documentación compuesta por**:

- **Nombre** del objeto.
- **Definición** del objeto.
- **Comentarios** o notas sobre el mismo. Este contenido es opcional.
- **Unidades de medida**, en caso necesario.

En segundo lugar, no se permite el **uso de tildes, diéresis y eñes** en el nombrado de los objetos de datos de la CMDB, pero sí en sus definiciones y comentarios. Se recomienda sustituir la 'ñ' por la construcción 'ny', como por ejemplo en 'anyo'. Se acepta la utilización de la 'ñ' sólo en los modelos conceptual y lógico, pues deberían ser validados por el responsable funcional y para mayor claridad se permite su uso. Sin embargo, es obligatorio que desaparezca en el paso al modelo físico, aunque el sistema o lenguaje de programación lo permita.

4.3.2. Uso de caracteres en una CMDB

Se utilizará un **juego de caracteres** restringido para la CMDB, para asegurar la portabilidad en entornos donde el juego de caracteres es reducido. El juego de caracteres permitido se debe basar en los admitidos para nombrar etiquetas de un documento XML [XML 2008], por considerarlo un estándar de facto independiente de cualquier plataforma.

La **tabla de caracteres** permitidos será

- Numéricos: 0-9 Códigos ASCII (48...57)
- Alfabéticos: A-Z, a-z Códigos ASCII (65..90) y (97-122)
- Subrayado _ Código ASCII (95)

El **juego de caracteres para las definiciones y comentarios** debe ser ISO-8859-1 [ISO-8859 1998].

4.3.3. Nomenclatura de los objetos de una CMDB

En lo que respecta al **nombre de cada objeto de la CMDB, se establecen las siguientes normas:**

- No hay un tamaño máximo preestablecido.
- No deben utilizarse números a menos que aporten un claro contenido semántico. Por ejemplo SERVIDOR3 no es válido, pero si lo sería LICENCIAWINDOSWS2000.
- Todos los nombres deben comenzar por letra.
- Usar el carácter subrayado como separador de palabras. Ej: SERVIDOR_NOMINAS.

- No se deben usar preposiciones, artículos ni conjunciones excepto en los casos siguientes:
  - Aportan un significado explícito. Ej: son correctos FECHA_LICENCIA_DESDE, FECHA_LICENCIA_HASTA, FECHA_COMPRA, pero no lo es FECHA_DE_COMPRA.
  - Forman parte de un término comúnmente reconocido en el contexto funcional del sistema y representan un único concepto en su totalidad. Es correcto BASES_DE_DATOS_NOMINA y sería incorrecto: SERVIDOR_DE_NOMINAS.
  - En el nombrado de relaciones.

- En nombres compuestos no usar el orden de las palabras para distinguirlo de otro nombre. Ej: Si existe el objeto SERVIDOR_NOMINAS no puede definirse NOMINAS_SERVIDOR.

- Limitar la utilización de verbos y en caso de usarse deben aparecer en infinitivo, participio o 3ª persona del singular del presente de indicativo. Ej: asignar, asignado, asigna. Excepción hecha del nombrado de relaciones.

- En el nombre no deben incluirse unidades de medida. Se indicarán obligatoriamente en la definición, descripción o comentarios del objeto de datos.

- No deben usarse aisladamente nombres propios o de organizaciones, estándares, sistemas informáticos, aplicaciones, directivas, formularios, pantallas o informes. Sólo se permiten nombres propios de organizaciones, estándares o directivas, siempre formando parte de un nombre compuesto. Ej: DIRECCIÓN_IP.

4.3.4. Principio de unicidad de nombres

Debe aplicarse el **principio de unicidad de nombres dentro del mismo nivel de abstracción de la CMDB** (en el caso de modelos conceptuales, dentro del mismo modelo y lo mismo aplicado al resto). Es decir, no está permitido repetir nombres de entidades (o tablas) y atributos (o columnas) en distintos modelos del mismo sistema en un determinado nivel de abstracción. Ejemplo: si el modelo de datos lógico de un sistema consta de dos modelos, no se pueden repetir nombres de entidades ni atributos en los dos modelos.

4.3.5. Siglas y acrónimos en una CMDB

Las reglas siguientes aplican a **abreviaturas, siglas y acrónimos:**

- Evitar en lo posible el uso de abreviaturas y acrónimos. En el caso de utilizarse, deben observarse las reglas siguientes y en este orden.

- El término al que corresponde debe estar documentado en el modelo, en la definición o en la descripción del objeto de modelo de datos (entidad, atributo, relación,...) en que se utilice.

- Debe ser comúnmente reconocido en el contexto funcional en el que se enmarca el sistema y, especialmente, por sus usuarios.

4.3.6. Organización de modelos y Control de Cambios en una CMDB

Como ya se ha mencionado con anterioridad, una CMDB es una base de datos federada, lo que significa que no todos los datos de configuración deben residir en una misma base de datos física. Podremos tener, en algunos casos, diveros modelos tanto conceptuales, lógicos como físicos [Clark et al. 2007].

Aunque cada uno de los modelos (el conceptual, el lógico o el físico) esté compuesto de varios submodelos, **la normativa propuesta no permite relaciones de entidades o tablas, en el caso de modelos lógicos o físicos, con entidades o tablas que no estén incluidas en el mismo modelo**, para evitar la complejidad del control de cambios.

Para modelos complejos, compuestos por múltiples submodelos, se recomienda utilizar las diferentes características que ofrecen las herramientas de modelado como las *áreas* o los *displays*, para facilitar la comprensión de los modelos y evitar, en la medida de lo posible, la generación de grandes "*sábanas"* o que los modelos estén dispersos en muchos modelos y ficheros.

Cada modelo será sometido a un control planificado de cambios y versiones y tendrá identificado, al menos, los siguientes campos:

- Nombre, con nombre que se le da al modelo.
- Autor, con el nombre de la persona o personas que han realizado el modelo.

- Definición, con una descripción general del modelo y cualquier información que se considere útil para su comprensión. Pueden incluirse explicaciones sobre las relaciones principales en el modelo, reglas de negocio que el modelo no puede expresar directamente, etc.

- Sistema/s, con el nombre del/los sistema/s al /los que pertenece el modelo. En la mayor parte de los casos, la CMDB se incluirá dentro del Sistema de Gestión de Configuración [Larry 2010].

En lo que se refiere a la Notación, **será obligatorio seleccionar IDEF1X** [Thomas 1992] **en el caso de modelos lógicos y físicos**. En el caso de modelos conceptuales no es obligatorio, aunque sí recomendable.

4.3.7. Dominios en una CMDB

Un **dominio** es un tipo de datos definido a partir de uno de los tipos de datos básicos de la metodología, técnica o herramienta utilizada, con significado en el contexto funcional del sistema y que se aplica a uno o más atributos del modelo [Date 2001].

En lo que se refiere a los dominios en una CMDB, la normativa propuesta recomienda que si un dominio se aplica a más de un atributo, debe escogerse un nombre significativo basado en el nombre de dichos atributos. Normalmente los atributos sólo difieren en el término de entidad, con lo que bastará con suprimirlo.

4.3.8. Reglas de Validación en una CMDB

Una **regla de validación** limita los valores posibles que puede tomar un atributo [Thomas M. et al. 2004]. En la mayoría de los casos estas reglas se establecen mediante una *lista de valores* posibles o especificando el valor mínimo y/o máximo que puede tomar el dato. Algunas veces se especifican mediante una expresión booleana (o condición) que deben cumplir los valores.

En lo que se refiere a las reglas de validación, si se aplica a más de un atributo, la normativa propone que se escoja un nombre significativo basado en el nombre de dichos atributos.

4.3.9. Relaciones en una CMDB

En una Base de Datos, una entidad representa un conjunto de ideas, abstracciones u objetos del mundo real agrupados en base a una serie de características comunes. En una CMDB, cada entidad representará un Ítem de Configuración (CI).

Una relación es una asociación o correspondencia entre dos entidades (o CIs). Se producen distintos tipos de relaciones para facilitar la representación de ciertas estructuras del mundo real: dependencia, generalización, especialización, jerarquía, categorización, agregación, etc.

El nombrado de relaciones se basa en construir frases cuyo sujeto sea una de las entidades que intervienen en la relación y su complemento la otra entidad.

La misma relación se puede nombrar con una frase "directa" o con una frase "inversa". Frase directa: "un SISTEMA está soportado por una o más BASE(S) DE DATOS", frase inversa: "una BASE DE DATOS soporta a un SISTEMA".

Las reglas siguientes aplican a las relaciones en la normativa propuesta:

- Escoger la frase directa o inversa que se considere que expresa mejor el significado de la relación, existiendo preferencia por la frase directa.
- Los nombres de las dos entidades implicadas NO forman parte del nombre la relación.
- Usar sólo minúsculas y el carácter de subrayado como separador de palabras.
- Se permite la repetición de nombres de relaciones en un mismo modelo, siempre que las entidades implicadas sean distintas.

4.4.Estándar para el Diseño Conceptual de una CMDB

En el modelo conceptual se identifican las entidades y atributos principales, claves candidatas, dominios y relaciones [Batini 2004]. El principal objetivo de este tipo de modelos es establecer el ámbito de información que gestionará el sistema. En una CMDB incluirá todos los Ítems de Configuración (CIs), dentro del alcance establecido.

Se recomienda utilizar, cuando sea necesario, **jerarquías de supertipos y subtipos** para representar ciertas estructuras del mundo real: generalización, especialización, categorización, herencia, etc. No es obligatorio utilizar una notación específica en este tipo de modelos, aunque se recomienda usar IDEF1X.

Las reglas siguientes aplican a las entidades de la CMDB:

- El nombre de una entidad debe corresponderse con un término del contexto funcional que se pretende modelizar, o con la combinación de varios.

- Los términos funcionales pueden ir acompañados de modificadores, normalmente adjetivos que detallan el significado de la entidad y, algunas veces, provienen de algún tipo de clasificación existente en el proceso de modelado.

- El orden recomendado es: <término funcional> [+ <modificador>]

Un atributo de entidad representa una característica abstracta o real que contribuye a describir las ocurrencias o instancias de esa entidad. **Las reglas siguientes aplican a los atributos:**

- Es necesario especificar su dominio, el tipo de dato, la regla de validación y el valor por defecto, siempre que esto sea posible.

- Al igual que en las entidades, se permite el uso de modificadores acompañando a cualquiera de los términos, aunque se recomienda minimizar su uso.

4.5.Las Transformaciones entre los Modelos

Una transformación es el proceso de generación de un modelo destino a partir de un modelo origen [Kleppe et al. 2003].

Para realizar este proceso es necesario especificar la definición de la transformación (en que va a consistir el proceso), esto es, cual es el conjunto de reglas que se van aplicar sobre el modelo de entrada para generar un modelo de salida. Estas reglas deberán ser ejecutadas por una herramienta, la herramienta de transformación.

Nos interesa que las transformaciones satisfagan un conjunto de propiedades [Kleppe et al. 2003]:

- Las transformaciones tendrían que ser configurables, adaptándose antes de ser aplicadas a un modelo.

- Las transformaciones deberían de ser trazables. Debemos de poder conocer que impacto tiene un elemento origen en un elemento destino, y dado un elemento destino conocer que elemento/s origen lo generaron.

- Las transformaciones deberían de mantener la consistencia en los modelos. Los cambios que realicemos en un modelo destino, no deberían de perderse al regenerar ese mismo modelo a partir de los modelos origen. Esto nos conduce también a una propiedad interesante que es la bidireccionalidad, que es la siguiente característica.

- Las transformaciones deberían de ser bidireccionales. Con este tipo de transformaciones, podríamos pasar de un modelo destino a un modelo origen.

Cuando se produce el paso de un modelo Conceptual un modelo Relacional (basado sólo en tablas) aparecen entidades asociativas y claves migradas. El modelo relacional sólo soporta directamente las relaciones 1:1 y 1:N. Para representar una relación M:N entre dos entidades es necesario crear una nueva entidad ficticia denominada entidad asociativa, con la que las dos entidades originales participantes en la relación establecen relaciones 1:N.

Para representar las relaciones 1:1 y 1:N es necesario "migrar" la clave del lado 1 al otro lado. Por migración se entiende incluir el atributo clave de la entidad del extremo 1 de la relación (también llamado padre) en la entidad del extremo N (también llamado hijo de la relación).

La clave migrada pasa a ser una clave ajena (foreign key) en la entidad a la que se ha migrado. En el caso particular 1:1 las claves de cada entidad se migran a la otra, es decir, se cruzan las claves.

En el paso del modelo lógico al modelo físico, será necesario implantar los objetos del modelo de datos lógico, adaptándose a las limitaciones de un determinado sistema software (RDBMS, sistema operativo, lenguaje, etc.).

También suele producirse un proceso de desnormalización, a causa del cual se producen nuevos objetos de modelos de datos de nivel físico. Por último, aparece la necesidad de identificar objetos específicos de modelos físicos como índices, secuencias, constraints, etc.

Una de las limitaciones habituales en el paso al modelo físico de datos se refiere al tamaño máximo de los nombres de las tablas, columnas, campos, etc. Este acortamiento de los nombres también suele obedecer a razones prácticas que facilitan la escritura del código, aunque no debe abusarse, pues hace más difícil su comprensión. Por tanto, el diseñador se verá obligado a acortar los nombres de los objetos de datos del modelo lógico, normalmente a través del uso de abreviaturas.

Para superar estos aspectos, se detallarán normas específicas para cada uno de los modelos (lógico y físico).

4.6.Estándar para el Diseño Lógico de una CMDB

El Modelo Lógico de datos debe derivar del modelo conceptual resolviendo las relaciones complejas, eliminando redundancias y ambigüedades, identificando relaciones de dependencia, completando entidades y atributos, identificando las claves de cada entidad y detallando la cardinalidad.

Deben incluirse todas las entidades y todos los atributos, no sólo los más significativos. No se permiten las relaciones "muchos a muchos", deben estar resueltas con las entidades asociativas correspondientes.

Una **clave primaria** es aquella columna (pueden ser también dos columnas o más) que identifica únicamente a esa fila. La clave primaria es un identificador que va a ser único para cada fila [Petersen 2002]. En multitud de ocasiones la clave primaria es auto-numérica. En una tabla puede que tengamos más de una clave, en tal caso se puede escoger una para ser la clave primaria, y las demás claves son las **claves candidatas** [Peter et al. 2000].

Una **clave ajena** (foreign key o clave foránea, FK) es aquella columna que existiendo como dependiente en una tabla, es a su vez clave primaria en otra tabla [Mark and George 1999]. Una **clave alternativa** es aquella clave candidata que no ha sido seleccionada como clave primaria, pero que también puede identificar de forma única a una fila dentro de una tabla [Peter et al. 2000].

Según la normativa propuesta, las claves foráneas (FK's) (una clave foránea es un campo de una tabla que contiene una referencia a un registro de otra tabla) deben estar migradas e identificadas. Las claves alternativas deben estar definidas e identificadas.

No es obligatorio resolver las jerarquías de supertipos y subtipos [Toby 1999]. Se recomienda no hacerlo en este nivel de abstracción.

Como ya se mencionó anteriormente, es obligatorio usar notación IDEF1X.

Se impone que el modelo esté, como mínimo, en 2ª forma normal (FN) y, muy recomendable, en 3ª FN. [Ken 1991].

Las reglas siguientes aplican a las tablas del Modelo Lógico de la CMDB:

- El nombre de una tabla debe corresponderse con un término del contexto funcional que se pretende modelizar, o con la combinación de varios.

- Los términos funcionales pueden ir acompañados de modificadores, normalmente adjetivos que detallan el significado de la tabla y, algunas veces, provienen de algún tipo de clasificación existente en el proceso de modelado.

- El orden recomendado es: <término funcional> [+ <modificador>].

Estas reglas fueron también incluidas en las entidades del modelo conceptual.

Las reglas siguientes aplican a las columnas del Modelo Lógico de la CMDB:

- Es necesario especificar su dominio, el tipo de dato, la regla de validación y el valor por defecto, siempre que esto sea posible.

- Al igual que en las tablas, se permite el uso de modificadores acompañando a cualquiera de los términos, aunque se recomienda minimizar su uso.

- En el comentario del objeto (columna), si es una clave ajena (foreign key) indicar la tabla y columna referenciadas.

La entidad que resulta de considerar una interrelación entre entidades como si fuese una entidad es una **entidad asociativa** [Covadonga 1987].

Las reglas siguientes aplican a las entidades asociativas de la CMDB:

- Aplican las mismas reglas que para las tablas y las entidades del modelo conceptual.

- Como recomendación, incluir los nombres de las dos entidades relacionadas.

- Opcionalmente, puede añadirse un modificador que añada claridad al modelo, haciendo referencia a la relación M:N soportada. El modificador es obligatorio en el caso de que exista más de una relación M:N entre las dos entidades participantes, para asegurar la unicidad en el modelo.

- En el caso que la entidad asociativa proceda de una relación M:N reflexiva (las que se producen entre dos ocurrencias de la misma entidad), es obligatorio el uso de un modificador acompañando al nombre de la entidad relacionada. Ejemplo: dada la relación M:N reflexiva esta_contectada_con sobre SERVIDOR, el nombre de la entidad asociativa podría ser SERVIDOR_CONECTADO_CON.

Las reglas siguientes aplican a las claves migradas:

- Aplican las mismas reglas que para las tablas y las entidades del modelo conceptual.

- Usar el nombre original de la clave migrada desde la entidad que actúa como padre de la relación. Esta regla supone una excepción al principio de unicidad, ya mencionado con anterioridad.

- Opcionalmente, puede añadirse un modificador indicando el rol desempeñado en la relación que justificó el migrado de la clave, para ratificar el significado del atributo.

- Es obligatorio incluir modificadores cuando la clave migrada corresponde a una relación reflexiva o bien sea necesario migrar la clave más de una vez sobre la misma entidad destino.

4.7.Las Claves Subrogadas en una CMDB

Una **clave subrogada** es una clave primaria que sustituye a una clave natural y que, desde el punto de vista del negocio, se ve como un identificador único [Steve 2009].

Sobre el tema de la **utilización de claves subrogadas**, muchos proponentes del modelo relacional, entre ellos Date [Date 1991] opinan que usar una clave natural puede llegar a desorganizar una base de datos si se producen cambios en los datos y abogan por la utilización exclusiva de claves subrogadas; para estos autores este tipo de claves tiene el beneficio de poder ser modificadas con más facilidad.

Claro que siempre se pueden esgrimir argumentos a favor del uso de claves semánticas. Por ejemplo, podemos argumentar que las claves semánticas son más intuitivas, y pueden evitar muchas consultas [Celko 1994]. No obstante, gran parte de los autores confirma que este argumento a favor, se vuelve en contra cuando las claves naturales están formadas por una cantidad no despreciable de atributos.

Usar una clave semántica es una buena idea siempre que se tenga la absoluta certeza de que la situación generada por su utilización no es susceptible de cambiar con el tiempo y que realmente facilite el manejo de las relaciones en lugar de dificultarlo.

Por otra parte, las claves subrogadas, al estar compuestas por un sólo atributo, que suele ser numérico, ahorran espacio de almacenamiento, lo que hay que tener en cuenta cuando el sistema es susceptible de crecer. En el caso de la CMDB, se recomienda la utilización de **claves subrogadas numéricas (o autonuméricas)**, que deben estar definidas e identificadas.

Para mayor claridad, y con el objetivo de evitar la propagación de numerosos atributos en las claves migradas, también recomendamos utilizar **un Identificador único** (clave subrogada) para algunas de las Entidades. De esta forma, el modelo estará compuesto por Claves Primarias que, únicamente, están formadas por uno o, como máximo, dos atributos.

4.8. Estándar para el Diseño Físico de una CMDB

El modelo físico se debe derivar del modelo lógico de datos normalizado, analizando las peculiaridades técnicas del gestor de bases de datos a utilizar. En el modelo físico deben incluirse las claves alternativas y las entradas inversas ("*inversion entries*") [Erwin 2009].

Las jerarquías de supertipos y subtipos deben estar resueltas, habiendo aplicado las transformaciones necesarias. Como ya se ha mencionado anteriormente, es obligatorio usar la notación IDEF1X.

Las reglas siguientes aplican a las tablas del Modelo Físico de la CMDB:

- El nombre de una tabla debe corresponderse con un término del contexto funcional que se pretende modelizar, o con la combinación de varios.

- Los términos funcionales pueden ir acompañados de modificadores, normalmente adjetivos que detallan el significado de la tabla y, algunas veces, provienen de algún tipo de clasificación existente en el proceso de modelado.

- El orden recomendado es: <término funcional> [+ <modificador>].

Estas reglas fueron también incluidas en las entidades del modelo conceptual y las tablas del modelo lógico.

Las reglas siguientes aplican a las columnas del Modelo Físico de la CMDB:

- Es necesario especificar su dominio, el tipo de dato, la regla de validación y el valor por defecto, siempre que esto sea posible.

- Al igual que en las tablas. se permite el uso de modificadores acompañando a cualquiera de los términos, aunque se recomienda minimizar su uso.

- El orden de creación de las columnas dentro de una tabla debe responder a la clasificación siguiente: primero, las columnas que forman parte de la clave primaria, luego las columnas con valores obligatorios (definidas como *'not null'*) y, por último, las columnas con valores opcionales (definidas como *'null'*).

- Opcionalmente, todos los nombres de columna pueden estar precedidos por una abreviatura del nombre de la tabla, seguida del carácter subrayado.

El nombre de un objeto de datos del modelo físico de la CMDB no puede incluir nombres de palabras reservadas del lenguaje de definición y manipulación de datos del RDBMS, usualmente Structured Query Language (SQL) [James and Paul 2002] con extensiones propietarias de cada fabricante como Transact-SQL [Kevin et

al. 1999], PL/SQL [Steven and Bill 2005] [Ewal and Hangs 2002] o FSQL [Galindo et al. 2006].

No está permitido usar el carácter 'ñ', aún cuando sí se haya utilizado en el modelo lógico; se recomienda sustituirlo por la construcción 'ny'.

Los **disparadores o triggers** son programas asociados a determinados eventos producidos en la base de datos (es decir, se ejecutan o "disparan" cuando sucede un determinado evento) y que suelen utilizarse para implementar reglas del negocio que no pueden ser soportadas directamente por el modelo relacional.

Los disparadores deben nombrarse con un nombre lo más significativo posible, con todas las letras en mayúsculas, y para los triggers asociados a eventos de acceso a una tabla (select, insert, update, delete) [Manish et al. 2004] debe incluirse el nombre de la tabla.

En sistemas RDBMS donde es necesario asignar un nombre a **la instancia de base de datos** [Sam R. 2008] se debe escoger un nombre lo más significativo posible desde el punto de vista funcional, con todos los caracteres en mayúsculas y evitar asignar nombres de máquina, de red, o con un significado demasiado genérico.

Para las restricciones de clave primaria (primary keys) y clave única (unique keys) se recomienda identificarlas con el nombre de la tabla asociada. En el caso de varias claves únicas, se pueden nombrar de forma secuencial.

Para las restricciones de clave ajena (foreign keys) se recomienda incluir en el nombre tanto la tabla restringida (el nombre de la tabla donde se establece la restricción de clave ajena: hijo, detalle o lado N de una relación 1:N) como la tabla

referenciada (la tabla que contiene la columna referenciada: padre, maestro o lado 1 de una relación 1:N).

Siempre que el RDBMS lo permita no deben acortarse los nombres lógicos, permaneciendo tal el cual en el paso al modelo físico. **En caso de que sea necesario acortar los nombres, se aplicarán las reglas siguientes:**

- Si el nombre lógico se compone de varias palabras, se acortará cada palabra por separado, conservando los caracteres de subrayado de separación (_).

- Para acortar cada palabra se usará, en primer lugar, el acrónimo o abreviatura ampliamente reconocido en el entorno funcional del sistema.

- Si después de aplicar las abreviaturas anteriores no se cumple con la restricción de tamaño máximo establecida, se acortará cada palabra por separado, cumpliendo las siguientes condiciones:

  - No debe desaparecer ninguna de las palabras originales del nombre lógico, y en el peor caso se dejará, al menos, su letra inicial.

  - En la documentación del modelo se detallarán todas las abreviaturas generadas.

  - La abreviatura utilizada para una palabra se usará en todos los nombres de objetos del modelo donde sea necesario. Es decir, no se permite que para una misma palabra se use más de una abreviatura en el mismo modelo.

5. La Ingeniería de Requisitos y el Análisis del Dominio

"Tenemos que cambiar la tradicional actitud
ante la construcción de software.
En vez de pensar que nuestra principal tarea es
indicar a un ordenador qué hacer,
concentrémonos en explicar a las personas
lo que queremos que el ordenador haga"
Donald E. Knuth

En los dos capítulos anteriores se han detallado la metodología original EODAM y una propuesta para la estandarización del desarrollo de bases de datos y su aplicación al desarrollo de bases de datos de gestión de la configuración. Algunas de las fases de la metodología EODAM incluyen aspectos relacionados con la ingeniería de requisitos y el análisis del dominio bajo estudio, que en nuestro caso es el desarrollo de un servicio de comunicación de los cambios que se producen en la infraestructura de TIC al resto de procesos de negocio relacionados con la gestión de las TIC. En este capítulo se describen todas estas fases.

Las primeras secciones del capítulo incluyen la Definición del Dominio bajo estudio, la Identificación y Clasificación de los Requisitos y el Análisis de los Procesos de negocio relacionados con el Dominio.

Posteriormente se seleccionan una serie de requisitos para el desarrollo de un ejemplar del dominio y se lleva a cabo un análisis de la flexibilización de este ejemplar, incluyendo un análisis de las funcionalidades necesarias para cubrir los requisitos expuestos.

Finalmente, se trata brevemente el análisis de la rentabilidad que será abordado más en detalle en capítulos posteriores.

5.1.La Ingeniería de Requisitos en EODAM

En la metodología EODAM (Exemplar Oriented Database Methodology) la ingeniería de requisitos juega un rol primordial. **Las fases principales de EODAM son**:

1. **La construcción de la Base de Datos,**
2. **el desarrollo de un Ejemplar y su Flexibilización y**
3. **la obtención del resto de los Productos de la línea.**

En todas estas fases se distinguen varias etapas que comprenden todo el alcance de la Ingeniería de Requisitos.

En primer lugar, durante la **construcción de la Base de Datos** se realiza el análisis de la base de datos que concluye con el **modelo conceptual** de la misma, siguiendo la normativa expuesta en el capítulo anterior.

Además, para el caso de la aplicación de EODAM a una base de datos de gestión de la configuración (CMDB), como aspecto adicional, se propone el desarrollo de un **Bussines Case** y la identificación de los requisitos y estándares que permitan delimitar el alcance de la CMDB.

Posteriormente, se identifican los interfaces que interactuarán con la CMDB. Todas estas fases relacionadas con la construcción de la CMDB serán expuestas en detalle en el capítulo relativo a la Construcción de la CMDB.

En segundo lugar, **durante la flexibilización y construcción de un ejemplar de la línea de productos, se lleva a cabo una fase que incluye una serie de actividades directamente relacionadas con la ingeniería de requisitos**: el análisis del dominio. Para mayor claridad, a continuación se muestra, como resumen, las actividades que tienen lugar en esta fase:

1. **Definición del Dominio de la Familia.**
2. **Identificación y Clasificación de Requisitos, que incluye un Análisis de procesos relacionados con el Dominio.**
3. **Modelado del Dominio y Análisis de Rentabilidad.**
4. **Selección de Requisitos para el Ejemplar.**
5. **Modelo del Dominio.**
6. **Implementación de la Flexibilización del Ejemplar.**

A lo largo del presente capítulo se describen estas actividades.

En EODAM, durante la etapa final de esta fase, **todos los requisitos identificados se incluirán en un modelo, que constituirá el modelo del dominio**. Desde el punto de vista metodológico, aunque el modelo sea único, dependiendo de las necesidades y complejidad del mismo, éste podrá ser estructurado en varios submodelos para facilidad de uso.

Desde EODAM se propone que el modelo inicial de requisitos sea implementado como un **modelo de objetivos** (*goals*) a cubrir por el dominio bajo estudio y que se desarrolla durante la fase de **Definición del Dominio de la Familia de Productos.**

Los objetivos (requisitos) a cubrir **serán de tipo *hard***, entendidos como objetivos funcionales. Los objetivos *soft* (objetivos no funcionales, no completamente demostrables) no formarán parte del dominio bajo estudio.

No se incluirán *tareas* (maneras de alcanzar los objetivos), *agentes* (entidades que utilizan tareas para alcanzar objetivos) ni *recursos* (recursos necesarios para alcanzar las tareas), entendiéndose éstos de un nivel de abstracción menor que los requisitos. Algunos de estos componentes, tratados dentro de la disciplina de ingeniería de requisitos, serán realizados con posterioridad en otras fases del proceso EODAM.

El modelo de objetivos (requisitos) podrá ser realizado con cualquier herramienta de modelización. En nuestro caso, se ha utilizado la herramienta QSEE-SuperLite de QSEE-Technologies [QSEE1]. Esta herramienta permite la creación genérica de modelos y metamodelos, es de fácil uso y gratuita. Además, soporta UML, DFD's, XML Schema, etc. [QSEE2].

Una vez finalizado el modelo, tal y como se propone en EODAM, se realizará la flexibilización del ejemplar mediante varias actividades: el análisis de funcionalidades, de elementos y de inconsistencias.

5.2. Definición del Dominio de la Familia

El Dominio de la Familia lo constituyen todos los productos software para dar soporte a un servicio de notificación de cambios en una base de datos de gestión de la configuración (CMDB). El requisito de más alto nivel, la notificación de cambios, por resumir, consiste en la implementación de un observador de los cambios producidos en la base de datos (CMDB) y de una serie de sistemas que se suscriben a este observador para conocer los cambios que se producen en la misma.

Las bases de datos, en general, disponen de mecanismos que permiten el desarrollo de productos para dar soporte a la notificación de cambios como los Sistemas Avanzados de Gestión de Colas, los Sistemas de Gestión de Pipes o los Sistemas de Gestión de Alertas y Señales [Coz et al. 2008]. Se pueden desarrollar soluciones ad hoc, pero los mecanismos que ponen a disposición las bases de datos permiten optimizar los desarrollos.

En el caso de los sistemas de alertas y señales y las pipes, cuando se produce un cambio en la base de datos, el servicio de notificación de cambios transmite los mensajes en tiempo real, pero no existe persistencia ni notificaciones de cambios en diferido [PIPES Oracle] [PIPES SQLS] [S.A. Oracle] [S.A. SQLS]. Esta puede ser una solución para muchos casos.

En el caso de los sistemas de gestión avanzada de colas, el servicio puede proporcionar persistencia y capacidad de realizar notificaciones de cambios en diferido [AQ Oracle] [AQ Oracle Java] [AQ SQLS]. Además, los sistemas de gestión avanzada de colas son la solución más completa, ofreciendo diferentes funcionalidades de gestión del tiempo, prioridad, etc. que posteriormente son analizadas en profundidad.

¿Qué tipos de cambios son objeto de este tipo de servicios? En principio, cualquier cambio que se produzca en la base de datos. A continuación, se desglosan los tipos de cambios más habituales que tienen lugar:

- Modificación de datos. Es el tipo de cambio más habitual. Un sistema o un usuario modifica uno o varios registros de la base de datos.

- Inserción de nuevos datos. Un sistema o un usuario inserta nuevos registros en la base de datos.

- Borrado de datos. Un sistema o un usuario borra registros de la base de datos.
- Acceso a la base de datos. Un usuario accede a la base de datos.
- Desconexión de la base de datos. Un usuario que ha accedido a la base de datos se desconecta.
- Arranque de la base de datos. La base de datos se arranca.
- Cierre de la base de datos. La base de datos se cierra.
- Modificaciones masivas de datos. Se producen modificaciones masivas de la base de datos, por ejemplo, en una importación de datos de una base de datos, durante la población inicial de la base de datos o en un proceso de migración de datos.

El observador, cuando se produce un cambio, notifica a aquellos usuarios o sistemas que estén suscritos al servicio de notificación de cambios. En función de las necesidades, se desarrollarán diferentes funcionalidades específicas. A continuación mostramos un resumen de este servicio.

El servicio de notificación de cambios constituye un único paquete con dos casos de uso: los tipos de cambios y la notificación de mensajes.

En lo que se refiere a la tipología de los cambios, cualquier usuario o sistema que utilice la base de datos y disponga de los permisos suficientes podrá realizar un cambio sobre la misma como eliminar datos, actualizarlos, insertar datos nuevos, contectarse a la base de datos o desconectarse.

Una vez realizada la operación, es necesario desarrollar un servicio que informe al observador de los cambios. Existen otros cambios como arrancar o cerrar la base de datos, o un cambio masivo, que también podrían ser informados. En la siguiente figura se muestra este caso de uso.

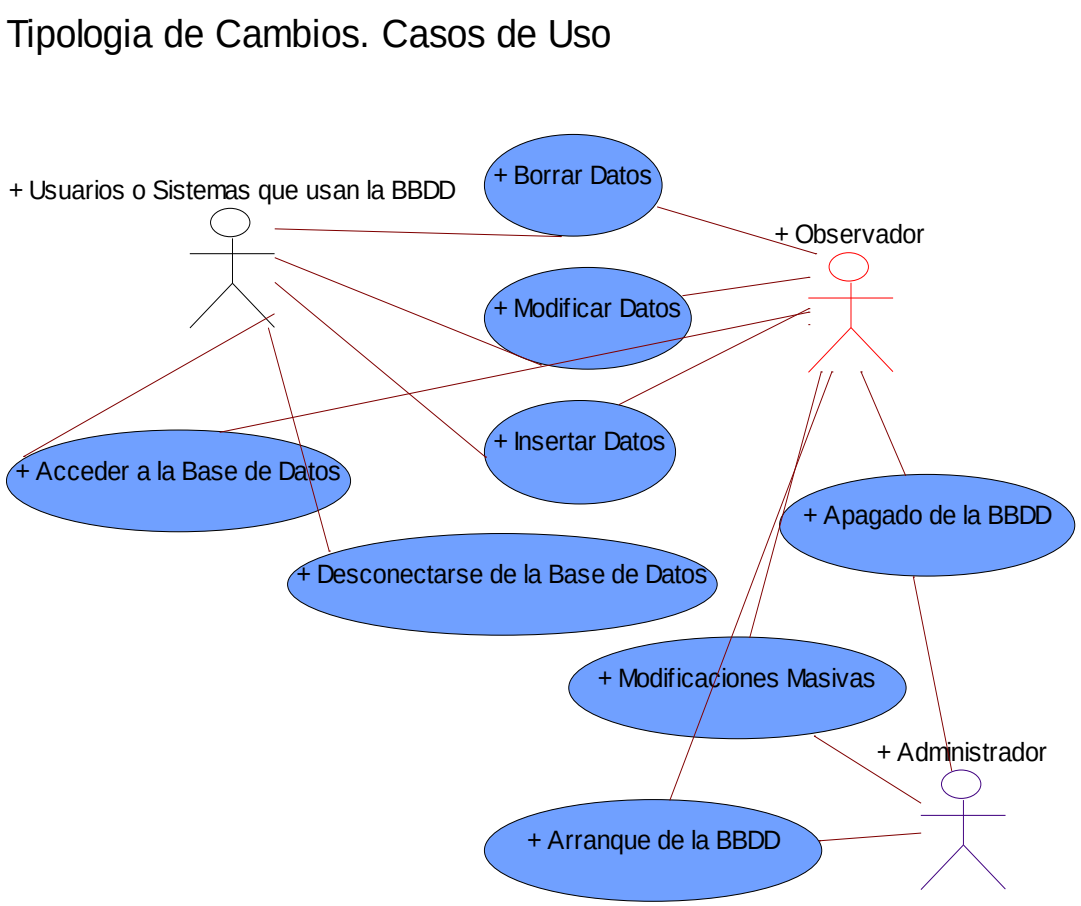


Figura 5-1 Tipos de Cambios. Casos de Uso.

Una vez que el observador conoce el cambio, procederá a informar al Servicio de Notificaciones.

Todos los usuarios o sistemas que estén interesados en ser comunicados sobre los cambios se suscribirán a este servicio.

El servicio, en base a las funcionalidades desarrolladas, notificará a los suscriptores del servicio. En la siguiente figura se muestra este caso de uso.

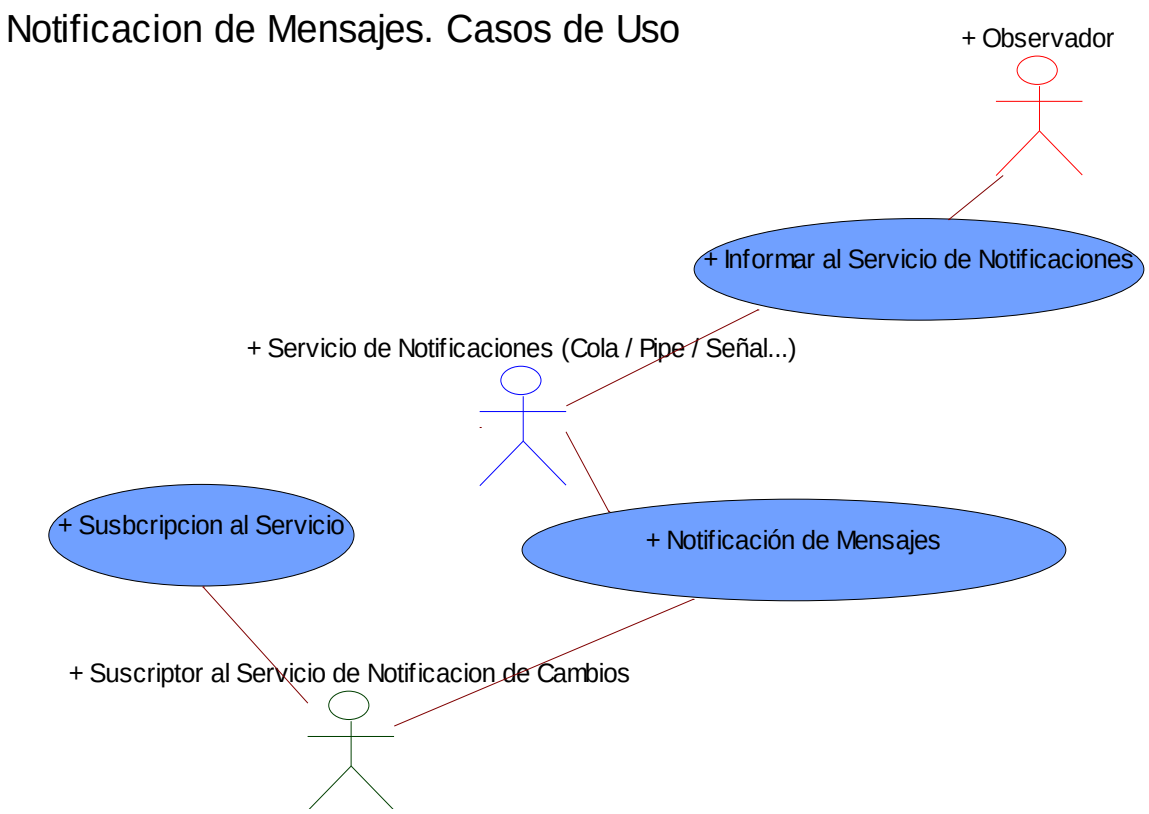


Figura 5-2 Tipos de Cambios. Notificación de Mensajes.

5.3.Identificación y Clasificación de Requisitos

Cualquiera que sea la solución elegida para la construcción de la CMDB: integración de varios repositorios de datos, un único repositorio de datos centralizado, varios repositorios de datos federados con un depósito central de datos, etc., en el dominio de la notificación de cambio es necesario tener en cuenta varios requisitos que deben ser analizados.

No es el propósito de esta sección exponer todos ellos, tan sólo dar una visión general de los más relevantes que han sido analizados.

5.3.1. Gestión del tiempo

El conjunto inicial de las necesidades del servicio de notificación de cambios en una CMDB, está relacionado con la gestión del tiempo.

Este conjunto de requisitos incluye, a su vez, varios requisitos más específicos. El primero de ellos es la **gestión de las retenciones**. En el dominio de notificación de cambios, y mediante mecanismos que permitan la persistencia de los mensajes y la transmisión en diferido como la gestión avanzada de colas, las retenciones se utilizan para “retener” de forma controlada los mensajes ya notificados.

Los mensajes de notificación permanecen en "activo" durante un tiempo que puede ser finito o no, y pueden ser vueltos a notificar. Este es el concepto de retención. Es el tiempo que los mensajes permanecen en la cola después de ser notificados para poder volver a notificarse. Podemos tener, en términos generales, tres tipos de requisitos:

- Gestión del Tiempo sin Retención (GTSR).
- Gestión del Tiempo con Retención (valor determinado) (GTCR).
- Gestión del Tiempo con Retención Infinita (GTCRI).

Cada uno de los posibles requisitos del dominio nos dará lugar a un conjunto de productos. Con el propósito de identificar los requisitos para, posteriormente, documentar y construir los correspondientes modelos, asignaremos unas siglas a cada uno de ellos.

Pongamos un ejemplo de retención que nos podrá facilitar su entendimiento. Nuestra CMDB está conectada a varios sistemas: el sistema de gestión financiera, el sistema de gestión de la configuración y el sistema de gestión del personal, entre otros.

Cuando se incorpora un nuevo empleado en la empresa, el sistema de gestión del personal notifica al sistema de gestión financiera para que se adquiera un nuevo Terminal. Una vez el Terminal es adquirido y configurado se inserta un nuevo registro en la CMDB, a través del sistema de gestión de la configuración. Hay un nuevo activo en servicio y debe de ser incorporado en la CMDB.

El servicio de notificación de cambios informa al sistema de gestión financiera de que el terminal adquirido ya está en uso, para dar por cerrado el proceso de adquisición. Esta notificación no tendría retención (caso GTSR), porque una vez notificado el mensaje desparecerá de la cola de mensajes.

Otro suscriptor a esta notificación, además del sistema del área financiera, es el sistema de mantenimiento. El sistema de mantenimiento esperará 72 horas antes de volver a ser notificado, porque es el tiempo en el que el terminal está a prueba como garantía de su funcionamiento. En este caso, sería Gestión del Tiempo con Retención (valor 72 horas) (caso GTCR).

Otro aspecto que incluye la gestión del tiempo es la **gestión de los retrasos y las expiraciones**. Los mensajes a notificar, y que se encuentren en la cola de mensajes, pueden tener varios estados: *en espera, listo y expirado*. Sólo los mensajes que están en estado *listo* se desencolan. Para pasar de un estado a otro se utilizan los tiempos de expiración y de retraso. *En espera* está el mensaje hasta que ha pasado el tiempo de retraso, entonces está *listo* y después de pasar el tiempo de expiración está en estado *expirado*.

Este requisito, denominado en nuestro análisis como **Gestión General del Tiempo** (GGT), podrá tener 4 valores:

- Gestión General del Tiempo Sencilla (no hay expiración ni retraso) (GTS).
- Gestión General del Tiempo con Expiración (GTE).
- Gestión General del Tiempo con Retraso (GTR).
- Gestión General del Tiempo con Expiración y Retraso (GTER).

Pongamos otro ejemplo para la Gestión General del Tiempo, que facilitará su entendimiento. En una situación similar al ejemplo anterior, nuestra CMDB incluye los activos de soporte a los sistemas del área de recursos humanos y financiera de nuestra organización. Tenemos como requisito que una vez alguien se da de baja en la organización, transcurridas 72 horas, se le da el finiquito al empleado.

El terminal del usuario se da de baja en el sistema de gestión de la configuración y el servicio de notificación de cambios espera 72 horas para notificar la baja al sistema del área financiera (tiempo para poder preparar el finiquito), mientras tanto el mensaje está en situación de *espera*. Una vez transcurridas las 72 horas lo notifica. En este caso, tendremos una Gestión General del Tiempo con Retraso (GTR), con valor de 72 horas.

En este mismo caso, para los empleados que no tienen derecho a finiquito, el sistema financiero consulta al servicio de notificación de cambios a las 100 horas, y si no recibe respuesta es porque el empleado no tiene derecho a finiquito. El tiempo de expiración será de 28 horas. El mensaje en 72 horas pasará a estado *listo* y 28 horas después expirará. En este caso, tendremos una Gestión General del Tiempo con expiración de 8 horas y retraso de 72 horas (GTER).

Otro componente de la gestión del tiempo es la **gestión de los tiempos de espera**. Los tiempos de espera son el tiempo transcurrido desde que la CMDB conoce un cambio hasta que ese cambio se introduce en la cola de mensajes a notificar.

Podemos tener varios casos:

- Sin Esperas. El mensaje se encola inmediatamente (DSE).
- Con valor. El mensaje espera a encolarse un tiempo predeterminado (DECV).
- Indefinidas. El mensaje espera a encolarse indefinidamente (DEI).

Podríamos tener en consideración, para este requisito, un ejemplo similar al anterior. La diferencia entre el retraso y la espera es que en este último caso, el mensaje no se incorpora en la cola de mensajes hasta que no pasa el tiempo de espera, mientras que en la gestión de retrasos el mensaje se introduce ya en la cola y puede ser notificado a los diferentes suscriptores del servicio de notificación de cambios en un tiempo determinado.

5.3.2. Gestión de suscriptores

El segundo conjunto de requisitos analizado es la gestión de los abonados al mecanismo de notificación de cambios. Puede haber un solo suscriptor, como por ejemplo pueda ser un sistema de mensajería que realice posteriormente la función de notificaciones de forma controlada, o puede haber varios suscriptores a los que se le notifique acerca de los cambios en la CMDB, que será el caso más común:

- Subscriptor Único (SU).
- Suscriptores Múltiples (SM).

Como se analizará posteriormente en el capítulo correspondiente a la construcción de la CMDB, existirán un conjunto de sistemas candidatos a ser los suscriptores al mecanismo de notificación de los cambios en la CMDB como el Sistema de Gestión de Activos, el Sistema de Gestión de Incidentes y Problemas, el Sistema de Gestión del Cambio y otros. Será necesario identificar todos estos sistemas suscriptores a este servicio en nuestra organización. Este paso está identificado en la metodología EODAM como una de las actividades de la Construcción de la CMDB: *Análisis de Interfaces de la CMDB.*

Como paso preliminar, será necesario analizar los procesos que interactúan con la CMDB. Esta tarea será realizada posteriormente en esta misma sección (*Análisis de Procesos relacionados con el Dominio*).

5.3.3. Granularidad

El tercer grupo de requisitos es la granularidad de la solución ofrecida por el servicio de notificación de cambios. Podemos distinguir, a grandes rasgos, tres tipologías:

- Granularidad de Grano Grueso (GG).
- Granularidad de Grano Medio (GM).
- Granularidad de Grano Fino (GF).

En el caso de la Granularidad de Grano Grueso (GG), la notificación de cambios se hace a nivel de la CMDB. Es decir, solamente operaciones generales sobre la CMDB serán notificadas. Por ejemplo, cuando un sistema accede a la CMDB el servicio envía una notificación a un sistema de auditoría, cuando se inserta un nuevo dato en la CMDB se notifica a un sistema de mensajería y cuando se cae o vuelve a arrancar la CMDB se envía una notificación a un servicio de gestión de la configuración. Este será, probablemente, el caso menos frecuente en nuestra organización.

Nos podemos encontrar con una granularidad de grano medio (GM), cuando se precise una notificación de cambios a nivel de entidad o tabla de la CMDB. Cualquier cambio, en determinadas tablas, será notificado. Por ejemplo, cuando hay un cambio de software en un sistema financiero (por ejemplo, ha entrado en producción la versión 2.0 del sistema) y éste se refleja en la correspondiente tabla de la CMDB, el servicio envía una notificación al sistema de Helpdesk para que los operadores lean el correspondiente manual de uso del sistema y puedan dar soporte sobre el mismo a los usuarios.

Hay multitud de ejemplos, pues eéste será, probablemente, el caso más frecuente en nuestra organización: una notificación de cambios a nivel de entidad.

Finalmente, podemos tener una solución de grano fino (GF), donde se requiera una notificación de cambios a de nivel registro de la CMDB. Cualquier dato cambiado podrá ser notificado. Siguiendo el mismo ejemplo anterior, solamente cuando cambie determinado interfaz gráfico del sistema financiero el sistema notificará este hecho a determinados usuarios con un rol de acceso a este interfaz gráfico.

Este caso, por su complejidad, no será tan frecuente, no obstante podría nuestra organización precisar algún producto software muy específico para este fin.

5.3.4. Gestión de prioridades

El cuarto grupo de requisitos analizados es la gestión de prioridades. Cada uno de los mensajes a ser notificado podría tener una prioridad. En términos generales nos encontramos con dos tipologías:

- Mensajes Sin Prioridad (MSP).
- Mensajes Con Prioridad (MCP).

Por ejemplo, un cambio en un componente de software del sistema financiero reflejado en la CMDB puede ser considerado muy crítico y, por tanto ser notificado por el servicio a los operadores de Helpdesk en primer lugar, mientras que un cambio de hardware del sistema de control de entrada al edificio, y que también esté reflejado en la CMDB, puede ser considerado mucho menos crítico, y ser notificado en último lugar.

5.3.5. Gestión de la Visibilidad

El quinto grupo de requisitos determina la manera en que los mensajes se encolan o desencolan. En el caso del encolado, la visibilidad establece la forma que en se encolan los mensajes. Puede ser de tipo transaccional o inmediato. En el primero de los casos, hasta que la transacción no se ha realizado, el mensaje no se encola.

En el segundo, se encola antes de producirse la transacción:

- Encolado con Visibilidad Inmediata (EVI).
- Encolado con Visibilidad Transaccional (EVT).

En el caso del desencolado, la visibilidad determina la forma que en se desencolan los mensajes. Puede ser de tipo transaccional o inmediato. En el primero de los casos, hasta que la transacción que tenga lugar en ese momento no se haya realizado, el mensaje no se desencola. En el segundo, se desencola antes de producirse la transacción:

- Desencolado con Visibilidad Inmediata (DVI).
- Desencolado con Visibilidad Transaccional (DVT).

En función de las necesidades, puede que nos interese no notificar los mensajes, hasta que realmente la transacción no haya ocurrido, para tener una total seguridad de que se han producido los cambios, o puede que nos interese notificarlo de forma inmediata.

Estos requisitos están relacionados con el mecanismo de gestión avanzada de colas.

5.3.6. Gestión de la Navegación

Otro grupo de necesidades se refiere a la gestión de la navegación por los mensajes a la hora de notificar los mensajes. En este caso, podríamos necesitar un modelo de navegación por los mensajes que no afecte a los mismos, es decir, se navega por el conjunto de mensajes, pero no hay ningún efecto sobre ellos.

Un ejemplo de este tipo de necesidad podría ser la suscripción al servicio de notificación de cambios de un sistema de auditoría que reciba una notificación cada vez que el administrador de la CMDB accede a la misma, pero esos mismos mensajes deben ser notificados cada 5 horas y no se deben eliminar de la cola de mensajes del servicio de notificación de cambios.

En otro caso, podríamos tener una situación donde al navegar por los mensajes se van bloqueando de tal forma que cuando se produce una notificación el mensaje está bloqueado y no pueda ser utilizado para otro fin.

Otro caso puede ser que los mensajes a notificar a los diferentes suscriptores se vayan eliminando al navegar por ellos. Las tres tipologías son:

- Navegación Modo Normal, sin afectar a los mensajes (MNN).
- Navegación Modo Bloqueo de mensajes (NMBL).
- Navegación Modo Borrado de mensajes (NMBO).

Otro requisito dependiente de la navegación es la forma de desplazarse por la cola de los mensajes a ser notificados. Se puede establecer este requisito como Primer Mensaje, de forma que siempre vayamos al primer mensaje de la cola, o Siguiente Mensaje, para desplazarnos al siguiente o siguiente Transacción, de forma que nos desplacemos al mensaje correspondiente a la siguiente transacción.

Las opciones, por tanto, son:

- Navegación Primer Mensaje (NPM).

- Navegación Siguiente Mensaje (NSM).
- Navegación Siguiente Transacción (NST).

5.3.7. Gestión de Búsquedas

Otro grupo general de requisitos es el relacionado con las búsquedas de los mensajes de notificación. Por ejemplo, tenemos en nuestra organización una CMDB que contiene todos los ítems de configuración (CI) relacionados con el sistema de finanzas y el sistema de la Web Corporativa. Ambos sistemas actúan como suscriptores al servicio de notificación de cambios construido.

El sistema de finanzas está interesado en que se le notifiquen todas las compras de hardware mayores de 100.000 euros (cuando, por ejemplo se inserta un registro en la tabla de hardware de la CMDB y su valor es mayor de 100.000 euros) y la Web corporativa desea que se le notifiquen solamente aquellos cambios de software cuyo texto en los comentarios aparezca la palabra: "*Web*".

Las dos tipologías generales serán:

- Búsquedas sin criterio (BCC).
- Búsquedas con criterio (BSC).

5.3.8. Agrupación de mensajes

Se pueden establecer diferentes agrupaciones entre los mensajes a ser notificados. Las tipologías generales serán:

- Mensajes Sin agrupación (MSA).
- Mensajes Con agrupación (MCA).

Por ejemplo, nos puede interesar agrupar todos los mensajes de notificación de los accesos a la CMBD en un único mensaje. Cuando se notifica este mensaje, por ejemplo a un sistema de Auditoria, le llega un único mensaje con un formato estándar, independientemente de los accesos realizados.

5.3.9. Mecanismos de Control

Como último grupo de requisitos incluimos las diferentes tipologías de mecanismos de control del servicio de notificación de cambios. Tenemos, por un lado, los mecanismos que nos permiten transmitir mensajes en diferido y con persistencia, como la gestión avanzada de colas y, por otro lado, los mecanismos que transmiten los mensajes de forma inmediata, sin persistencia. Las tipologías generales son:

- Servicio con Persistencia y en Diferido (SPD).
- Servicio Sin Persistencia (SSP).

En la mayor parte de los casos tendremos la primera tipología, pues nos permite un mayor control sobre los mensajes e incluir un mayor número de requisitos, como en el caso de la Gestión de la Visibilidad.

5.4.Análisis de Procesos relacionados con el Dominio

Para realizar un análisis completo del dominio es imprescindible estudiar los principales procesos organizativos que están relacionados con la CMDB. En primer lugar, destacamos el proceso de **gestión de incidentes**. La Gestión de Incidentes tiene como objetivo resolver cualquier incidente que cause una interrupción en el servicio de la manera más rápida y eficaz posible.

La base de datos de gestión de la configuración (CMDB) ofrece una fuente de información muy valiosa para el manejo de incidentes [Gupta et al. 2008].

Los operadores responsables de la gestión de incidentes pueden rápidamente acceder a la situación de cada ítem de configuración (CI), determinar el impacto del incidente mediante la revisión de las relaciones entre los CIs y las aplicaciones del negocio e identificar los CIs relacionados con la restauración del servicio. Un servicio de notificación de cambios de los CIs en la CMDB tendrá será de gran utilidad para este proceso.

Por otro lado, tenemos el proceso de **gestión de problemas.** La Gestión de Incidentes no debe confundirse con la Gestión de Problemas, que se preocupa de encontrar y analizar las causas subyacentes a un determinado incidente, mientras que la gestión de incidentes su objetivo es exclusivamente restaurar el servicio [InfoWorld Oct. 2006].

La CMDB ofrece una fuente de datos para la gestión proactiva de problemas, acelerando y simplificando el análisis de las causas y la resolución de problemas. La CMDB puede proporcionar el estado inmediato de los CI afectados por un problema. La CMDB puede vincular los incidentes a los problemas y ayudar a visualizar el problema y sus CIs relacionados, con todas sus dependencias. También, la CMDB puede mostrar el historial de cambios que puede haber causado el problema. En este contexto, un servicio de notificación de cambios tendrá un gran impacto.

El siguiente proceso relevante en nuestro análisis es el proceso de **gestión de la configuración y gestión de activos**. La CMDB es la parte fundamental de estos dos procesos, permitiendo la identificación coherente, precisa y controlada de todos los ítems de configuración, que constituyen la base de estos procesos [Sharifi et al. 2008].

Otro proceso con un gran impacto en nuestro dominio es la **gestión de versiones**. La Gestión de Versiones es la encargada de la implementación y control de calidad de todo el software y hardware instalado en el entorno de producción.

La Gestión de Versiones debe colaborar estrechamente con la Gestión de Cambios y de Configuraciones para asegurar que toda la información relativa a las nuevas versiones se integra adecuadamente en la CMDB, de forma que ésta se halle correctamente actualizada y ofrezca una imagen real de la configuración de la infraestructura TI [Osiatis GV 2010]. Un servicio de notificación de cambios se puede considerar de gran importancia en este proceso.

El proceso de **Servicedesk** es otro proceso que está relacionado con el dominio bajo estudio. La CMDB puede permitir una mejora significativa en las funciones de la oficina de atención a usuarios (Servicedesk), al proporcionar información detallada sobre los ítems de configuración relacionados con las solicitudes de servicio.

La información sobre el estado de los CI, la configuración actual de los sistemas y su configuración base, las dependencias y todos sus cambios pueden ayudar a los gerentes de Servicedesk a satisfacer las solicitudes de servicio. La CMDB también puede proporcionar los datos a esta oficina, responsable de notificar a los usuarios las interrupciones y el estado de la resolución de problemas [Stackpole and Hanrion 2007]. En este escenario, el servicio de notificación de cambios también puede jugar un papel relevante.

Otro proceso capital dentro de las buenas prácticas recomendadas por ITIL es la Gestión de los niveles de servicio.

La CMDB puede dar soporte a la **gestión de los niveles de servicio**, proporcionando información detallada sobre los CI y sus relaciones con la infraestructura subyacente, vinculada a los servicios de TI. Proporcionar estos datos puede ser crucial para los acuerdos a nivel de servicio (SLA). La CMDB puede hacer referencias dinámicas a los componentes del SLA.

Cualquier cambio en un SLA o en un CI que tenga un impacto en el servicio puede llegar a ser clave en este proceso. Probablemente, y siguiendo algunas metodologías que así lo proponen como PRINCE2, PMP o PSP/IQ [PRINCE2 Tolerance 2009] [PMP 2003] [Baskarada 2009], los SLA estarán "*gestionados por tolerancias*". Mediante dispositivos que permitan disparar alarmas cuando esas tolerancias se sobrepasen, como pueda ser un mecanismo de notificación de cambios, estos servicios pueden ser controlados.

Como proceso adicional, podemos destacar el proceso de **gestión financiera**, un proceso común en todas las organizaciones de gestión TIC. La CMDB proporciona información que es crítica para una gestión financiera eficaz de las TIC.

La CMB contiene una lista completa de los ítems de configuración, de la que pueden reproducirse fácilmente los costes de operación, las licencias software, los contratos de mantenimiento, las fechas de renovación de las licencias, la obsolescencia de los activos, el cote de su reemplazo, etc. En secciones anteriores hemos expuesto una serie de ejemplos de interacción del servicio de notificación de cambios con este proceso.

Otro proceso importante, dentro de nuestro análisis, es la **gestión de la continuidad** de negocio, según las directrices marcadas por ITIL [Blokdijk and Menken 2008]. La CMDB puede almacenar información sobre los componentes de infraestructura de TI, su configuración y sus dependencias entre sí y los procesos clave de negocio.

Este proceso identifica la prioridad y el acuerdo de nivel de servicios TI tras una interrupción del servicio. Una de las actividades clave dentro del marco de la gestión de la continuidad del negocio son los procedimientos de emergencia en caso de desastre y la gestión de avisos. En este escenario, un servicio de notificación de cambios en la CMDB puede dar un soporte muy útil a este tipo de procedimientos.

Recorriendo las buenas prácticas ITIL, destacamos también los procesos de **gestión de la disponibilidad y gestión de la capacidad**. La gestión de la disponibilidad tiene que garantizar que los niveles de disponibilidad que se entregan en todos los servicios cumplen o superan todos los requisitos acordados, de manera eficiente en costes.

La gestión de la capacidad es un proceso técnico que convierte los requisitos del cliente en especificaciones de servicios TIC y predice los futuros; además predice, gestiona y controla los recursos, el rendimiento y la capacidad de los activos [Guía ITIL 2008].

La CMDB puede proporcionar un repositorio de información central que integra disponibilidad, fiabilidad, capacidad y mantenimiento de servicios con los componentes de la infraestructura de TI subyacente. La CMDB puede aportar importantes datos sobre el impacto en el negocio de estos procesos, muestra los componentes relacionados en una cadena de disponibilidad, se aportan datos de análisis de riesgos y ayuda a identificar las CI que son la causa raíz de los fallos de disponibilidad.

Para la gestión automatizada de la capacidad y de la monitorización de los servicios la CMDB es un elemento muy destacado. Los cambios sobre la CMDB y su gestión pueden ser muy importantes para estos procesos.

Otros procesos con una relación estrecha con la CMDB son la **gestión de proyectos TIC y el proceso de gestión del cambio**. La CMDB puede proporcionar un mecanismo para identificar, planificar, dar un seguimiento y actualizar los CI dentro del marco de un proyecto o de un cambio.

También podemos identificar otros procesos que pueden interactuar con una CMDB como la **gestión de la calidad** [Sharifi et al. 2008] y la **gestión de contratos**; los OLAs (Operational Level Agreements) o los UCs (Underpinning Contract), suelen formar parte de la CMDB [Bon ITSM 2008].

Por último, mencionaremos los **procesos de auditoría, gobernanza** y, en general, el cumplimiento de los procesos de control. La CMDB ofrece un repositorio de datos esencial relacionado con el control TIC, tanto para auditorías internas como externas. Los Objetivos de Control para la información y las tecnologías relacionadas, el Framework COBIT, recomiendan controles que efectivamente puedan aprovechar la información de una CMDB [Cobit 4.1 2007] [Sharifi et al. Cobit 2008].

Cada vez que se produce un cambio en una CMDB todos estos procesos mencionados podrían verse afectados y, en este escenario, el servicio de notificación de cambios se convierte en una necesidad. Posteriormente, en el capítulo correspondiente a la construcción de la CMDB, analizaremos todos los posibles suscriptores a este servicio.

5.5.Selección de los Requisitos de un Ejemplar

En la fase de análisis de requisitos del dominio bajo estudio se han descrito algunos de los casos de uso del servicio de notificación de cambios para una CMDB. En esta sección realizaremos una selección de los requisitos que formarán parte del ejemplar a desarrollar.

Según la directriz expuesta en [Heradio 2007] *convendrá que el ejemplar, para facilitar su posterior flexibilización, satisfaga un conjunto representativo de los requisitos de la familia de productos*. Si el modelo del dominio se describe con un diagrama de características, dicho conjunto estará formado, al menos, por todas las características obligatorias.

Como ya se mencionó anteriormente, se dispone de varios mecanismos para implementar el requisito de notificación de cambios.

Seleccionamos un mecanismo que nos permita persistencia y notificaciones en diferido para nuestro ejemplar, por ser el mecanismo que nos permite incorporar un mayor número de requisitos y es el más flexible y parametrizable. Opción: Servicio con Persistencia y en Diferido (SPD).

En segundo lugar, no tendremos agrupación de mensajes, pues cada mensaje se gestionará de forma individual. De esta forma, el desarrollo del ejemplar se simplifica y es más abierto. Opción: Mensajes Sin agrupación (MSA). El ejemplar no tendrá búsquedas especializadas. Opción: Búsquedas sin criterio (BCC).

La navegación por los mensajes a la hora de su notificación, para nuestro ejemplar, se realizará de forma que no se afecte a los mensajes. Esta será la opción, probablemente, más común en un servicio de notificación de cambios. Además, tendremos la opción de "siguiente mensaje", esto es, cada vez que se notifique un mensaje, la cola se desplazará al siguiente mensaje para ser notificado, que también será la opción más común. Opciones: Navegación Modo Normal, sin afectar a los mensajes (MNN) y Navegación Siguiente Mensaje (NSM).

El desencolado y encolado de los mensajes a notificar se realizará de forma inmediata; esto es, en cuanto se produzca un cambio en la CMDB, éste se encolará en la cola de mensajes a notificar y se desencolará también de forma inmediata, sin esperar a que concluya ninguna transacción. Opciones: Encolado y Desencolado con Visibilidad Inmediata (EVI y DVI).

En nuestro ejemplar los mensajes podrán tener prioridades, porque es la opción más completa. Opción: Mensajes Con Prioridad (MSP).

La granularidad de la solución ofrecida por nuestro ejemplar será de grano fino, por ser la opción que, entendemos, será más completa: los cambios gestionados por nuestro servicio se realizarán a nivel de registro de la CMDB. Opción: Granularidad de Grano Fino (GF).

Nuestro ejemplar será capaz de gestionar varios suscriptores. Como ya hemos analizado anteriormente, el conjunto de procesos que interactúan con una CMDB es muy variado y, generalmente, existirán múltiples suscriptores al servicio de notificación de cambios. Opción: Suscriptores Múltiples (SM).

En lo que se refiere a la gestión del tiempo, nuestro ejemplar podrá gestionar esperas con un cierto valor; esto es, podremos seleccionar un tiempo para que los mensajes se encolen en la lista a notificar y retenciones también con valor, esto es, retener los mensajes una vez notificados. Opciones: Esperas y Retenciones con valor (DECV y GTCR). Además, la gestión del tiempo de nuestro ejemplar podrá gestionar expiraciones y retrasos. Opción GTER.

Esto nos permitirá disponer de un ejemplar con bastante complejidad y muy flexible.

Con posterioridad serán realizados los correspondientes diagramas de características y se analizarán con detalle todos los requisitos en el estudio detallado del dominio completo.

Además de los requisitos, nuestro ejemplar tiene una complejidad bastante mayor, pues dentro del proceso de implementación de la flexibilización del ejemplar es necesario considerar otros factores como las funcionalidades a desarrollar, los elementos y las incompatibilidades entre requisitos.

Estas tareas se describen en la siguiente sección. Para nuestro ejemplar, debemos considerar que funcionalidades va tener, que elementos va a incluir y que incompatibilidades se generan dentro del propio ejemplar.

En resumen, la tabla que se muestra a continuación tiene los principales requisitos seleccionados para nuestro ejemplar.

REQUISITOS DEL EJEMPLAR DE BASE	
GRUPO DE REQUISITOS	**CODIGO**
Servicio con Persistencia y en Diferido	SPD
Mensajes Sin agrupación	MSA
Búsquedas sin criterio	BCC
Navegación Modo Normal, sin afectar a los mensajes	MNN
Navegación Siguiente Mensaje	NSM
Desencolado con Visibilidad Inmediata	DVI
Encolado con Visibilidad Inmediata	EVI
Mensajes Con Prioridad	MSP
Granularidad de Grano Fino	GF
Suscriptores Múltiples	SM
Esperas con valor.	DECV
Gestión General del Tiempo con Expiración y Retraso	GTER
Gestión del Tiempo con Retención (valor determinado)	GTCR

Tabla 5-1 Requisitos del Ejemplar.

5.6.Implementación de la Flexibilización del Ejemplar

En el proceso metodológico EODAM otra de las actividades relacionadas con la ingeniería de requisitos es la implementación de la flexibilización del ejemplar. Una vez seleccionados los requisitos de nuestro ejemplar, EODAM propone la realización de tres actividades:

- El análisis de Funcionalidades
- El análisis de Elementos
- El análisis de Incompatibilidades.

La figura siguiente muestra el resumen de esta fase:

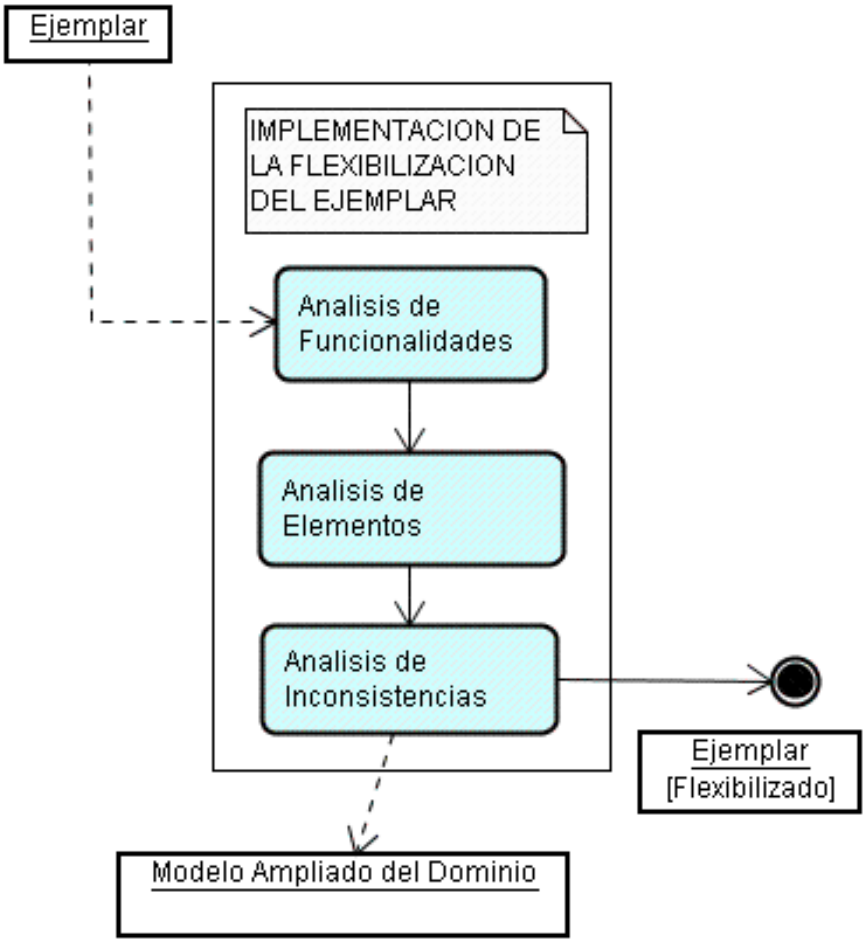


Figura 5-3 Implementación de la Flexibilización del Ejemplar.

5.6.1. Análisis de Funcionalidades

Esta actividad consiste en el análisis de las funcionalidades que es necesario flexibilizar en el ejemplar para la obtención de los productos.

Este análisis, principalmente, consiste en:

1. Obtener aquellas funcionalidades que son opcionales y obligatorias. Existirán algunas funcionalidades que siempre son requeridas y otras que serán opcionales.

2. Determinar las funcionalidades que deben ser flexibilizadas y cuáles no. Algunas de las funcionalidades serán comunes a todos los productos del dominio y otras deberán ser desarrolladas a medida y, por tanto, deberán ser flexibilizadas.

En primer lugar, destacaremos una funcionalidad obligatoria y que deberá de ser flexibilizada en el ejemplar: el desarrollo de los disparadores. Es necesario desarrollar diversos disparadores de tal forma que cuando se produzca un cambio en la CMDB, estos disparadores envíen los cambios producidos al observador de los cambios. Esta funcionalidad, a su vez, se divide en varias:

- **Creación de Disparadores (FCD).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten crear un disparador por cada uno de los cambios que queremos gestionar desde nuestro servicio. Por cada cambio a gestionar habrá, al menos, una función (disparador). En función de la tipología de cambios a considerar, el desarrollo de estas funcionalidades tendrá un alcance mayor o menor. Por ejemplo, podemos solamente considerar cambios a nuevas inserciones de datos por entidad (granularidad de grano medio), o podríamos considerar también cambios a cualquier modificación de datos o borrado y a nivel de registro, con lo que el número de funcionalidades podría ser muy elevado.

- **Habilitación de Disparadores (FHD).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten habilitar los disparadores creados para cada uno de los cambios que queremos gestionar desde nuestro servicio. Una vez creado un disparador será necesario habilitarlo para su uso. Por cada cambio a gestionar habrá, al menos, una función (habilitación de disparador).

- **Deshabilitación de Disparadores (FDD).** Esta funcionalidad, de tipo opcional, estará soportada por el conjunto de programas que nos permiten deshabilitar los disparadores creados para cada uno de los cambios que queremos gestionar desde nuestro servicio. Esta funcionalidad será opcional,

pues existirán algunos cambios que podamos habilitar/deshabilitar a conveniencia según las necesidades de nuestra organización. Por ejemplo, imaginemos que tenemos implementada un función disparador cada vez que se adquiere una nueva licencia del sistema operativo de red de nuestra organización y que ésta se refleja en la CMDB. Si, según nuestro procedimiento de compras, los meses de junio a septiembre no se adquieren este tipo de licencias, podríamos deshabilitar este disparador durante este tiempo.

- **Borrado de Disparadores (FBD).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten eliminar los disparadores creados para cada uno de los cambios que queremos gestionar desde nuestro servicio.

Hemos nombrado cada una de las funcionalidades con una sigla, comenzando por la letra F (Función) y las iniciales que identifican la misma, para que posteriormente sean enumeradas de forma breve.

En segundo lugar destacaremos una funcionalidad obligatoria y que deberá de ser flexibilizada en el ejemplar: la gestión de operaciones masivas. Nuestro servicio de notificación de cambios será capaz de gestionar operaciones masivas en la CMDB; por ejemplo, una importación de datos, la población inicial de la CMDB, una inserción masiva por una migración de datos, por una gran actualización de sistemas en producción, por un cambio general en la red, etc.

Cuando se realicen este tipo de operaciones masivas sobre la base de datos, será interesante disponer de una funcionalidad que nos permita notificar de dicho evento.

Esta funcionalidad se divide, a su vez, en varias:

- **Inicio de Operaciones Masivas (FIOM).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten enviar una notificación al observador para cada una de las operaciones masivas, cuando éstas se inician. Por ejemplo: *"Se está iniciando el proceso de migración de datos del nuevo sistema de contabilidad"*.

- **Fin de Operaciones Masivas (FFOM).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten enviar una notificación al observador para cada una de las operaciones masivas, cuando éstas concluyen. Por ejemplo: *"Ha finalizado el proceso de migración de datos del nuevo sistema de contabilidad"*. En este contexto si, por ejemplo, el sistema Helpdesk está suscrito a este tipo de notificaciones, cuando los operadores atiendan a los usuarios del sistema de contabilidad recibirán esta notificación y podrán informar a los usuarios de forma adecuada.

- **Borrado de Operaciones Masivas (FBOM).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten eliminar los procedimientos de operaciones masivas.

Una vez llegados a este punto insistimos en que **todas aquellas funcionalidades que son comunes a todos los productos**, que no son susceptibles de flexibilización y que deben ser desarrolladas para el ejemplar, **no serán abordadas en este análisis**, por tratarse de funciones que solamente se desarrollan para el ejemplar.

Estas funciones formarán parte del "*core*" de la familia de productos.

En tercer lugar, tenemos para nuestro ejemplar las funciones relacionadas con la gestión de las colas de notificación.

Para el caso de nuestro ejemplar, ya mencionamos que necesitábamos notificaciones con persistencia y en diferido y, por tanto, la gestión de colas es el mecanismo que da soporte a este tipo de sistemas. Estas funcionalidades nos permitirán configurar e inicializar las colas y todos los objetos relacionados con las mismas.

Esta funcionalidad se divide en:

- **Creación de Colas (FCC).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten crear las colas de notificación. En función de los diferentes requisitos, podremos necesitar crear un mayor o un menor número de colas. Por ejemplo, si todas las colas tienen los mismos requisitos de gestión del tiempo, visibilidad, etc. puede que solo sea necesario desarrollar una cola.

- **Configuración de la Cola (FCFC).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten configurar las colas de notificación. Esta configuración variará en función de los requisitos seleccionados.

- **Creación de los Mensajes de Notificación (FCMN).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten crear los tipos de mensajes de notificación.

- **Encolado (FE).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten encolar los mensajes de notificación. El observador que recibe los mensajes los enviará a las colas.

- **Desencolado (FD).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten desencolar los mensajes de las colas de notificación.

- **Eliminación de las Colas (FEC).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten eliminar las colas de notificación y todas las funcionalidades asociadas.

Por último, tendremos una serie de funcionalidades para asignar los suscriptores al servicio de notificación de cambios y asignar los correspondientes permisos.

Es necesario establecer diferentes permisos para la utilización del servicio de notificación de cambios y crear todos los suscriptores del mismo. El análisis detallado de los posibles suscriptores será realizando en el capítulo correspondiente a la construcción de la CMDB.

Esta funcionalidad se divide en:

- **Creación de Suscriptores (FCS).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten crear los suscriptores al mecanismo de notificación de cambios.

- **Asignación de Permisos (FAP).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten otorgar los permisos necesarios para el mecanismo de notificación de cambios.

- **Eliminación de Suscriptores (FES).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten eliminar los suscriptores.

- **Denegación de Permisos (FDP).** Esta funcionalidad, de tipo obligatorio, estará soportada por el conjunto de programas que nos permiten denegar los permisos otorgados para el mecanismo de notificación de cambios.

En el caso de que la solución elegida no fuera la notificación de cambios en diferido y con persistencia, variarían únicamente las funcionalidades de gestión de las colas.

En el capítulo correspondiente a la implementación del dominio trataremos todos estos aspectos.

Se han identificado todas las funcionalidades necesarias para configurar y desarrollar las notificaciones de los diferentes cambios producidos en la base de datos. Además, se han incluido también las correspondientes funcionalidades para desinstalar y eliminar las funcionalidades desarrolladas.

La tabla siguiente resume el conjunto de funcionalidades del ejemplar a ser flexibilizadas:

FUNCIONALIDADES DEL EJEMPLAR DE BASE	
FUNCIONALIDAD	**CODIGO**
Creación de Disparadores	FCD
Habilitación de Disparadores	FHD
Deshabilitación de Disparadores	FDD
Borrado de Disparadores	FBD
Inicio de Operaciones Masivas	FIOM
Fin de Operaciones Masivas	FFOM
Borrado de Operaciones Masivas (FBOM).	FBOM
Creación de Colas	FCC
Configuración de Colas	FCFC
Creación de los Mensajes de Notificación	FCMN
Encolado	FE
Desencolado	FD
Eliminación de las Colas	FEC
Creación de Suscriptores	FCS
Asignación de Permisos	FAP
Eliminación de Suscriptores	FES
Denegación de Permisos	FDP

Tabla 5-2 Funcionalidades del Ejemplar a Flexibilizar.

5.6.2. Análisis de Elementos

Dentro de la metodología EODAM, la segunda de las actividades a realizar durante la implementación de la flexibilización del ejemplar es el análisis de elementos. Existe una dependencia entre el desarrollo de los productos y la Base de Datos objetivo, en nuestro caso la CMDB.

Como ya se ha mencionado con anterioridad, en EODAM se considerarán Elementos a aquellas entidades que, teniendo una dependencia interna con la Base de Datos (como por ejemplo son: las tablas de la base de datos, las columnas, las claves, las prioridades de cada tabla a la hora de realizar las notificaciones, los usuarios de la base de datos, etc.), tienen algún tipo de incidencia en el desarrollo de los productos del dominio. Es necesario realizar un análisis de los elementos del dominio objetivo. Esta fase trata de realizar esta actividad.

En función del tipo de granularidad ofrecida por el servicio de notificación de cambios, tendremos un mayor o menor impacto en este análisis. En nuestro caso el ejemplar tiene una granularidad de tipo fino, lo que significa que los cambios se aplican a nivel de registro de la CMDB.

Por cada una de las tablas / entidades de la CMDB sobre la que nuestro servicio realice notificaciones de cambios, tendremos a continuación que hacer el siguiente análisis: ¿qué operaciones sobre estas tablas / entidades tenemos que notificar?

Para nuestro ejemplar consideraremos un número importante de operaciones de tal forma que su flexibilización nos permita adaptarnos a un alcance mayor de nuestro dominio. Incluiremos inserciones, borrados y actualizaciones de datos a nivel registro (granularidad de grano fino).

Consideraremos, en nuestro ejemplar, varios tipos de prioridades para estas operaciones. Además, todas estas operaciones tendrán retrasos en su notificación y expirarán.

En nuestro ejemplar, habrá varios suscriptores a nuestro servicio. Estos suscriptores serán también usuarios, con cierto rol, en la CMDB.

También, consideraremos otro tipo de operaciones, no sobre las entidades y los registros, sino sobre toda la base de datos: conexiones y desconexiones a la CMDB y operaciones masivas. Cuando tengamos ya construida la CMDB seleccionaremos algunas entidades como punto focal de nuestro servicio de notificación de cambios. No incluiremos todas las entidades.

5.6.3. Análisis de Inconsistencias

La última fase de la implementación de la flexibilización del ejemplar es el análisis de inconsistencias. Para ello, en primer lugar, es necesario estudiar las dependencias entre los requisitos del dominio.

Este estudio preliminar de dependencias nos dará como primer resultado la detección de diversas combinaciones no permitidas, denominadas "*Inconsistencias*" o "*Incompatibilidades*" entre requisitos.

Posteriormente, es necesario realizar un pequeño análisis de la implicación de esas inconsistencias a la hora de flexibilizar los ejemplares del dominio.

En nuestro caso, este análisis nos muestra que no es necesario realizar ninguna flexibilización de los ejemplares específica para tratar esas incompatibilidades, ya que únicamente será necesario comprobar dichas inconsistencias en nuestro DSL de tal forma que, si son detectadas, no se realicen las transformaciones correspondientes.

Como se detallará posteriormente en el capítulo dedicado a la implementación del servicio de notificación de cambios, durante la fase de obtención de los productos serán desarrollados tanto el DSL como los generadores de los productos.

En nuestro Framework de desarrollo de los generadores implementaremos un módulo de análisis preliminar del DSL de tal forma que si se detectan las mencionadas inconsistencias, no se ejecutarán las transformaciones desde el ejemplar al resto de los productos. Realizar dichas transformaciones en caso de que hubiera inconsistencias en el DSL nos generaría productos erróneos, no existentes en el mundo real.

A modo de ejemplo resumiremos algunas de las posibles inconsistencias entre los requisitos ya mencionados con anterioridad.

La opción del mecanismo en tiempo real sin persistencia no será compatible con varios requisitos relacionados con la gestión del tiempo. En este caso, no habrá expiración ni retraso, ni esperas con valor. Tendremos una Gestión del Tiempo Sencilla. No es el caso seleccionado para nuestro ejemplar, pero habrá que considerarlo para el proceso de flexibilización. Además, en el caso de utilizar este mecanismo, no existirán las prioridades, pues todos los mensajes serán notificados en tiempo real con la misma prioridad.

Otra incompatibilidad está relacionada con la granularidad. Una granularidad de grano grueso no será compatible con el establecimiento de prioridades ni tiempos por entidades, ya que las notificaciones se realizan a nivel de operaciones sobre la CMDB no sobre entidades. Una granularidad de grano fino podrá establecer prioridades o tiempos a nivel de registro.

Otra serie de incompatibilidades están relacionadas con el requisito de navegación de los mensajes por las colas de notificación. Las búsquedas con criterio solo son posibles con la opción de navegación en modo normal, sin afectar a los mensajes. Lógicamente, si al desplazarnos por la cola se afecta a la notificación de los mensajes, no será posible realizar búsquedas.

En el próximo capítulo se modelarán estas inconsistencias en forma de *“constraints”* de nuestro modelo ampliado del dominio.

5.7.Modelo del Dominio y Análisis de Rentabilidad

Una vez puestos de relieve algunos de los requisitos mas importantes, y según establece la metodología EODAM, se proceder a realizar un modelo ampliado del dominio. Este modelo será realizando en el próximo capítulo e incluirá los requisitos, las funcionalidades, las incompatibilidades y todos los componentes del dominio.

Para decidir si vale la pena flexibilizar el ejemplar o, por el contrario, es más conveniente construir cada producto por separado, EODAM establece la realización de una estimación entre los costes de desarrollar y mantener cada producto por separado y la flexibilización del ejemplar y los productos obtenidos por ésta.

En nuestro caso el objetivo es automatizar la generación de cualquier producto del dominio que cubra todos los requisitos o cualquier combinación de éstos.

El número de productos posibles es muy elevado, ya que no solamente hay que tener en consideración todas las combinaciones posibles de requisitos, sino que también sería necesario establecer la dependencia con la CMDB objetivo, las funcionalidades y las compatibilidades.

Por ejemplo, en el caso de que fuera necesario desarrollar un producto para cubrir unos requisitos determinados, como por ejemplo de granularidad fina (con un control de las notificaciones de los cambios a nivel de registro), con múltiples suscriptores (el número posible de suscriptores puede ser muy elevado) y con diferentes prioridades, debería de desarrollarse un producto a medida para dar respuesta a esta problemática planteada.

En el caso de que fuera necesario un problema similar, pero donde el requisito de la prioridad únicamente afectara a ciertas entidades, el producto debería ser desarrollado a medida, de la misma forma. En nuestro caso el objetivo es que la transformación del ejemplar nos permita obtener cualquier otro producto del dominio, y, por lo tanto, la rentabilidad de la solución parece garantizada.

No obstante**, se abordará un análisis detallado de la rentabilidad obtenida por el trabajo como finalización del proceso en el décimo capítulo**. **En este capítulo se presentará un modelo económico específico para la metodología EODAM**. **Este modelo podrá extenderse a otros problemas de desarrollo de familias de productos software.**

6. Propuesta de Documentación para la Variabilidad del Dominio

"En 2031,
los abogados serán
componentes habituales
de la mayoría de los equipos de desarrollo"
Grady Booch.

En el capítulo anterior fueron expuestas las fases relacionadas con el análisis del dominio bajo estudio. Una vez determinado este análisis, se procede a modelar nuestro dominio y para ello es necesario documentar la variabilidad del mismo.

En el capítulo dos, *el estado del arte,* fueron analizadas diversas propuestas y líneas de investigación relacionadas con la gestión de la variabilidad en líneas de productos software. Estas iniciativas presentan algunas carencias que serán expuestas en este capítulo. Para superar estas carencias se propone un nuevo lenguaje para la documentación de la variabilidad: NFTE (*Neutral Tree Feature Easy*). Las primeras secciones de este capítulo presentan este lenguaje y en las secciones posteriores se detallan los diferentes modelos del dominio y las métricas principales del mismo.

6.1.Neutral Tree Feature (NTF)

Como ya hemos mencionado en el capítulo dos, **VFD+ se considera por varios autores como el lenguaje más versátil para gestionar la variabilidad. No obstante, VFD+ tiene algunas limitaciones que son importantes destacar.**

En primer lugar, VFD+ introduce una complejidad innecesaria en la gestión automatizada de los diagramas, tal y como reconocen los propios autores [Schobbens et al. 2007], mencionando que el tratamiento en forma de árboles simplificaría la gestión.

En segundo lugar, un lenguaje en forma de árbol es expresivamente completo si incluye restricciones o constraints en forma de lógica proposicional textual. Pero en el caso de VFD+, las restricciones textuales se limitan a los nodos primitivos.

Para superar estas limitaciones, se propone el lenguaje Neutral Tree Feature (NFT). Tal y como se demuestra en [David Fdez. et al. 2009], este lenguaje, NFT, tiene la misma expresividad, concisión y capacidad de integración que VFD+; además, NFT supera este tipo de limitaciones. En [David Fdez. et al. 2009] se presenta la sintaxis y semántica de NFT.

6.2.La variabilidad en EODAM

La gestión de la variabilidad es un punto clave en la metodología EODAM. Como ya se ha explicado con anterioridad, durante la fase de la flexibilización del ejemplar se obtienen las salidas principales del proceso: un modelo del dominio en forma de DSL y un modelo ampliado del dominio. Este modelo ampliado es consecuencia del proceso de implementación de la flexibilización del ejemplar.

En todos estos modelos es necesario especificar la variabilidad de nuestro ejemplar y del dominio. Además, durante la fase de obtención de los productos se congela una versión de este modelo completo del dominio y se desarrolla un DSL, que será la entrada para la generación de todos los productos de la SPL.

Será de vital importancia para EODAM disponer de un soporte para la gestión de la variabilidad: disponer de un lenguaje adecuado para la representación de los modelos y contar con las herramientas adecuadas para su manipulación.

6.3.Neutral Tree Feature Easy (NTFE)

Como ocurría con VFD+, NFT, pese a presentar una serie de mejoras sobre las anteriores notaciones, carece del soporte de herramientas y la dificultad de definir un diagrama NFT del mundo real es alta. Por ello, **en EODAM proponemos trabajar con una notación más sencilla: NFTE (Neutral Tree Feature Easy)**, cuya sintaxis definimos a continuación.

NFTE (Neutral Feature Tree Easy) es una notación que representa un árbol de características. La sintaxis de NFTE será representada por dos tipos de términos de expresión:

- Un conjunto de expresiones que permitirá establecer las dependencias y relaciones entre las características.

- Un conjunto de términos de restricción que permitirá definir las restricciones del árbol.

Cada característica de NFTE es un Nodo de un árbol de NFT y sus relaciones se representan por la palabra "*node*".

La representación de NFTE es:

node('nodo_ padre', [nodo_hijo1,nodo_hijo2,nodo_hijo3…nodo_hijoN], A, B)

Donde N es el número de nodos hijo y A, B son números naturales (el mínimo y máximo número de nodos hijos). Por defecto, A y B tienen el valor 1.

Los diferentes valores de A, B nos dan todas las opciones de relación entre las características.

A continuación mostramos las diferentes opciones y como son representadas en NFT frente a FODA. NFT utiliza el operador *card* para generalizar cualquier tipo de relación entre las características o nodos padre e hijos y utiliza elementos auxiliares para representar algunas relaciones tal y como se muestra en [David Fdez. et al. 2009]:

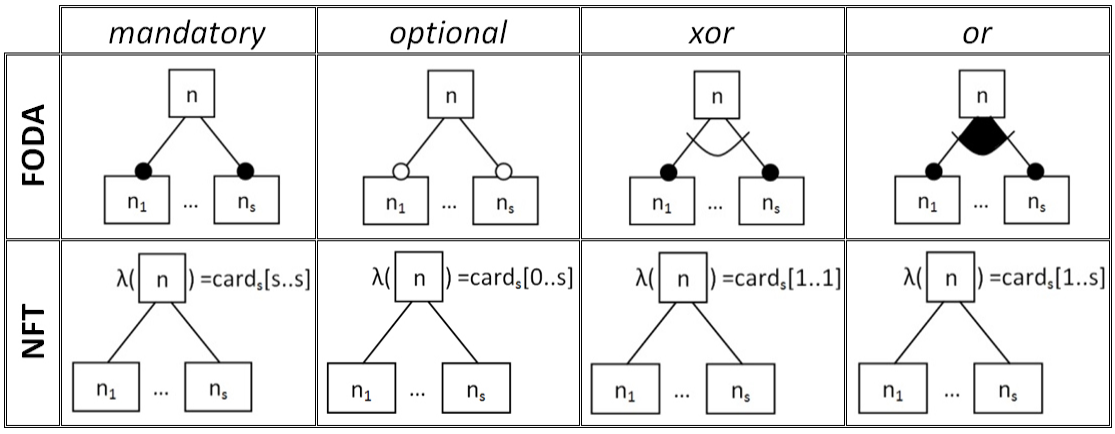


Figura 6-1 El operador card en NTF.

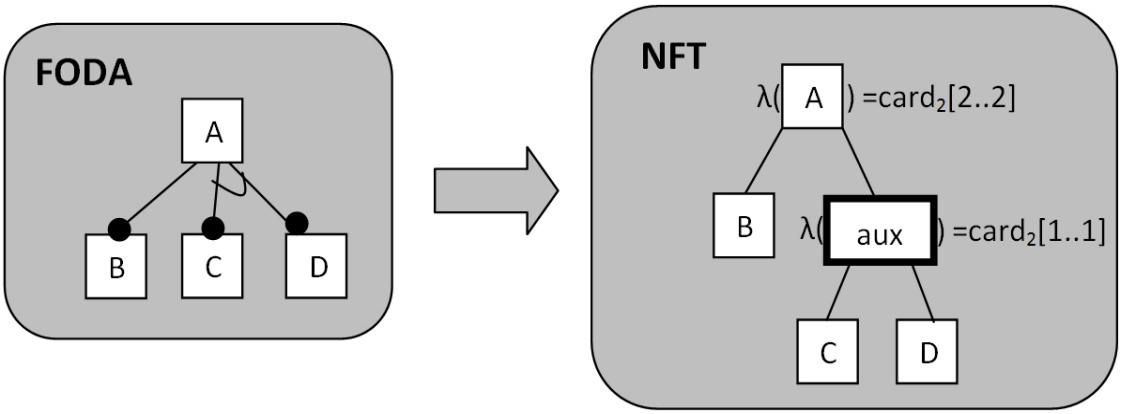


Figura 6-2 Representación en NFT de relaciones en FODA.

En el caso de NFTE, como ya hemos mencionado, mediante diferentes valores de A y B tenemos todas las opciones de relación entre las características, tal y como se muestra en la siguiente tabla, donde N es el número de nodos hijo:

Tipo de Relación	Representación en NTF	Representación en NFTE
Mandatory	$Card_s$ [s..s]	A = N, B = N
Optional	$Card_s$ [0..s]	A = 0, B = N
Or_s	$Card_s$ [1..s]	A = 1, B = N
Xor_s	$Card_s$ [1..1]	A = 1, B = 1

Tabla 6-1 Representación en NFTE de las relaciones.

Tenemos varios tipos de características en la representación NFTE:

- Padres.
- Hijas.
- Hojas.
- Raíz.

Los Padres constituyen las características padres de la representación en árbol NFT. Son el conjunto de características que, en su serialización (sintaxis concreta de NFTE), aparecen en primer lugar después de la palabra “node”.

Las Hijas constituyen las características hijas de la representación en árbol NFT. Son el conjunto de características que, en su serialización (sintaxis concreta de NFTE), están contenidas a continuación del nodo padre, entre corchetes y separadas por comas.

Las Hojas constituyen las características hojas de la representación en árbol NFT. Son el conjunto de características hijas de la representación NFTE que, en su serialización (sintaxis concreta), están contenidas, únicamente, entre corchetes y separadas por comas. Nunca aparecen como padres.

La Raíz constituye el nodo raíz de la representación en árbol NFT. Por definición, es la única característica que no está contenida entre corchetes en su serialización (sintaxis concreta de NFTE). No es hija de ningún nodo, no tiene padre.

Los comentarios en la notación NFTE se inician con el símbolo #.

Por último, las restricciones se representan de la siguiente manera:

constraint('boolean expression')

Por ejemplo,

constraint('nodo1=> nodo2')

Significa que si seleccionamos el nodo 1 obligatoriamente debemos seleccionar el nodo2 en nuestra configuración. Un producto del mundo real tendría obligatoriamente ambas características.

constraint('nodo1=> not nodo2)

Significaría que si seleccionamos el nodo 1, obligatoriamente, no debemos seleccionar el nodo2. Un producto del mundo real no tendría ambas características.

La expresión booleana incluye las operaciones NOT, AND y OR.

La representación global, por tanto, de NFTE es:

 # Comentarios…
 # Nodos y relaciones
node('nodo_ padre', [nodo_hijo1,nodo_hijo2,nodo_hijo3…nodo_hijoN], A, B)
……..
 # Restricciones
constraint('nodo1=> (boolean expression))
……

Veamos algunos ejemplos obtenidos de la bibliografía para mayor claridad sobre nuestra notación. En el siguiente ejemplo, obtenido de la figura 5.1:

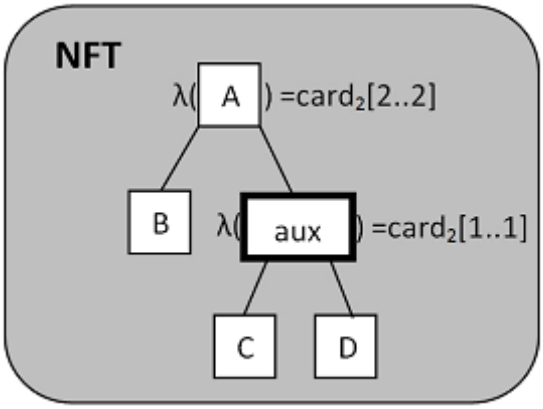


Figura 6-3 Ejemplo nº 1 de notación NFTE.

Su representación en formato NTFE sería:

node('A',['B','aux'],2,2)
node ('aux',['C','D'],1,1)

Veamos otro ejemplo con constraints. El siguiente ejemplo está obtenido del artículo *Inferring Information from Feature Diagrams to Product Line Economic Models* [David Fdez. et al. 2009], que utiliza la notación NTF para la representación del modelo de características:

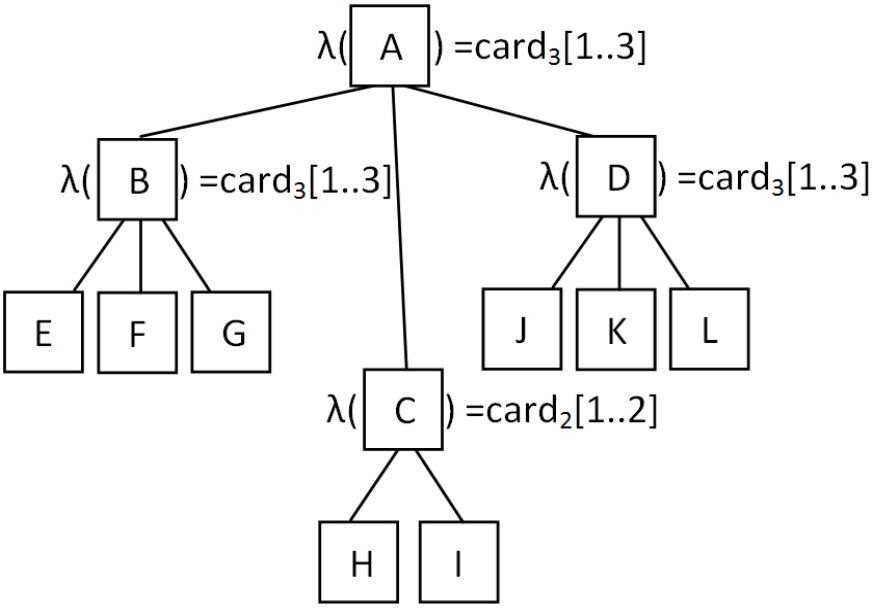


Figura 6-4 Ejemplo nº 2 de notación NFTE.

Su representación en formato NTFE sería:

```
node('A',['B','C','D'],1,3)
node('B',['E','F','G'],1,3)
node('C',['J','K','L'] ,1,3)
node('D',['H','I'],1,2)
constraint('E=>H')
constraint('G=>H')
constraint('J=>I')
```

Para dar soporte gráfico a NFTE, con el objetivo de facilitar la visualización de los diagramas, se han desarrollado una serie de herramientas de soporte que permiten obtener desde la notación NFTE una serie de salidas que constituyen las entradas de

una herramienta visual denominada GRAPHVIZ [Graph 2010], un proyecto de *Open Source* para la representación de diagramas.

Desde los ejemplos mencionados con anterioridad se obtiene la siguiente representación para el ejemplo nº 1:

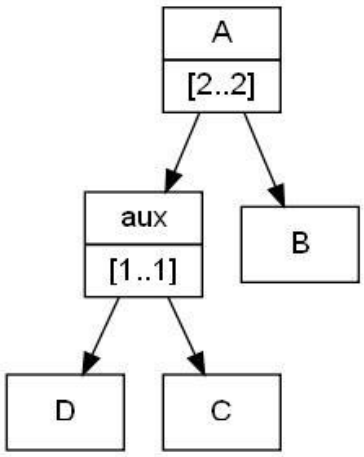


Figura 6-5 Ejemplo nº 1 en representación Graphviz.

El fichero de entrada a Graphviz es obtenido de forma automática desde un algoritmo:

```
digraph NFT {
        A [shape=record, label="{A|{[2..2]}}"];
        aux [shape=record, label="{aux|{[1..1]}}"];
        D [shape=box];
        C [shape=box];
        B [shape=box];
        A->B;
        A->aux;
        aux->C;
        aux->D;
}
```

Para el ejemplo nº 2 la representación de Graphviz que obtenemos desde nuestra notación NTFE es:

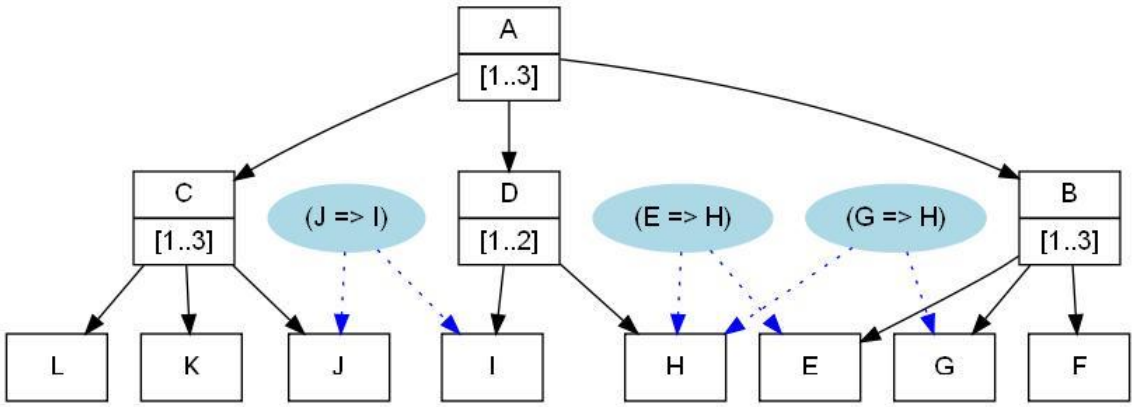


Figura 6-6 Ejemplo nº 2 en representación Graphviz.

El fichero de entrada a Graphviz es obtenido de forma automática desde un algoritmo, distribuido bajo licencia GNU Lesser General Public License (LGPL) [David y Heradio 2010].

```
digraph NFT {
        A [shape=record, label="{A|{[1..3]}}"];
        D [shape=record, label="{D|{[1..2]}}"];
        I [shape=box];
        H [shape=box];
        C [shape=record, label="{C|{[1..3]}}"];
        J [shape=box];
        B [shape=record, label="{B|{[1..3]}}"];
        G [shape=box];
        F [shape=box];
        E [shape=box];
        L [shape=box];
        K [shape=box];
        A->B;
        A->C;
        A->D;
        D->H;
        D->I;
        C->J;
        C->K;
        C->L;
        B->E;
        B->F;
        B->G;
```

```
constraint_1 [label="(E => H)", style=filled, color = lightblue];
constraint_1 -> E [style = dotted, color = blue];
constraint_1 -> H [style = dotted, color = blue];
constraint_2 [label="(G => H)", style=filled, color = lightblue];
constraint_2 -> G [style = dotted, color = blue];
constraint_2 -> H [style = dotted, color = blue];
constraint_3 [label="(J => I)", style=filled, color = lightblue];
constraint_3 -> J [style = dotted, color = blue];
constraint_3 -> I [style = dotted, color = blue];}
```

Como ya hemos mencionado, la gestión de la variabilidad requiere de diversas funcionalidades. Para dar respuesta a estas necesidades se han desarrollado un conjunto de herramientas que permiten resolver diferentes funcionalidades.

En primer lugar, un algoritmo obtiene el número de productos de nuestra SPL a partir de la representación de NFTE. Este algoritmo se distribuye bajo la licencia GNU Lesser General Public License (LGPL) y se encuentra disponible en: http://www.issi.uned.es/miembros/pagpersonales/ruben_heradio/spl.htm

Esta métrica, como ya hemos comentado, es de vital importancia. Además, este algoritmo presentado en [David Fdez. et al. 2009] presenta una gran escalabilidad.

En segundo lugar, y desde la notación NFTE, a través de un algoritmo complementario se obtienen el número de productos donde aparecen todas y cada una de las características. Para ello existe la implementación de un algoritmo que obtiene esta métrica.

En tercer lugar, y con el uso de la misma herramienta, se obtiene la comunalidad de todas y cada una de las características. La comunalidad de una característica "feature(i)" es calculada por la ecuación:

$$\boldsymbol{Comunalidad_{feature(i)} = \frac{PNP_{feature(i)}}{n}}$$

Ecuación 6-1 Comunalidad de una característica.

Donde n es el número de productos de la SPL y $PNP_{feature(i)}$ es el número de productos que contienen la característica "feature(i)".

Por último, haciendo uso de las mismas herramientas, se obtiene la homogeneidad de nuestra SPL. La homogeneidad de una línea de productos se obtiene por la ecuación:

$$Homogeneidad_{SPL} = 1 - \frac{NF_U}{NF_T}$$

Ecuación 6-2 Homogeneidad de una SPL.

Donde NF_U es el número de características únicas a un producto y NF_T es el número total de características diferentes en nuestra SPL.

Siguiendo con los ejemplos mencionados con anterioridad, y haciendo uso de las herramientas disponibles, obtenemos la siguiente información para el ejemplo nº 1:

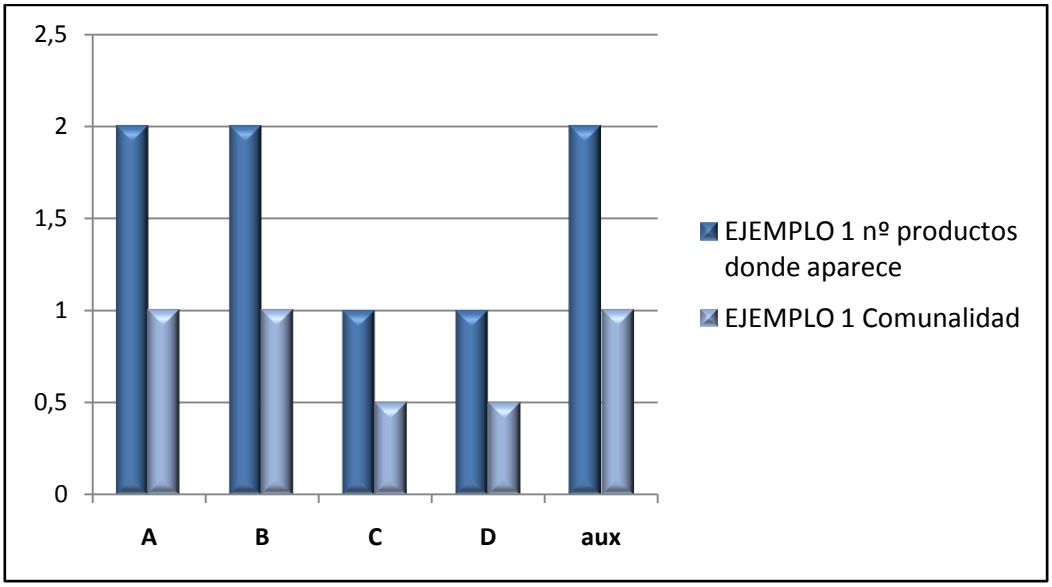


Figura 6-7 Métricas del Ejemplo 1.

Como datos globales:

Homogeneity *= 0.600000.*

The number of products of the SPL is 2

Para el ejemplo nº 2 obtenemos la siguiente información:

EJEMPLO 2		
Elemento	nº productos donde aparece	Comunalidad
A	119	1,00
B	96	0,80
C	100	0,40
D	112	0,94
E	48	0,40
F	60	0,50
G	48	0,40
H	96	0,80
I	80	0,67
K	60	0,50
J	40	0,33
L	60	0,50

Tabla 6-2 Métricas del Ejemplo 2.

Como datos globales:

Homogeneity *= 1.000000*

The number of products of the SPL is 119

Como complemento a las herramientas de soporte a NTFE se ha desarrollado un sistema que permite la generación automática de modelos tal y como se describe en [NFTE Random 2010] y, además, con objeto de ganar en escalabilidad con los diagramas visuales, se puede trabajar con *GNU MP Bignum Library*, disponible en http://gmplib.org/.

Como ya hemos analizado, con nuestra propuesta damos soporte a las siguientes funcionalidades:

I. **La validación del Modelo.** Las herramientas y propuestas dan soporte a esta operación.

II. **Detección de características muertas.** Las herramientas y propuestas dan soporte a esta operación.

III. **Optimización**. Esta funcionalidad es soportada por las herramientas, presentando una gran escalabilidad.

IV. **Cálculo del número de productos.** Esta funcionalidad es soportada por las herramientas, presentando una gran escalabilidad frente a otras propuestas.

V. **Porcentaje de uso de una característica.** Esta funcionalidad es soportada por las herramientas, presentando una gran escalabilidad.

VI. **Escalabilidad**. La escalabilidad de las herramientas de soporte a NFTE es muy alta. Además, con el complemento de herramientas como GNU MP Bignum Library, se amplía el alcance de esta funcionalidad.

VII. **Control de errores.** Las herramientas disponen de un control de errores.

VIII. **Control sobre los productos generados.** En nuestro caso, completaremos esta funcionalidad con un desarrollo ad hoc para nuestro dominio tal y como describiremos en el capítulo dedicado al desarrollo del servicio de notificación de cambios.

IX. **Generación automática de modelos.** Esta funcionalidad está cubierta.

X. **Portabilidad e interoperabilidad**. La portabilidad e interoperabilidad de las herramientas construidas es muy amplia. Desde la notación NFTE puede, fácilmente, transformarse a cualquier tipo de formato de entrada, por su sencillez.

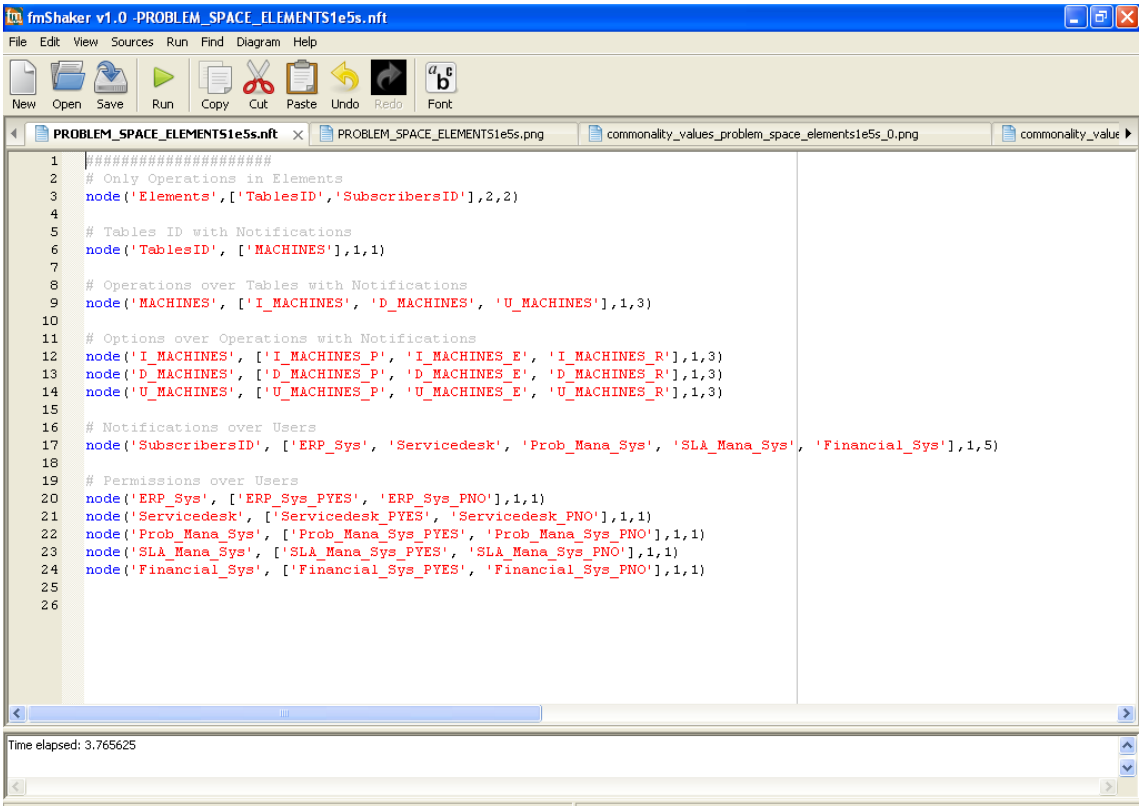
XI. **Facilidad de uso.** Describir un diagrama en NFTE es sumamente sencillo, como puede observarse en los ejemplos analizados, frente al diseño de diagramas visuales o la configuración de complejas herramientas. Los roles que participan en el espacio del problema, que pueden estar muy alejados del campo de la ingeniería del software, podrán definir con facilidad sus requisitos mediante la notación NFTE.

XII. **Otras funcionalidades adicionales.** Además, las herramientas de soporte a NFTE nos permiten obtener información muy importante de nuestro dominio, como la comunalidad de cada una de las características o la homogeneidad de nuestra línea de productos.

Destacamos también la **herramienta fmShaker [FMSHAKER 2010], un Entorno de Desarrollo Integrado (IDE)**, que nos ofrece un soporte para algunas de las funcionalidades que precisamos como el número total de productos, la homogeneidad de la SPL, el grado de reutilización (comunalidad) o la obtención de los diagramas gráficos desde la notación NFTE.

La herramienta fmShaker, que fue testeada en el marco de esta Tesis, admite como fichero de entrada un diagrama de características en formato NFTE y proporciona un editor para estos ficheros. Además, la herramienta está integrada con Graphviz y obtiene de forma automática ficheros en formato PNG (Portable Network Graphic) desde la notación NFTE.

A continuación se muestran algunos interfaces de usuario de la herramienta.



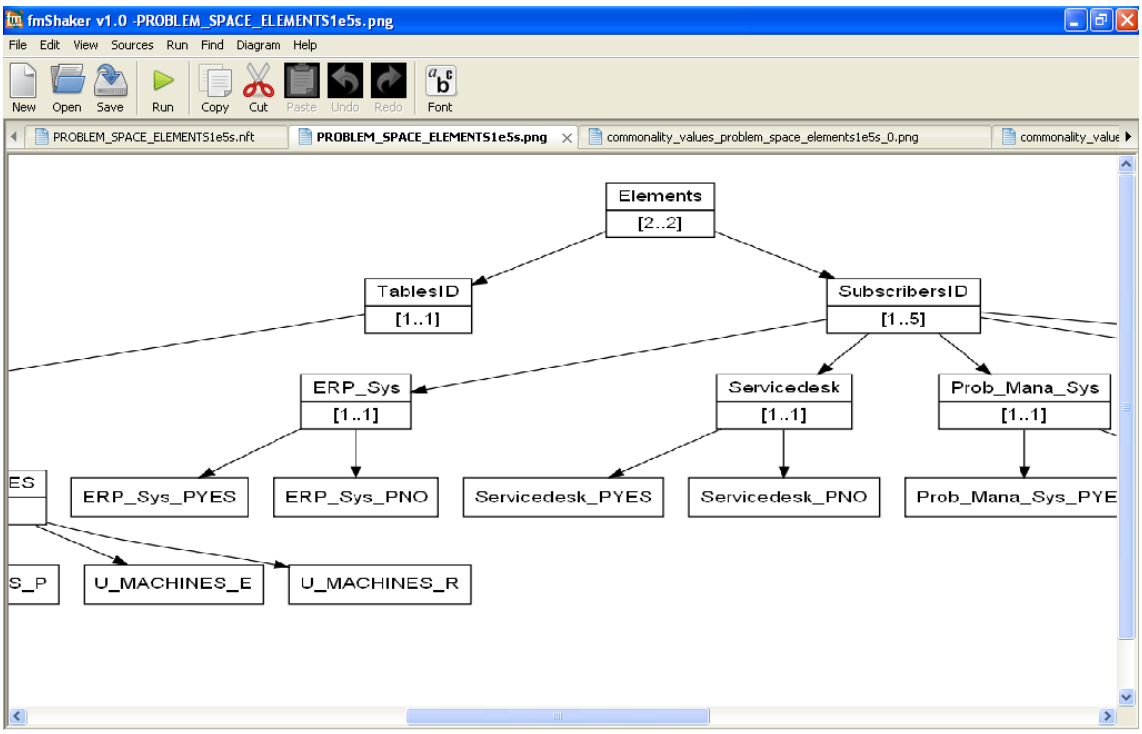


Figura 6-8 Interfaces de la herramienta fmShaker.

6.4.Diagramas del Dominio

Una vez presentada nuestra propuesta de documentación para la gestión de la variabilidad y en base al análisis del dominio realizado en el capítulo quinto, en esta sección expondremos algunos de los diagramas del dominio bajo estudio: la implementación de un servicio de notificación de cambios en una CMDB.

No es el objeto de esta sección exponer todos y cada uno de los diagramas del dominio, solamente algunos representativos. Además, y como ya hemos mencionado, en base a la metodología EODAM nos quedan pendientes algunas partes del análisis del dominio que nos permitirán matizar los diagramas con mayor detalle.

6.4.1. Diagrama de Requisitos

El primero de los diagramas a considerar es el de los requisitos. A continuación mostramos la notación NFTE de este diagrama, sin incluir las restricciones y únicamente considerando el mecanismo que nos permiten transmitir mensajes en diferido y con persistencia, como la gestión avanzada de colas (Servicio con Persistencia y en Diferido, SPD), ya que es el mecanismo que nos permite considerar el mayor número de requisitos.

```
######################
# NFTE DE REQUISITOS #
######################
# GT-Gestion del Tiempo. GGT-Gestion General del Tiempo
# GE-Gestion de Esperaso
# GS-Gestion de Suscriptores
# G-Granularidad de la Solucion
# GP-Gestion de Prioridades
# TV-Tipos de Visibilidad
# TN-Tipos de Navegacion
```

```
# GB-Gestion de Busquedas
# AM-Agrupacion de Mensajes
node('SPD',['GT','GGT','GE','GS','G','GP','TV','TN','GB','AM'],10,10)
node('GT',['GTSR','GTCR','GTCRI'],1,1)
node('GGT',['GTS','GTE','GTR','GTER'],1,1)
node('GE',['DSE','DECV','DEI'],1,1)
node('GS',['SU','SM'],1,1)
node('G',['GG','GM','GF'],1,1)
node('GP',['MSP','MCP'],1,1)
# VE-Visibilidad de Encolado
# VD-Visibilidad de Desencolado
node('TV',['VE','VD'],2,2)
node('VE',['EVI','EVT'],1,1)
node('VD',['DVI','DVT'],1,1)
# MN-Modo de Navegacion
# DM-Desplazamiento de los Mensajes
node('TN',['MN','DM'],2,2)
node('MN',['MNN','NMBL','NMBO'],1,1)
node('DM',['NPM','NSM','NST'],1,1)
node('GB',['BCC','BSC'],1,1)
node('AM',['MSA','MCA'],1,1)
```

Desde esta definición del modelo de requisitos, y haciendo uso de las herramientas de transformación gráfica obtenemos la siguiente **representación visual de este diagrama:**

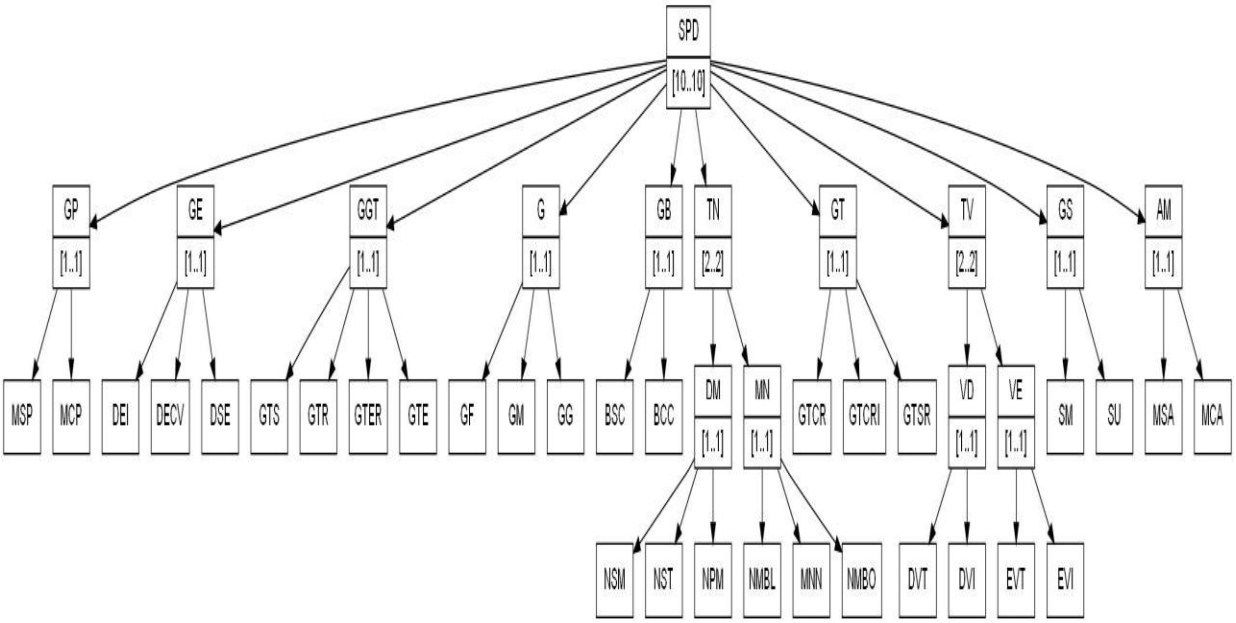


Figura 6-9 Modelo de Requisitos sin restricciones.

Si, además, consideramos algunas de las restricciones analizadas, incluiremos la siguiente información en el diagrama:

```
#Restricciones
constraint('BCC => MSA')
constraint('BCC => MNN')
constraint('MCA => MNN')
constraint('MCA => MNN')
```

Haciendo uso de las herramientas de transformación gráfica obtenemos la siguiente representación visual de este diagrama:

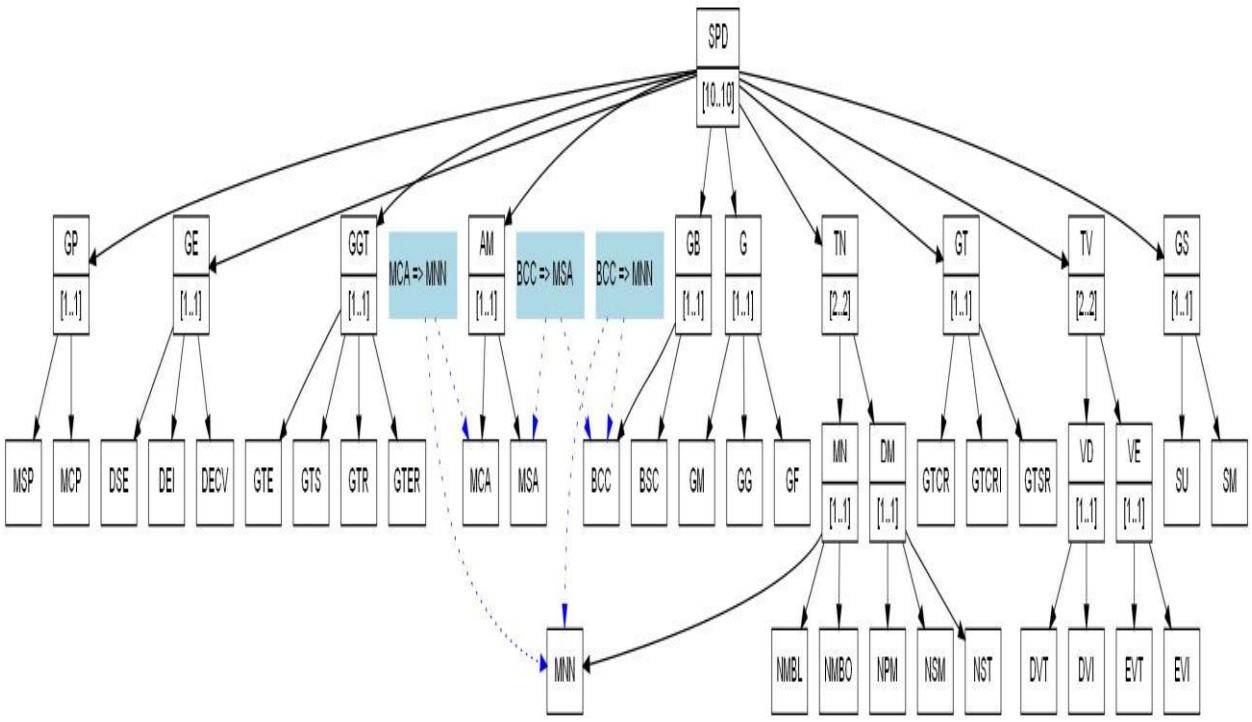


Figura 6-10 Modelo de Requisitos con restricciones.

Este modelo no ofrece completamente el análisis de requisitos realizado en el capítulo quinto porque, por ejemplo, no se incluye el mecanismo de notificaciones en tiempo real, pero es un modelo con un gran alcance que nos sirve de base para continuar con nuestro estudio.

6.4.2. Diagrama de Funcionalidades

El segundo de los diagramas a considerar es el de las funcionalidades. A continuación mostramos la notación NFTE de este diagrama, incluyendo las restricciones pero sin considerar las dependencias entre las funcionalidades y los elementos y el análisis de suscriptores al servicio de notificación de cambios. No incluimos esta información porque ni los elementos ni los suscriptores han sido, aún, analizados.

Además, tampoco consideramos la inclusión de operaciones generales sobre la CMDB como conexiones o desconexiones.

```
####################
# NFTE DE FUNCIONES #
####################
# FG-Funcionalidades Generales
# ID-Inserciones, AD-Actualizaciones, BD-Borrados
# OM-Funcionalidades de Operaciones Masivas
node('FG',['FCD','FHD','FDD','OM','FCX'],5,5)
node('FCD',['FCD_ID','FCD_AD','FCD_BD'],1,3)
node('FHD',['FHD_ID','FHD_AD','FHD_BD'],1,3)
node('FDD',['FDD_ID','FDD_AD','FDD_BD'],1,3)
node('OM',['FIOM','FFOM','FBOM'],3,3)
# Restricciones
constraint('FCD_ID => FHD_ID and FDD_ID')
constraint('FCD_AD => FHD_AD and FDD_AD')
constraint('FCD_BD => FHD_BD and FDD_BD')
```

Haciendo uso de las herramientas de transformación gráfica obtenemos la siguiente representación visual de este diagrama:

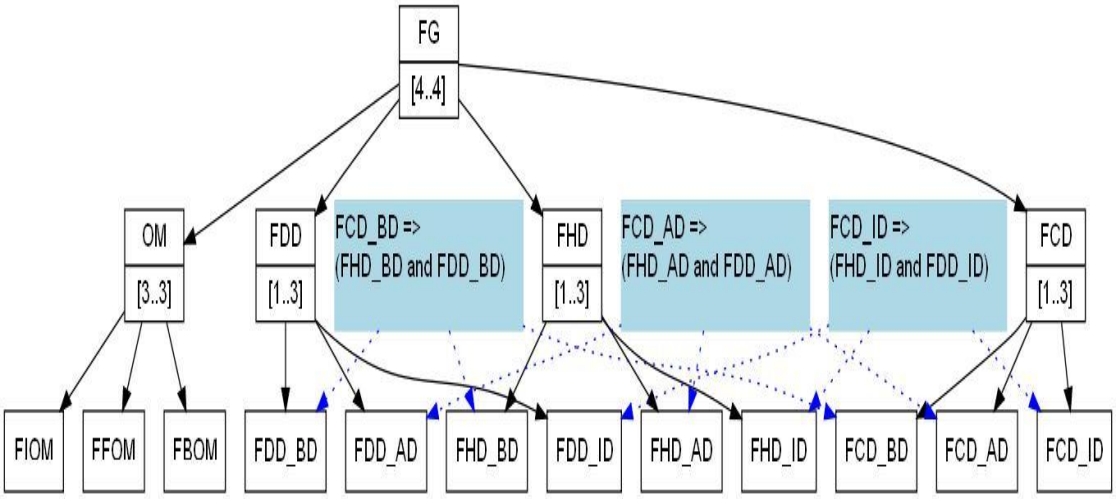


Figura 6-11 Modelo de Funcionalidades.

Este modelo no ofrece completamente el análisis de funcionalidades realizado en el capítulo quinto, pero es un modelo con un gran alcance que nos sirve de base para continuar nuestro estudio.

6.4.3. Modelos del Dominio

El tercer diagrama a considerar sería el de elementos. Recordemos que los elementos son aquellas entidades que, teniendo una dependencia con la estructura de la CMDB, tienen una incidencia en el desarrollo de los productos del dominio.

En el capítulo anterior, hicimos un análisis preliminar de los elementos de nuestro dominio y su dependencia con el ejemplar seleccionado. Una vez construida la CMDB podremos realizar el correspondiente modelo de elementos y escribirlo en su notación NFTE.

El modelo completo del dominio incluirá, además de las funcionalidades, requisitos y restricciones, los elementos del dominio.

En nuestro caso, como no partimos de una CMDB ya construida, debemos de realizar antes esta fase. En caso de que ya tuviéramos una CMDB, el modelo completo sería la unión de todos esos modelos. **Nuestro modelo ampliado del dominio será únicamente la unión de los modelos anteriores** que, en notación NFTE, nos quedaría:

```
##############################
# MODELO AMPLIADO DEL DOMINIO #
##############################
node('MAD',['SPD','FG'],2,2)
node('SPD',['GT','GGT','GE','GS','G','GP','TV','TN','GB','AM'],10,10)
node('GT',['GTSR','GTCR','GTCRI'],1,1)
node('GGT',['GTS','GTE','GTR','GTER'],1,1)
node('GE',['DSE','DECV','DEI'],1,1)
node('GS',['SU','SM'],1,1)
node('G',['GG','GM','GF'],1,1)
node('GP',['MSP','MCP'],1,1)
node('TV',['VE','VD'],2,2)
node('VE',['EVI','EVT'],1,1)
node('VD',['DVI','DVT'],1,1)
node('TN',['MN','DM'],2,2)
node('MN',['MNN','NMBL','NMBO'],1,1)
node('DM',['NPM','NSM','NST'],1,1)
node('GB',['BCC','BSC'],1,1)
node('AM',['MSA','MCA'],1,1)
node('FG',['FCD','FHD','FDD','OM','FCX'],5,5)
node('FCD',['FCD_ID','FCD_AD','FCD_BD'],1,3)
node('FHD',['FHD_ID','FHD_AD','FHD_BD'],1,3)
node('FDD',['FDD_ID','FDD_AD','FDD_BD'],1,3)
node('OM',['FIOM','FFOM','FBOM'],3,3)
constraint('BCC => MSA')
constraint('BCC => MNN')
constraint('MCA => MNN')
constraint('MCA => MNN')
constraint('FCD_ID => FHD_ID and FDD_ID')
constraint('FCD_AD => FHD_AD and FDD_AD')
constraint('FCD_BD => FHD_BD and FDD_BD')
```

Una vez llegados a este punto, conviene aclarar que el **modelo completo del dominio** se obtiene, tal y como se propone en EODAM, como primera actividad dentro de la fase de obtención de los productos mientras que el **modelo ampliado del dominio** se obtiene como última fase de la implementación de la flexibilización del ejemplar del dominio.

Evidentemente, el modelo completo del dominio incluye el modelo ampliado, pero mientras que el modelo ampliado incluye funcionalidades, requisitos y restricciones, el modelo completo del dominio incluye la integración de todos ellos y los elementos. Además, incluirá las dependencias entre ellos.

Imaginemos que queremos seleccionar la funcionalidad de inserciones pero solamente sobre ciertas entidades de nuestra CMDB. Por ejemplo, cada vez que se inserta un dato en una tabla que contiene licencias Software, que nuestro sistema de gestión financiera reciba una notificación; además, que sobre la tabla que almacena las versiones de los sistemas operativos queremos seleccionar la funcionalidad de actualizaciones, es decir, cada vez que se actualice esa tabla, se requiere una notificación, pero esta vez al sistema de gestión de la configuración.

Esta dependencia entre elementos, funcionalidades y requisitos hace que el modelo completo del dominio, del cual se desarrollará una versión en el próximo capítulo, sea bastante más complejo que el modelo ampliado.

6.5.Métricas iniciales del Dominio

Una vez completado el Modelo Ampliado del Dominio obtendremos las primeras métricas de nuestro dominio. No es el objeto de esta sección exponer todas y cada una de las métricas de nuestro dominio, sino solamente algunas de las más representativas.

Estas métricas serán las iniciales del dominio, porque las métricas completas las obtendremos del modelo completo.

En lo que a requisitos se refiere, haciendo uso de las herramientas de soporte a NFTE, si utilizamos el modelo de requisitos del dominio (sin restricciones) como entrada, obtenemos que el número de productos distintos son 62.208 productos.

Pero este modelo no nos da todos los posibles productos diferentes de nuestro dominio, porque sin considerar las restricciones obtendríamos productos no existentes en el mundo real. Podríamos desarrollar un producto no real.

Considerando las restricciones, en realidad, el número de productos de nuestro dominio sería de 25.920.

El número de productos diferentes para cada uno de los requisitos viene reflejado por la siguiente gráfica:

Figura 6-12 Nº de Productos por Requisito.

Las comunalidades de cada uno de los requisitos vienen reflejadas por la siguiente gráfica:

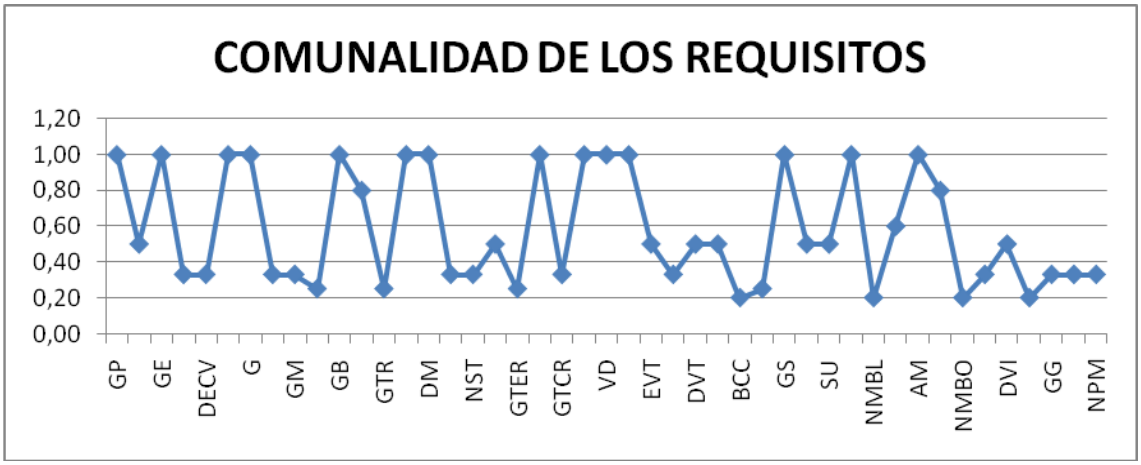


Figura 6-13 Requisitos y Comunalidad.

Desde el punto de vista del espacio del problema, los usuarios percibirían estos datos como muy interesantes a la hora de tomar decisiones de tipo funcional.

Desde el punto de vista del espacio de la solución, si realizamos un estudio de las posibles funcionalidades a desarrollar, obtenemos la siguiente información.

El número de opciones distintas de funcionalidades es de 61. Este dato tendrá un impacto directo en los desarrolladores del servicio de notificación de cambios, que tomarán este dato como muy interesante. Solamente existen 61 combinaciones diferentes de funcionalidades a desarrollar **(*"productos de desarrollo"***), frente al número de posibles ***"productos funcionales"***, que es muy elevado. Esto tiene sentido porque un grupo reducido de funcionalidades abarca multitud de productos.

A la hora de tomar decisiones en lo que al desarrollo se refiere, es importante considerar que, por ejemplo, funcionalidades como la creación de los disparadores (FCD_ID, FCD_AD y FCD_BD) incluyen a 25 de los posibles productos de desarrollo mientras que funcionalidades como la habilitación o deshabilitación de disparadores (FDD_ID, FDD_AD, FDD_BD, FHD_ID, FHD_AD y FHD_BD) incluyen a 43 de los posibles productos de desarrollo.

Como ya hemos mencionado con anterioridad, las métricas completas las obtendremos a partir del modelo completo del dominio, pero con el modelo ampliado tenemos ya una información muy importante para tomar decisiones.

7. Construcción de una CMDB

"No hay lenguaje de programación,
no importa su estructura,
que impida que los programadores
hagan malos programas."
— Larry Flon

Para llevar a cabo el desarrollo de nuestro dominio, el servicio de comunicación de los cambios desde la CMDB al resto de procesos de negocio relacionados con la gestión de las TIC, nos podemos encontrar en tres situaciones:

1. Nuestra organización dispone ya de una CMDB operativa que no va a ser modificada.
2. Nuestra organización dispone de una CMDB que debe ser modificada.
3. Nuestra organización no dispone de una CMDB.

En el caso 3, partimos de la situación más precaria, y nuestra organización precisa el desarrollo de una CMDB. Nos plantearemos esta situación y desarrollaremos una CMDB específica para nuestro dominio, siguiendo las directrices marcadas por la metodología EODAM y los estándares propuestos en el capítulo cuatro.

El presente capitulo describe el desarrollo de la CMDB. Como ya vimos, una de las actividades principales dentro de la metodología EODAM es la construcción de la base de datos. En nuestro caso, nos centraremos en la construcción de una base de datos de gestión de la configuración (CMDB).

Como ya fue analizado en el capítulo tres, nuestra propuesta incluye una serie de fases para la construcción de la CMDB, en la particularización que realizamos de EODAM para una CMDB, y que son: el análisis de la CMDB, el diseño y la construcción. En este capítulo analizaremos cada una de esas fases y construiremos la CMDB.

Además, en la sección dedicada al análisis de la CMDB, se incluye una planificación del trabajo a llevar a cabo para construir nuestro dominio y una serie de interfaces desde nuestra metodología EODAM a las buenas prácticas de gestión de proyectos para garantizar el control y seguimiento de cualquier proyecto bajo el paraguas de la metodología EODAM.

En las secciones dedicadas al diseño y desarrollo de la CMDB se analizarán las iniciativas actuales de diseño y construcción de CMDB's, enmarcando nuestra propuesta en dichas iniciativas.

Una vez obtenida la CMDB estamos en condiciones de obtener todo el modelo de nuestro dominio. Por ello, se incluye una última sección dedicada al Modelo Completo del Dominio.

7.1.Análisis de la CMDB

En nuestra adaptación de EODAM a una CMDB propusimos como primera fase **el análisis de la CMDB.**

Esta fase incluye las siguientes actividades:

1. ***Elaboración del Bussines Case.***
2. ***Análisis de Requisitos, estándares y Servicios.***
3. ***Inventario de Activos y análisis de Interfaces.***
4. ***Análisis de Ítems de Configuración.***

7.1.1. El Bussines Case

Como ya mencionamos en el capítulo tres, **esta actividad se detallará según establecen las buenas prácticas de gestión de proyectos PRINCE2** [PRINCE2 BC 2010]. PRINCE2 constituye un estándar de facto en la gestión de proyectos TIC a nivel internacional.

Para que EODAM y PRINCE2 estén perfectamente integrados es necesario el desarrollo de un *Bussines Case* *(BC)*. Según PRINCE2, el BC es el primer elemento destacado en el desarrollo de un proyecto, una vez recibido el *mandato* del mismo (su orden de ejecución) [Colin 2010].

La integración entre EODAM y PRINCE2 se realiza considerando todas las fases de EODAM, salvo el desarrollo del BC, como fases técnicas, dentro de la fase de gestión de PRINCE2 *"Managing Product Delivery"* y dejamos para PRINCE2 el resto de las fases de gestión.

En PRINCE2 se diferencian **fases técnicas**, cuyo objetivo son el desarrollo de los productos del proyecto, y **fases de gestión**, cuyo objetivo son el control y seguimiento del proyecto por parte de los roles con esa función.

De la misma forma, PRINCE2 diferencia entre **productos técnicos**, como podrían ser en EODAM: el modelo ampliado del dominio, el modelo completo, los productos generados, el ejemplar, etc. **y productos de gestión**, que según PRINCE2

son: el plan de calidad, la descripción de los productos, el documento de inicio del proyecto (PID), los informes, los planes del proyecto, las lecciones aprendidas, etc.

A continuación, se muestran, en forma de resumen, las fases principales de PRINCE2.

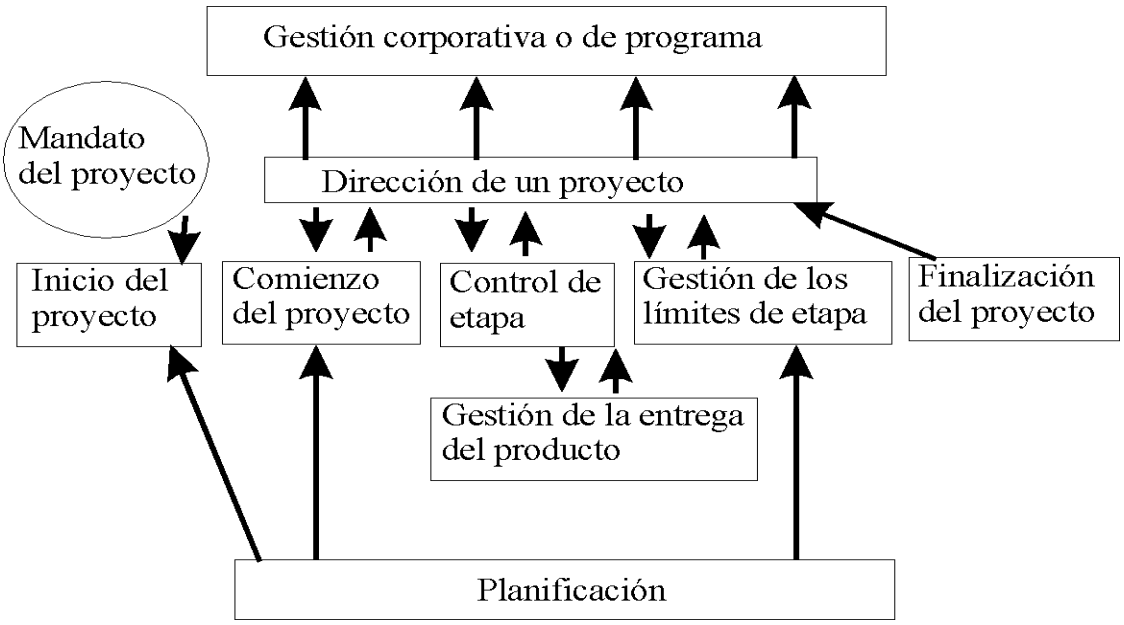


Figura 7-1 Visión General de PRINCE2.

Elaborar el *Bussines Case* nos permite sentar las bases de las fases Inicio y Comienzo del Proyecto. El Control de cada etapa nos permite dar un seguimiento a los entregables de gestión de forma ad hoc, como la dirección del proyecto, pero en el caso de la toma de decisiones de alto nivel.

En cada etapa de gestión se controlan los límites y los entregables de gestión de forma estandarizada siguiendo las directrices de PRINCE2 y la gestión de la entrega de productos se realizará siguiendo las directrices de EODAM.

El *Bussines Case* incluirá la siguiente información: las razones para el desarrollo y los servicios de soporte, las opciones para el desarrollo, los beneficios esperados, los riesgos, una planificación del trabajo y el proceso de evaluación del desarrollo, tal y como recomienda PRINCE2. En nuestro caso**, analizamos nuestro Bussines Case:**

- En lo que se refiere a las **razones**, el desarrollo de una CMDB y los servicios de soporte nos permitirán centralizar todos los ítems de configuración de nuestra organización y el servicio de notificación de cambios nos informará de cualquier cambio a los sistemas de información de nuestra organización, proporcionando un mayor control sobre la infraestructura tecnológica.

- En cuanto a las **opciones para el desarrollo**, en nuestro caso, se plantean dos opciones: la ingeniería de aplicación que conllevaría al desarrollo de todos y cada uno de los productos necesarios y la ingeniería de dominio, que consiste en la aplicación de la metodología EODAM.

- Los **beneficios** que aporta una CMDB para la gestión de la infraestructura de una organización son muy extensos y están relacionados con la calidad del control, la integración y el soporte a la hora de tomar decisiones. Verificar y corregir los CI's da un mayor grado de control sobre la infraestructura que se posee y sobre su seguridad. Además, controlar todos los activos y la configuración o relaciones que cada CI tiene reduce la complejidad y los costes de soporte. Disponer de un servicio de notificación de cambios nos permitirá controlar, por ejemplo, que todo aquello que desaparece o que aparece sin haber sido pagado se pueda notificar inmediatamente, ayudando a controlar los activos y evitando problemas legales. Cuando los procesos de gestión de incidentes, gestión de entregas, riesgos y gestión de cambios están basados en la información de una CMDB, se pueden integrar cada uno de estos procesos, reduciendo costes administrativos y errores.

- En lo que respecta a los **riesgos** en la implementación y el mantenimiento de la CMDB, podemos destacar la falta de recursos, perfiles especializados y/o el coste del desarrollo y el mantenimiento y la propia complejidad en la definición del alcance de la CMDB, que son unos riesgos comunes en este tipo de implementaciones.

- La **planificación del trabajo** se realizará siguiendo las directrices marcadas por las recomendaciones de PRINCE2, y el proceso de Planificación [PRINCE2 PL 2009] incluyendo: un plan de calidad, un plan del proyecto y un plan de excepción. El plan de proyecto contiene un diagrama de actividades y recursos, con hitos y fechas, el plan de calidad y el plan de excepción nos permiten tener procedimentada la forma de actuar ante una emergencia o excepción que se produzca durante el desarrollo del proyecto.

A continuación mostramos los diagramas GANTT del proyecto:

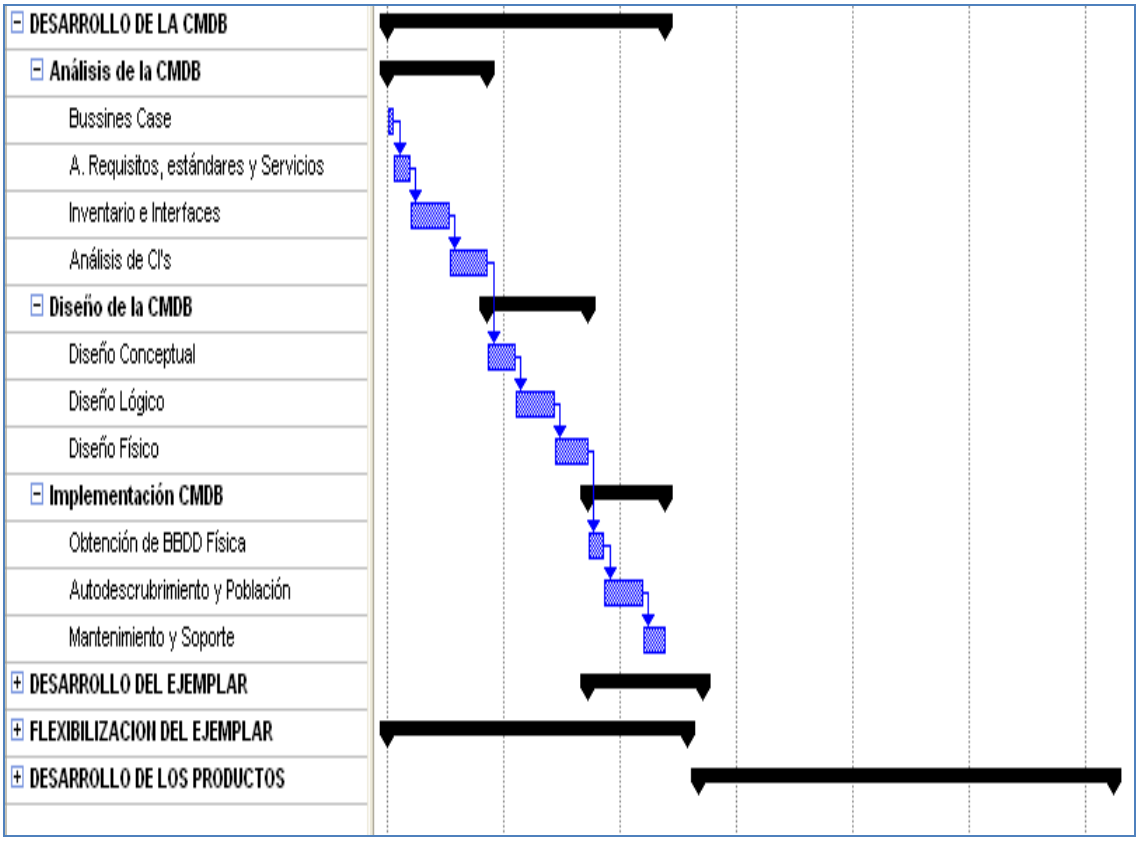


Figura 7-2 Planificación inicial del Proyecto 1 de 2.

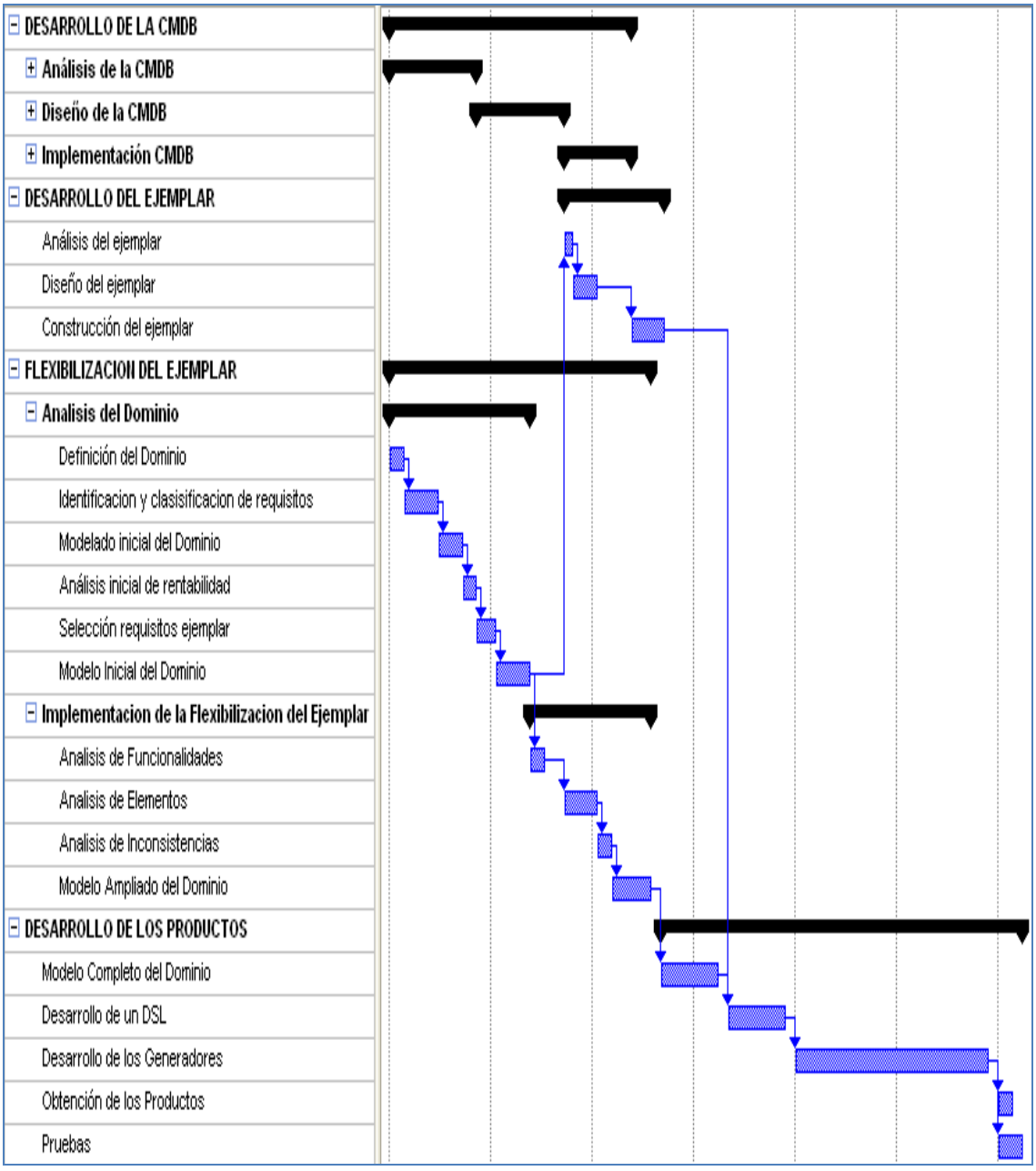


Figura 7-3 Planificación inicial del Proyecto 2 de 2.

Como podemos observar en la planificación inicial del proyecto, existen ciertas dependencias entre las tareas a desarrollar en la metodología EODAM, a parte de la secuencialidad de algunas de ellas.

Por ejemplo, el Análisis del Ejemplar no puede ser realizado hasta que dispongamos del modelo inicial del dominio y hayamos seleccionado los requisitos de nuestro ejemplar.

La construcción del ejemplar no se realizará hasta no disponer de la base de datos construida y el análisis de elementos no se podrá realizar hasta que no se haya diseñado la base de datos. Con respecto al desarrollo de los productos, la construcción del DSL no podrá ser realizada si no tenemos construido nuestro ejemplar.

- Por último, en lo que se refiere al **proceso de evaluación** del desarrollo llevado a cabo, para evaluar la buena marcha del proyecto, seguiremos las directrices marcadas por PRINCE2. La fases de gestión de todo el proyecto nos permitirán poner controles intermedios entre las fases técnicas (según la metodología EODAM). Siguiendo las indicaciones de PRINCE2, las fases de gestión no se solapan y nos permiten, mediante el proceso *"Gestión de los límites de una fase"* [PRINCE2 GLF 2009], que los roles de decisión revisen los entregables del proyecto, costes y tiempos y los validen.

  En nuestro caso, los entregables a validar por los responsables de del proyecto serán los diferentes productos de gestión de nuestro proyecto: el plan de calidad, la descripción de los productos, el documento de inicio del proyecto (PID), los informes, los planes del proyecto y las lecciones aprendidas. Para la revisión de los productos técnicos (el modelo ampliado del dominio, el modelo completo, los productos generados, las pruebas realizadas, el ejemplar, etc.), haremos uso de la técnica propuesta por PRINCE2 *"Técnica de Revisión de la Calidad"* [Hedeman et al. 2006].

7.1.2. Requisitos, estándares y Catalogo de Servicios

La segunda de las actividades de la fase de Análisis de la CMDB es la identificación de los requisitos, que incluirá los requisitos legales, de seguridad y de negocio y los estándares a ser aplicados en su desarrollo.

En nuestro caso, en primer lugar, **elaboraremos la CMDB y el servicio de notificación de cambios, utilizando la metodología EODAM**. **Este desarrollo se integrará en un proyecto que se controlará mediante las buenas prácticas de gestión de proyectos propuestas por PRINCE2**, tal y como ha sido ya analizado en la sección anterior.

Además, **haremos uso de las buenas prácticas recomendadas por ITIL, en lo que se refiere a la servicios a los que dará soporte nuestra CMDB y el servicio de notificación de cambios.**

No estableceremos ningún requisito de seguridad y de negocio adicionales a los ya analizados.

Con respecto a la **identificación del Catálogo de Servicios TIC de la organización** y, en concreto, de aquellos servicios que tienen un impacto en el desarrollo de la CMDB, incluiremos los siguientes servicios:

- ⇒ Servicio de gestión de la configuración y versionado.
- ⇒ Servicio de gestión de incidentes y Servicedesk.
- ⇒ Servicio de gestión de problemas.
- ⇒ Servicio de gestión de niveles de servicio.
- ⇒ Servicio de gestión de contratos.
- ⇒ Servicio de gestión financiera.

Estos servicios de nuestra organización dan soporte a una serie de procesos y su relación con la CMDB ya fue analizada en el capítulo quinto, en la sección ”*Análisis de Procesos relacionados con el Dominio”*.

No se han incluido todos los procesos analizados (como el caso de procesos más especializados como la gestión de la gobernanza, la auditoria, la gestión de la continuidad...), para limitar el alcance de nuestra aplicación. Consideramos que estos procesos seleccionados son muy habituales en las organizaciones responsables de la gestión TIC.

7.1.3. Inventario de Activos y análisis de Interfaces

A continuación, es necesario realizar un **inventario de los activos TIC** de la organización **que van a estar soportados por la CMDB**. En nuestro caso, partiremos de una situación en la que no existe dicho inventario o, al menos, no hay ningún sistema automatizado que nos gestiona los activos de nuestra organización. Es el caso más extremo, pero habitual en muchas organizaciones con un grado de madurez bajo en la gestión TIC.

De existir, este inventario de activos nos permitiría delimitar el alcance de nuestra CMDB. Imaginemos que tenemos inventariados todos los PC’s de los usuarios. Además, que tenemos inventariadas las aplicaciones, las bases de datos y los elementos de la red. Podríamos limitar el alcance de nuestra CMDB a ciertos usuarios y ciertas aplicaciones. Además, será necesario **analizar todos los interfaces sobre los que va a actuar la CMDB**.

En base a los servicios seleccionados, que nos limitan la cobertura de nuestra CMDB, **tendremos los siguientes interfaces:**

- ⇒ **El Sistema de gestión de la configuración y versionado.**
- ⇒ **El Sistema de soporte a la gestión de incidentes y Servicedesk.**
- ⇒ **El Sistema de soporte a la gestión de problemas.**
- ⇒ **El Sistema de soporte a la gestión de niveles de servicio y contratos.**
- ⇒ **El Sistema de gestión financiera.**

Estos sistemas, que dan soporte a los servicios ya seleccionados, interactuarán con nuestra CMDB y con el Servicio de Notificación de Cambios.

Desde el punto de vista de nuestro servicio, **estos sistemas se podrán considerar como suscriptores del servicio**. Cada uno de ellos podrá estar interesado en recibir notificaciones sobre los cambios que se produzcan en la CMDB.

Por ejemplo, el sistema financiero podrá tener interés en recibir los cambios que se produzcan en los PC's de los usuarios. Si, por ejemplo, a un usuario se le va quedar obsoleto un equipo, el sistema financiero podría recibir una notificación para su adquisición.

El sistema de gestión de incidentes y Servicedesk podrá recibir notificaciones cuando un sistema esté caído, así los técnicos de soporte conocerán esta situación cuando un usuario se ponga en contacto con ellos.

El sistema de gestión de la configuración podrá interesarse por cualquier cambio que se produzca y el sistema de gestión de niveles de servicio y contratos puede tener también interés, por ejemplo, cuando haya un nuevo contrato de soporte incluido en la CMDB.

7.1.4. Análisis de Ítems de Configuración

La última fase del Análisis es el estudio de los ítems de configuración, CI's, a los que dará soporte la CMDB. El alcance de los CI viene limitado por el alcance que se establece a nivel de inventario de activos. En nuestro caso, partiremos de una situación en la que no hay establecido un inventario inicial, como ya mencionamos anteriormente. Los ítems de configuración, o entidades de nuestra CMDB que vamos a considerar en nuestro estudio, se describen a continuación.

- **Áreas** de la organización (*locations*). Las localizaciones nos determinarán las zonas físicas de nuestra organización donde están ubicados los ítems de configuración.

- **Proveedores** (*third companies*). Los proveedores serán las empresas externas que proporcionan servicios TIC a nuestra organización, como por ejemplo las empresas de soporte.

- **Contratos** (*contracts*). Los contratos que nuestra organización tiene con las empresas externas de servicios TIC.

- **Recursos humanos** (*people*). Los recursos humanos al servicio de la organización TIC. No se incluyen los usuarios de las aplicaciones. También se considerará como entidad adicional los propietarios (*managers*) de los equipos. Siguiendo las directrices marcadas por las buenas prácticas de seguridad como la ISO/IEC 27002 [ISO27002 2005], designaremos a todos los propietarios de cada equipo.

- **Servicios** (*services*). Los servicios TIC que proporciona nuestra organización.

- **Aplicaciones** (*applications*). Las aplicaciones que soportan nuestros servicios TIC.

- **Paquetes** (*packages*). Los paquetes software de cada una de las aplicaciones.

- **Equipos** (*machines*). Los equipos TIC, incluyendo terminales de usuario, servidores y equipos de red.

Con estos CI's podemos cubrir un gran alcance. En la fase posterior de diseño obtendremos las relaciones y los atributos de los CI's.

7.2.Diseño de la CMDB

Tomaremos como modelo de referencia para el diseño conceptual y lógico de nuestra CMDB el modelo propuesto por [Haziri and Burgess 2009]. Este modelo hereda del **Modelo CIM** (*Common Information Mode*l) [CIM 2010].

El modelo CIM constituye un estándar de facto para representar y organizar toda la información de gestión de un entorno TIC en una organización. CIM puede ser representado haciendo uso del Lenguaje Unificado de Modelado (UML) [UML 2003] y contiene un esquema y una especificación [CIM Sc. Sp. 2010].

El esquema de CIM está compuesto de tres capas: el núcleo de CIM (o "*core*"), el modelo común y el modelo extendido. **El "*core*" de CIM** cubre todos los conceptos que pueden ser aplicables a cualquier infraestructura tecnológica. El **modelo común** describe, de una forma más detallada, diferentes áreas de aplicación específicas como aplicaciones, usuarios, elementos de la red, etc. El **modelo extendido** está parametrizado según el entorno tecnológico, incluyendo aspectos más técnicos como sistemas operativos, bases de datos, servidores de aplicaciones, etc.

La mayor parte de las CMDB del mercado heredan del modelo CIM. Por ejemplo, Hewlett Packard implementa un subconjunto de este modelo denominado *Universal CMDB* [HP CMDB 2010]; IBM también toma CIM como referencia en su modelo *Common Data Model (CDM)* [Tai et al. 2009] y BMC basa su CMDB en un modelo llamado *BMC Atrium CMDB* [BMC CMDB 2010] que también es una herencia del modelo CIM.

El **modelo propuesto tiene varias ventajas**, tal y como se describen en [Haziri and Burgess 2009]: integración con LDAP, facilidad y agilidad en las búsquedas, optimización en la actualización de datos, simplicidad, escalabilidad hacia otros modelos más amplios como el *Universal CMDB*, el *CDM* o el *BMC Atrium* y una comunicación eficiente con los *sistemas de autodescubrimiento*, un elemento clave para una CMDB.

La simplicidad es una propiedad muy importante en una CMDB. El coste de su soporte y mantenimiento puede llegar a ser tan elevado que los beneficios que aporte no superen sus costes. Esto sucede en muchos casos, por lo que en la mayor parte de los proyectos de implantación de las buenas prácticas de ITIL éstos se plantean de forma evolutiva, permitiendo partir de una base más gestionable para ir evolucionado y ampliando el alcance de la CMDB.

Además, este modelo incluye todas las entidades identificadas en nuestra fase de Análisis y cubre nuestro alcance. Realizando la correspondiente adaptación a la normativa expuesta en el capítulo cuatro, el diseño conceptual de **nuestra CMDB quedaría de la siguiente manera:**

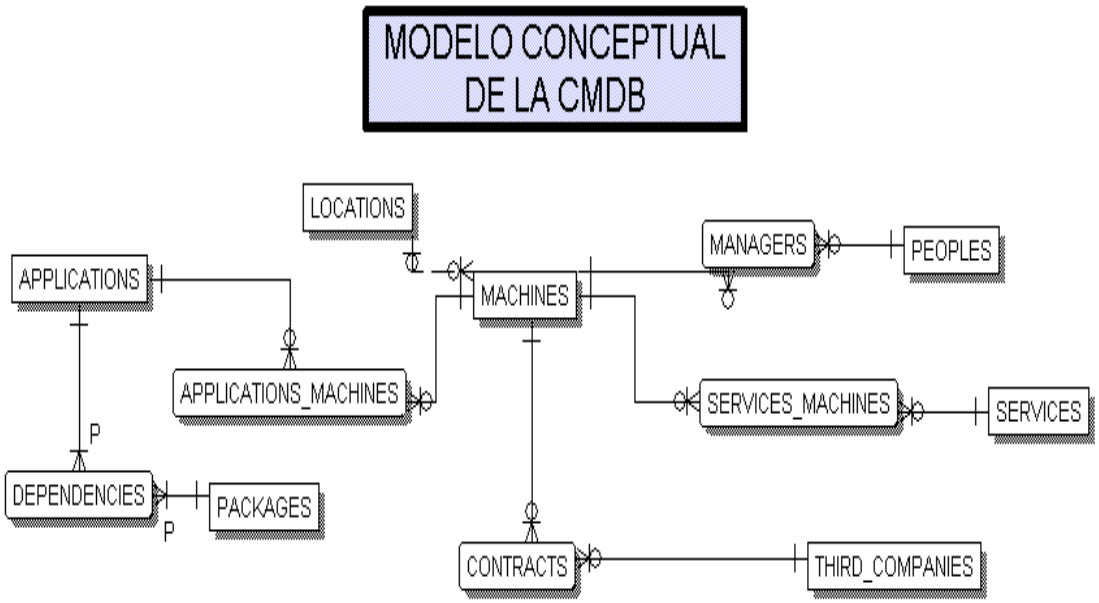


Figura 7-4 Modelo Conceptual de la CMDB.

Se ha utilizado la notación *Information Engineer* [Martin and Finkelstein 1981] para el modelo conceptual.

Como podemos observar, se han incluido algunas entidades de apoyo al modelo como Services_Machines o Applications_Machines que nos establecen la relación entre los Servicios y los Equipos y entre los Paquetes de Software y las Aplicaciones.

Determinando los atributos de los CI's y **transformando al modelo lógico quedaría de la siguiente manera:**

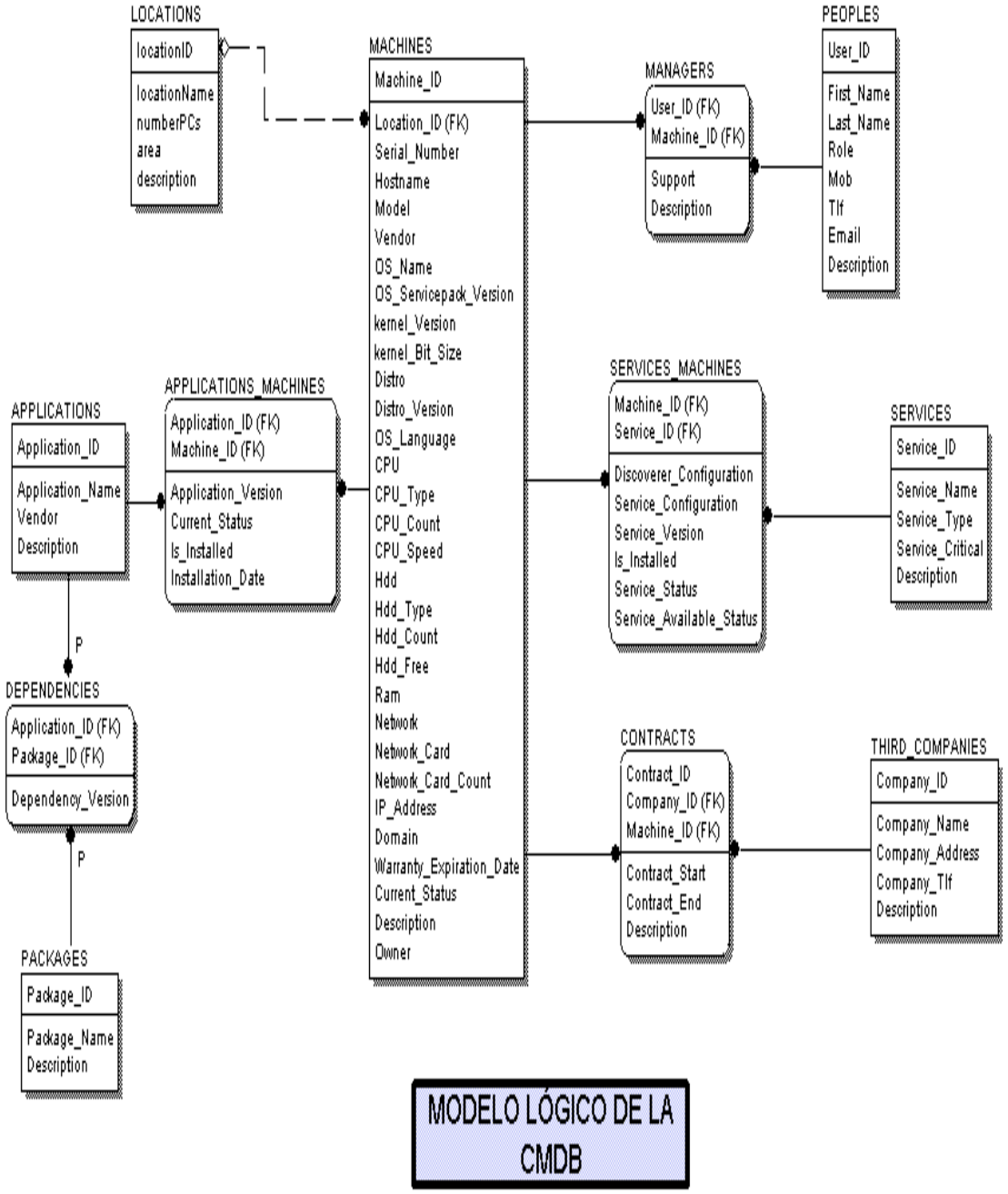


Figura 7-5 Modelo Lógico de la CMDB.

Se ha utilizado la notación IDEF1X tal y como propone la normativa expuesta en el capítulo cuatro.

A la hora de transformar al modelo físico, y considerando su parametrización a una base de datos Oracle, obtenemos el siguiente modelo:

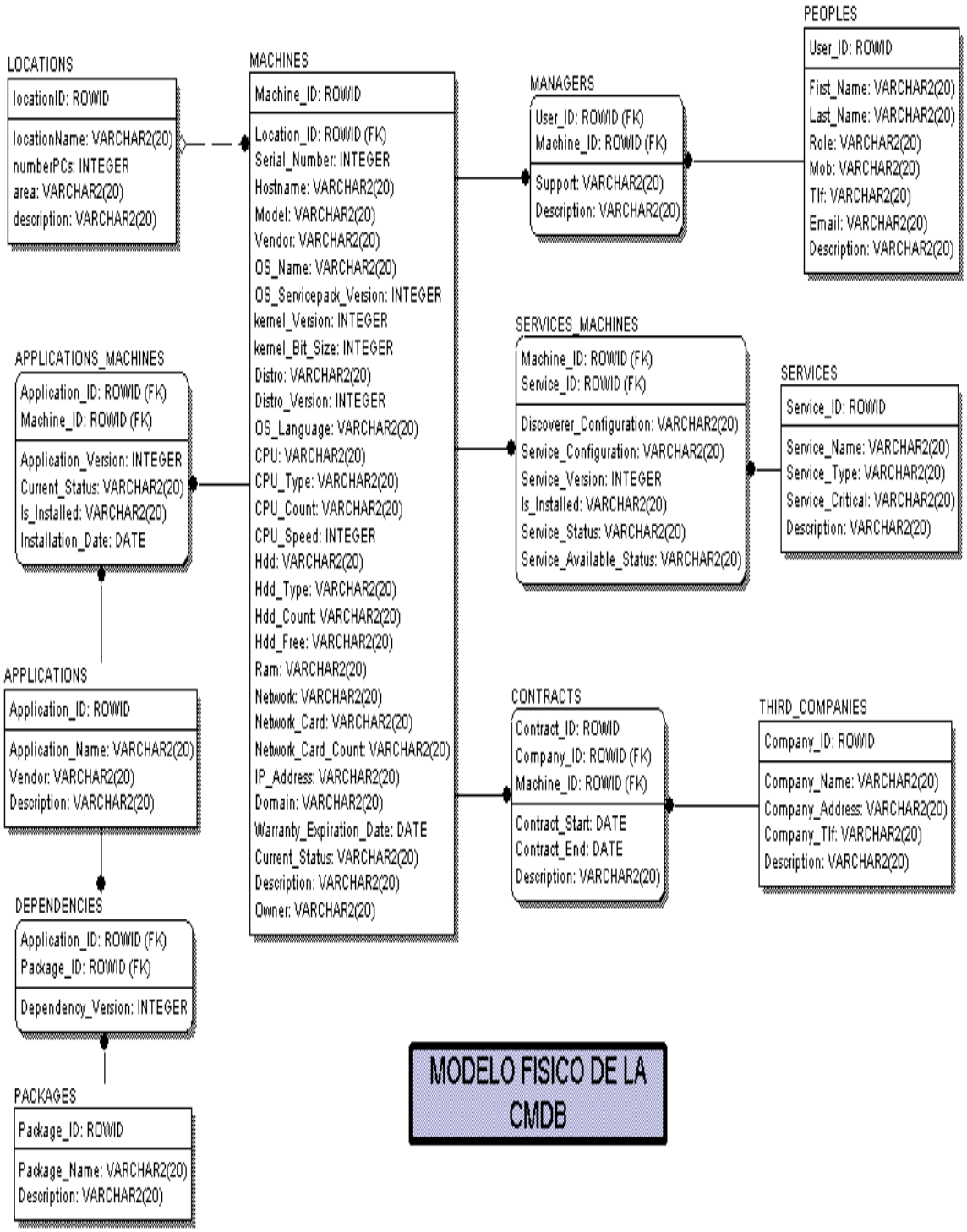


Figura 7-6 Modelo Físico de la CMDB.

De la misma forma que con el Modelo Lógico, se ha seguido la notación IDEF1X, tal y como propone la normativa expuesta en el capítulo cuatro.

7.3.Desarrollo de la CMDB

En la etapa de desarrollo de la CMDB se diferencian tres fases:

1. ***Proceso de auto-descubrimiento.***
2. ***Población inicial de la CMDB.***
3. ***Métricas y mejora continua.***

La primera actividad en la construcción de la base de datos es el **proceso de auto-descubrimiento.** Este proceso es el responsable de obtener toda la información de CI's de la organización de forma automatizada.

Los diferentes fabricantes de soluciones de soporte a ITIL ofrecen diferentes opciones para dar soporte a este proceso. Por ejemplo, en el caso del fabricante Hewlett Packard proporciona la herramienta *Discovery and Dependency Mapping tool (DDM)* [DDM 2010] que descubre componentes de configuración de forma automática, para poder poblar la CMDB (el proceso de población es el posterior al proceso de autodescubrimiento, tal y como se propone en nuestra metodología).

En el caso de IBM, se hace uso de la herramienta *Tivoli Application and Dependency Manager (TADDM)* [TADDM 2010]. Tanto las soluciones de IBM como de HP pueden utilizar un mecanismo de autodescubrimiento sin agentes, lo que significa que no es necesario pre instalar componentes de la herramienta en los diferentes elementos de la red, como puedan ser los servidores de aplicaciones o de bases de datos.

En el caso de BMC, se utiliza el sistema *BMC Atrium Discovery* [BMCAD 2010], una solución basada en servicios web que descubre de forma automática los CI's de nuestra organización y posteriormente los mapea, como las otras soluciones mencionadas, a un esquema predefinido de nuestra CMDB para poder poblar con datos la misma.

Todos estos proveedores permiten trabajar con una base de datos Oracle para dar soporte a las funcionalidades de gestor de base de datos; no obstante, son compatibles con otros productos. En nuestro caso, haremos una parametrización a Oracle y a Hyper SQL [HSQLDB 2010], una base de datos *open source*, entendiendo que de esta manera podemos cubrir alcances muy amplios en nuestros casos de estudio.

En el caso de HSQLDB podemos hacer uso del sistema de soporte para la gestión de la configuración OneCMDB [OCMDB 2009], una herramienta *open source*.

Una vez poblada la base de datos, y según nuestra propuesta metodológica, se formalizarán los procesos de mantenimiento y mejora evolutiva de la misma, dando por finalizada la fase de desarrollo de la CMDB.

7.4.Modelo completo del Dominio

Ya estamos en disposición de obtener el **modelo completo de nuestro dominio**, a partir del modelo ampliado del dominio que describimos en el capítulo seis y la información de la CMDB y de los suscriptores al servicio de notificación de cambios, que actúan como interfaces entre los sistemas de nuestra organización y la CMDB.

En primer lugar, obtendremos el **modelo de elementos**, posteriormente realizaremos una integración de este modelo y el de funcionalidades y finalizaremos con el modelo completo del dominio, que incluye también los requisitos de nuestro dominio.

En el modelo de elementos el primer elemento a considerar son las entidades de nuestra CMDB sobre las cuales nuestro servicio de notificación de cambios informará si hay un cambio.

Los tipos de cambios posibles serán actualizaciones, inserciones y borrados, de tal forma que, considerando las principales entidades, el primer modelo que obtenemos es:

```
####################
# MODELO DE ELEMENTOS
####################

# Tables ID with Notifications
node('TablesID', ['LOCATIONS', 'MACHINES', 'MANAGERS', 'PEOPLE', 'CONTRACTS',
'THIRD_COMPANIES', 'PACKAGES', 'APPLICATIONS'],1,8)

# Operations over Tables with Notifications
node('LOCATIONS', ['I_LOCATIONS', 'D_LOCATIONS', 'U_LOCATIONS'],1,3)
node('MACHINES', ['I_MACHINES', 'D_MACHINES', 'U_MACHINES'],1,3)
node('MANAGERS', ['I_MANAGERS', 'D_MANAGERS', 'U_MANAGERS'],1,3)
node('PEOPLE', ['I_PEOPLE', 'D_PEOPLE', 'U_PEOPLE'],1,3)
node('CONTRACTS', ['I_CONTRACTS', 'D_CONTRACTS', 'U_CONTRACTS'],1,3)
node('THIRD_COMPANIES', ['I_THIRD_COMPANIES', 'D_THIRD_COMPANIES',
'U_THIRD_COMPANIES'],1,3)
node('PACKAGES', ['I_PACKAGES', 'D_PACKAGES', 'U_PACKAGES'],1,3)
node('APPLICATIONS', ['I_APPLICATIONS', 'D_APPLICATIONS',
'U_APPLICATIONS'],1,3)
```

Otro elemento importante en nuestro análisis son los **suscriptores al servicio de notificación de cambios.**

Los posibles suscriptores, ya identificados anteriormente (el sistema de gestión de la configuración, el Servicedesk, el de gestión de problemas, el de soporte a la gestión de SLA's y el sistema financiero), en caso de que estén interesados en recibir las notificaciones, será necesario establecer los correspondientes permisos. Podríamos representarlo de la siguiente manera:

```
########################
# MODELO DE SUBSCRIPTORES
########################

# Notifications over Subscribers
node('SubscribersID', ['Conf_Sys', 'Servicedesk', 'Prob_Mana_Sys',
'SLA_Mana_Sys', 'Financial_Sys'],1,5)

# Permissions over Subscribers
node('Conf_Sys', ['Conf_Sys_PYES', 'Conf_Sys_PNO'],1,1)
node('Servicedesk', ['Servicedesk_PYES', 'Servicedesk_PNO'],1,1)
node('Prob_Mana_Sys', ['Prob_Mana_Sys_PYES', 'Prob_Mana_Sys_PNO'],1,1)
node('SLA_Mana_Sys', ['SLA_Mana_Sys_PYES', 'SLA_Mana_Sys_PNO'],1,1)
node('Financial_Sys', ['Financial_Sys_PYES', 'Financial_Sys_PNO'],1,1)
```

Haciendo uso de las herramientas de transformación gráfica obtenemos la siguiente representación visual de este diagrama:

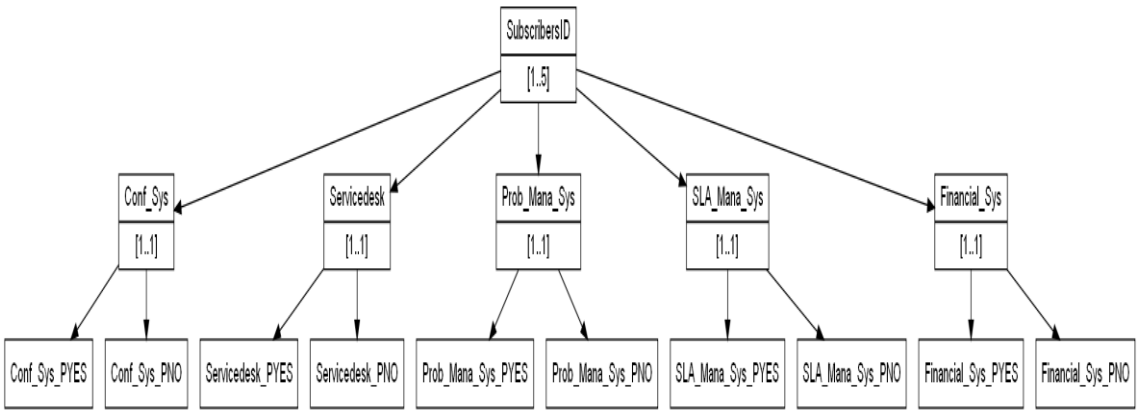


Figura 7-7 Modelo de Elementos. Subscriptores.

De esta forma, podremos parametrizar si estamos o no interesados en que ciertos suscriptores reciban o no notificaciones.

Podemos incluir, además, ciertos requisitos adicionales a cada uno de estos cambios. Por ejemplo, incluir las prioridades, las expiraciones y los retrasos de cada uno de esos cambios. Este aspecto complica más el modelo de elementos, obteniendo:

```
################################
# MODELO SUBSCRIPTORES Y ELEMENTOS
################################
# Only Operations in Elements
node('Elements',['TablesID','SubscribersID'],2,2)
# MODELO DE SUBSCRIPTORES (código)
# MODELO DE ELEMENTOS (código)
# Options over Operations with Notifications
node('I_LOCATIONS', ['I_LOCATIONS_P', 'I_LOCATIONS_E', 'I_LOCATIONS_R'],1,3)
node('I_MACHINES', ['I_MACHINES_P', 'I_MACHINES_E', 'I_MACHINES_R'],1,3)
node('I_MANAGERS', ['I_MANAGERS_P', 'I_MANAGERS_E', 'I_MANAGERS_R'],1,3)
node('I_PEOPLE', ['I_PEOPLE_P', 'I_PEOPLE_E', 'I_PEOPLE_R'],1,3)
node('I_CONTRACTS', ['I_CONTRACTS_P', 'I_CONTRACTS_E', 'I_CONTRACTS_R'],1,3)
node('I_THIRD_COMPANIES', ['I_THIRD_COMPANIES_P', 'I_THIRD_COMPANIES_E',
'I_THIRD_COMPANIES_R'],1,3)
node('I_PACKAGES', ['I_PACKAGES_P', 'I_PACKAGES_E', 'I_PACKAGES_R'],1,3)
node('I_APPLICATIONS', ['I_APPLICATIONS_P', 'I_APPLICATIONS_E',
'I_APPLICATIONS_R'],1,3)
node('D_LOCATIONS', ['D_LOCATIONS_P', 'D_LOCATIONS_E', 'D_LOCATIONS_R'],1,3)
node('D_MACHINES', ['D_MACHINES_P', 'D_MACHINES_E', 'D_MACHINES_R'],1,3)
node('D_MANAGERS', ['D_MANAGERS_P', 'D_MANAGERS_E', 'D_MANAGERS_R'],1,3)
node('D_PEOPLE', ['D_PEOPLE_P', 'D_PEOPLE_E', 'D_PEOPLE_R'],1,3)
node('D_CONTRACTS', ['D_CONTRACTS_P', 'D_CONTRACTS_E', 'D_CONTRACTS_R'],1,3)
node('D_THIRD_COMPANIES', ['D_THIRD_COMPANIES_P', 'D_THIRD_COMPANIES_E',
'D_THIRD_COMPANIES_R'],1,3)
node('D_PACKAGES', ['D_PACKAGES_P', 'D_PACKAGES_E', 'D_PACKAGES_R'],1,3)
node('D_APPLICATIONS', ['D_APPLICATIONS_P', 'D_APPLICATIONS_E',
'D_APPLICATIONS_R'],1,3)
node('U_LOCATIONS', ['U_LOCATIONS_P', 'U_LOCATIONS_E', 'U_LOCATIONS_R'],1,3)
node('U_MACHINES', ['U_MACHINES_P', 'U_MACHINES_E', 'U_MACHINES_R'],1,3)
node('U_MANAGERS', ['U_MANAGERS_P', 'U_MANAGERS_E', 'U_MANAGERS_R'],1,3)
node('U_PEOPLE', ['U_PEOPLE_P', 'U_PEOPLE_E', 'U_PEOPLE_R'],1,3)
node('U_CONTRACTS', ['U_CONTRACTS_P', 'U_CONTRACTS_E', 'U_CONTRACTS_R'],1,3)
node('U_THIRD_COMPANIES', ['U_THIRD_COMPANIES_P', 'U_THIRD_COMPANIES_E',
'U_THIRD_COMPANIES_R'],1,3)
node('U_PACKAGES', ['U_PACKAGES_P', 'U_PACKAGES_E', 'U_PACKAGES_R'],1,3)
node('U_APPLICATIONS', ['U_APPLICATIONS_P', 'U_APPLICATIONS_E',
'U_APPLICATIONS_R'],1,3)
```

No representaremos este modelo de elementos de forma gráfica por su complejidad.

En lo que se refiere a las restricciones, en el modelo completo del dominio debemos de incluir todas ellas. De lo contrario, por ejemplo, tendríamos combinaciones de requisitos no permitidas y la configuración completa de nuestro dominio no sería la correcta.

Por ejemplo, una de las restricciones de nuestro modelo será que si seleccionamos el requisito de prioridad, todas las operaciones sobre las entidades de nuestra CMDB deberán tener una prioridad asignada. Su representación en formato NFTE sería:

```
# With Priority every Operation has Priority
constraint('MCP => (I_LOCATIONS_P or D_LOCATIONS_P or U_LOCATIONS_P or
I_MACHINES_P or D_MACHINES_P or U_MACHINES_P or I_MANAGERS_P or D_MANAGERS_P
or U_MANAGERS_P or I_PEOPLE_P or D_PEOPLE_P or U_PEOPLE_P or I_CONTRACTS_P or
D_CONTRACTS_P or U_CONTRACTS_P or I_THIRD_COMPANIES_P or D_THIRD_COMPANIES_P
or U_THIRD_COMPANIES_P or I_PACKAGES_P or D_PACKAGES_P or U_PACKAGES_P or
I_APPLICATIONS_P or D_APPLICATIONS_P or U_APPLICATIONS_P)')
# Without Priority is Not possible Operations with Priority
constraint('MSP => not (I_LOCATIONS_P or D_LOCATIONS_P or U_LOCATIONS_P or
I_MACHINES_P or D_MACHINES_P or U_MACHINES_P or I_MANAGERS_P or D_MANAGERS_P
or U_MANAGERS_P or I_PEOPLE_P or D_PEOPLE_P or U_PEOPLE_P or I_CONTRACTS_P or
D_CONTRACTS_P or U_CONTRACTS_P or I_THIRD_COMPANIES_P or D_THIRD_COMPANIES_P
or U_THIRD_COMPANIES_P or I_PACKAGES_P or D_PACKAGES_P or U_PACKAGES_P or
I_APPLICATIONS_P or D_APPLICATIONS_P or U_APPLICATIONS_P)')
```

Otra restricción de nuestro modelo se refiere a los requisitos de expiración o retraso. Si seleccionamos estos requisitos todas las operaciones sobre las entidades de nuestra CMDB deberán tener requisitos de expiración y/o retraso. Su representación en formato NFTE sería:

```
# With Expiration or Delay and Expiration every Operation has Expiration
constraint('GTE => (I_LOCATIONS_P or D_LOCATIONS_P or U_LOCATIONS_P or
I_MACHINES_P or D_MACHINES_P or U_MACHINES_P or I_MANAGERS_P or D_MANAGERS_P
or U_MANAGERS_P or I_PEOPLE_P or D_PEOPLE_P or U_PEOPLE_P or I_CONTRACTS_P or
D_CONTRACTS_P or U_CONTRACTS_P or I_THIRD_COMPANIES_P or D_THIRD_COMPANIES_P
or U_THIRD_COMPANIES_P or I_PACKAGES_P or D_PACKAGES_P or U_PACKAGES_P or
I_APPLICATIONS_P or D_APPLICATIONS_P or U_APPLICATIONS_P)')
constraint('GTER => (I_LOCATIONS_P or D_LOCATIONS_P or U_LOCATIONS_P or
I_MACHINES_P or D_MACHINES_P or U_MACHINES_P or I_MANAGERS_P or D_MANAGERS_P
or U_MANAGERS_P or I_PEOPLE_P or D_PEOPLE_P or U_PEOPLE_P or I_CONTRACTS_P or
D_CONTRACTS_P or U_CONTRACTS_P or I_THIRD_COMPANIES_P or D_THIRD_COMPANIES_P
```

```
or U_THIRD_COMPANIES_P or I_PACKAGES_P or D_PACKAGES_P or U_PACKAGES_P or
I_APPLICATIONS_P or D_APPLICATIONS_P or U_APPLICATIONS_P)')
# Without Expiration or with only Delay is Not possible Operations with
Expiration
constraint('GTS => not (I_LOCATIONS_P or D_LOCATIONS_P or U_LOCATIONS_P or
I_MACHINES_P or D_MACHINES_P or U_MACHINES_P or I_MANAGERS_P or D_MANAGERS_P
or U_MANAGERS_P or I_PEOPLE_P or D_PEOPLE_P or U_PEOPLE_P or I_CONTRACTS_P or
D_CONTRACTS_P or U_CONTRACTS_P or I_THIRD_COMPANIES_P or D_THIRD_COMPANIES_P
or U_THIRD_COMPANIES_P or I_PACKAGES_P or D_PACKAGES_P or U_PACKAGES_P or
I_APPLICATIONS_P or D_APPLICATIONS_P or U_APPLICATIONS_P)')
constraint('GTR => (I_LOCATIONS_P or D_LOCATIONS_P or U_LOCATIONS_P or
I_MACHINES_P or D_MACHINES_P or U_MACHINES_P or I_MANAGERS_P or D_MANAGERS_P
or U_MANAGERS_P or I_PEOPLE_P or D_PEOPLE_P or U_PEOPLE_P or I_CONTRACTS_P or
D_CONTRACTS_P or U_CONTRACTS_P or I_THIRD_COMPANIES_P or D_THIRD_COMPANIES_P
or U_THIRD_COMPANIES_P or I_PACKAGES_P or D_PACKAGES_P or U_PACKAGES_P or
I_APPLICATIONS_P or D_APPLICATIONS_P or U_APPLICATIONS_P)')
```

Del mismo modo, tendríamos con el requisito de retención de los mensajes. Su representación en formato NFTE sería:

```
# With Retention (value or infinite) every Operation has Retention
constraint('GTCRI => (I_LOCATIONS_P or D_LOCATIONS_P or U_LOCATIONS_P or
I_MACHINES_P or D_MACHINES_P or U_MACHINES_P or I_MANAGERS_P or D_MANAGERS_P
or U_MANAGERS_P or I_PEOPLE_P or D_PEOPLE_P or U_PEOPLE_P or I_CONTRACTS_P or
D_CONTRACTS_P or U_CONTRACTS_P or I_THIRD_COMPANIES_P or D_THIRD_COMPANIES_P
or U_THIRD_COMPANIES_P or I_PACKAGES_P or D_PACKAGES_P or U_PACKAGES_P or
I_APPLICATIONS_P or D_APPLICATIONS_P or U_APPLICATIONS_P)')

constraint('GTCR => (I_LOCATIONS_P or D_LOCATIONS_P or U_LOCATIONS_P or
I_MACHINES_P or D_MACHINES_P or U_MACHINES_P or I_MANAGERS_P or D_MANAGERS_P
or U_MANAGERS_P or I_PEOPLE_P or D_PEOPLE_P or U_PEOPLE_P or I_CONTRACTS_P or
D_CONTRACTS_P or U_CONTRACTS_P or I_THIRD_COMPANIES_P or D_THIRD_COMPANIES_P
or U_THIRD_COMPANIES_P or I_PACKAGES_P or D_PACKAGES_P or U_PACKAGES_P or
I_APPLICATIONS_P or D_APPLICATIONS_P or U_APPLICATIONS_P)')
# Without Retention is Not posible Operations with Retention
constraint('GTSR => not (I_LOCATIONS_P or D_LOCATIONS_P or U_LOCATIONS_P or
I_MACHINES_P or D_MACHINES_P or U_MACHINES_P or I_MANAGERS_P or D_MANAGERS_P
or U_MANAGERS_P or I_PEOPLE_P or D_PEOPLE_P or U_PEOPLE_P or I_CONTRACTS_P or
D_CONTRACTS_P or U_CONTRACTS_P or I_THIRD_COMPANIES_P or D_THIRD_COMPANIES_P
or U_THIRD_COMPANIES_P or I_PACKAGES_P or D_PACKAGES_P or U_PACKAGES_P or
I_APPLICATIONS_P or D_APPLICATIONS_P or U_APPLICATIONS_P)')
```

Para obtener nuestros diferentes diagramas NFTE hemos desarrollado unos algoritmos que nos permiten **automatizar el NFTE a partir de las entidades y los suscriptores**, **de tal forma que si operásemos con una CMDB y unos sistemas diferentes, los diagramas serían automáticamente obtenidos.**

Si incluimos, además, las restricciones, las funcionalidades y los requisitos, nuestro modelo completo del dominio quedaría:

```
##############################
# Modelo Completo del Dominio #
##############################
node('MCD',['MAD','Elements'],2,2)
node('MAD',['SPD','FG'],2,2)
node('SPD',['GT','GGT','GE','GS','G','GP','TV','TN','GB','AM'],10,10)
node('GT',['GTSR','GTCR','GTCRI'],1,1)
node('GGT',['GTS','GTE','GTR','GTER'],1,1)
node('GE',['DSE','DECV','DEI'],1,1)
node('GS',['SU','SM'],1,1)
node('G',['GG','GM','GF'],1,1)
node('GP',['MSP','MCP'],1,1)
node('TV',['VE','VD'],2,2)
node('VE',['EVI','EVT'],1,1)
node('VD',['DVI','DVT'],1,1)
node('TN',['MN','DM'],2,2)
node('MN',['MNN','NMBL','NMBO'],1,1)
node('DM',['NPM','NSM','NST'],1,1)
node('GB',['BCC','BSC'],1,1)
node('AM',['MSA','MCA'],1,1)
node('FG',['FCD','FHD','FDD','OM','FCX'],5,5)
node('FCD',['FCD_ID','FCD_AD','FCD_BD'],1,3)
node('FHD',['FHD_ID','FHD_AD','FHD_BD'],1,3)
node('FDD',['FDD_ID','FDD_AD','FDD_BD'],1,3)
node('OM',['FIOM','FFOM','FBOM'],3,3)
constraint('BCC => MSA')
constraint('BCC => MNN')
constraint('MCA => MNN')
constraint('MCA => MNN')
constraint('FCD_ID => FHD_ID and FDD_ID')
constraint('FCD_AD => FHD_AD and FDD_AD')
constraint('FCD_BD => FHD_BD and FDD_BD')
# MODELO SUBSCRIPTORES Y ELEMENTOS (código)
# CONTRAINTS DEL MODELO (código)
```

A continuación se muestran algunas notaciones para mayor comprensión del código:

- **MCD:** Modelo Completo del Dominio.
- **MAD**: Modelo Ampliado del Dominio.
- **GT**: Gestión del Tiempo.
- **GGT**: Gestión General del Tiempo.
- **GE**: Gestión de Esperas.
- **GS**: Gestión de Suscriptores.
- **G**: Granularidad de la Solución.
- **GP**: Gestión de Prioridades.
- **TV**: Tipos de Visibilidad.
- **TN**: Tipos de Navegación.
- **GB**: Gestión de Búsquedas.
- **AM**: Agrupación de Mensajes.
- **FG**: Funcionalidades Generales.
- **SPD**: Sistema con Persistencia y en Diferido.
- **MAD**: Modelo Ampliado del Dominio.
- **VE**: Visibilidad de Encolado.
- **VD**: Visibilidad de Desencolado.
- **MN**: Modo de Navegación.
- **DM**: Desplazamiento de los Mensajes.
- **ID**: Inserciones de datos, AD: Actualizaciones de datos, BD: Borrado.
- **OM**: Funcionalidades de Operaciones Masivas.

7.5.Métricas del Dominio

Estudiaremos las métricas principales de nuestro Dominio. **Hay diferentes ámbitos de nuestro Dominio sobre los que podemos analizar estas métricas**; cada uno de ellos determinará unas métricas para nuestro dominio. Los principales ámbitos son:

- **Los Requisitos.** Los requisitos del dominio constituyen el primer eslabón de la cadena de nuestro dominio. Una configuración específica de requisitos dará lugar a una serie de productos concretos. El alcance de la variabilidad de los requisitos determinará, en gran medida, el alcance de nuestro dominio.

- **Los Suscriptores.** Los suscriptores al servicio de notificación de cambios también tienen un impacto en nuestro dominio. En función del número de suscriptores, nuestro dominio tendrá un determinado alcance. Por ejemplo, si nuestro dominio tiene un solo suscriptor, un servicio de mensajería que recibe los cambios producidos en los ítems de configuración, nuestro dominio tendrá un menor alcance que si tenemos varios: el sistema financiero, el de gestión de incidentes y problemas, el de gestión del cambio, el sistema de control de versiones, etc.

- **Los Elementos.** Los elementos o entidades que, teniendo una dependencia interna con la CMDB, tienen algún tipo de incidencia en el desarrollo de los productos del dominio, también determinarán el alcance de nuestro dominio. Una CMDB formada por una sola entidad lógica conllevará un alcance mucho más limitado que una CMDB formada por ocho entidades principales, como en nuestro caso de estudio.

- **Las Funcionalidades.** Las funcionalidades necesarias para dar soporte a nuestro dominio es otro aspecto a considerar que en el alcance de nuestro dominio. A mayor número de funcionalidades mayor será el alcance.

A continuación analizamos las métricas de cada uno de ellos.

Las **métricas relacionadas con el ámbito de los requisitos y de las funcionalidades** ya fueron analizadas en el capítulo seis, en la sección 6.5 *Métricas iniciales del Dominio*.

El número de productos diferentes que podemos obtener, según la elección de nuestros requisitos, es de de 25.920 y **de 61, según las funcionalidades.** Además, en esta sección se obtuvieron métricas como la homogeneidad, la comunalidad de los requisitos y el número de productos diferentes por cada requisito.

Estas métricas iniciales nos determinaban un gran alcance para nuestro dominio, pero que debe ser completado con las nuevas métricas del resto de ámbitos.

Con respecto al **ámbito de los suscriptores**, haciendo uso de las herramientas de soporte a NFTE, si utilizamos el modelo de suscriptores completo (ver figura 7-7), con los cinco sistemas de información que actúan como suscriptores, seleccionados para nuestro caso de estudio, obtenemos la siguiente información:

El número de productos diferentes para cinco suscriptores es de 242 y la comunalidad de cada suscriptor es de 0.67. En función del número de suscriptores que seleccionemos, el número de productos variará. **A mayor número de suscriptores, mayor número de productos. Si hacemos un análisis de esta variación obtenemos la siguiente gráfica:**

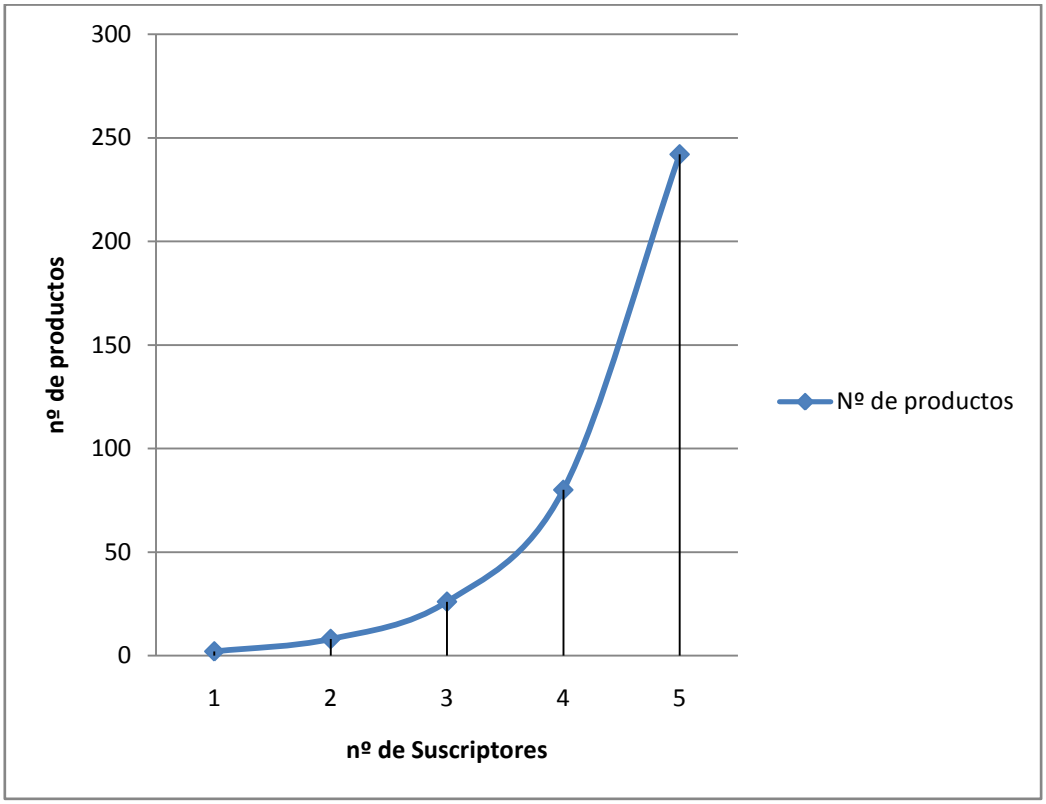


Figura 7-8 nº de productos según el nº de Suscriptores.

Con respecto al **ámbito de los Elementos**, la complejidad de nuestro dominio aumenta considerablemente. Como ya hemos analizado con anterioridad, en este caso, no solamente hay que considerar el número de entidades de nuestro dominio, sino que existe una relación entre los requisitos y las entidades.

Por ejemplo, si seleccionamos el requisito de prioridad podríamos establecer una prioridad de algunas de nuestras entidades; si seleccionamos los requisitos de expiración, retrasos o esperas podríamos establecer algunos parámetros con ciertas entidades y con otras no y un largo etcétera.

Para nuestro caso de estudio hemos seleccionado ocho entidades de la CMDB que estarán bajo el control de los cambios: las localizaciones de los CI, los equipos TIC (máquinas), los empleados TIC, los contratos, las aplicaciones, las compañías proveedoras, los gestores TIC y los paquetes software.

Considerando el modelo de elementos descrito en la sección anterior, que incluye requisitos como las prioridades, retrasos, expiraciones y retenciones, y cualquier tipo de operación sobre las entidades: actualizaciones, borrados o inserciones, **el nº de productos de nuestro dominio sería del orden de 1,6*10^{7}**. La comunalidad de cada una de las entidades tiene un valor de 0,875 y la de cada una de las operaciones de 0,5.

En función del número de elementos que seleccionemos, el número de productos variará. **A mayor número de elementos, mayor número de productos. Si hacemos un análisis de esta variación obtenemos la gráfica 7-10,** donde se puede apreciar como varía el alcance de nuestro dominio en función del número de entidades consideradas.

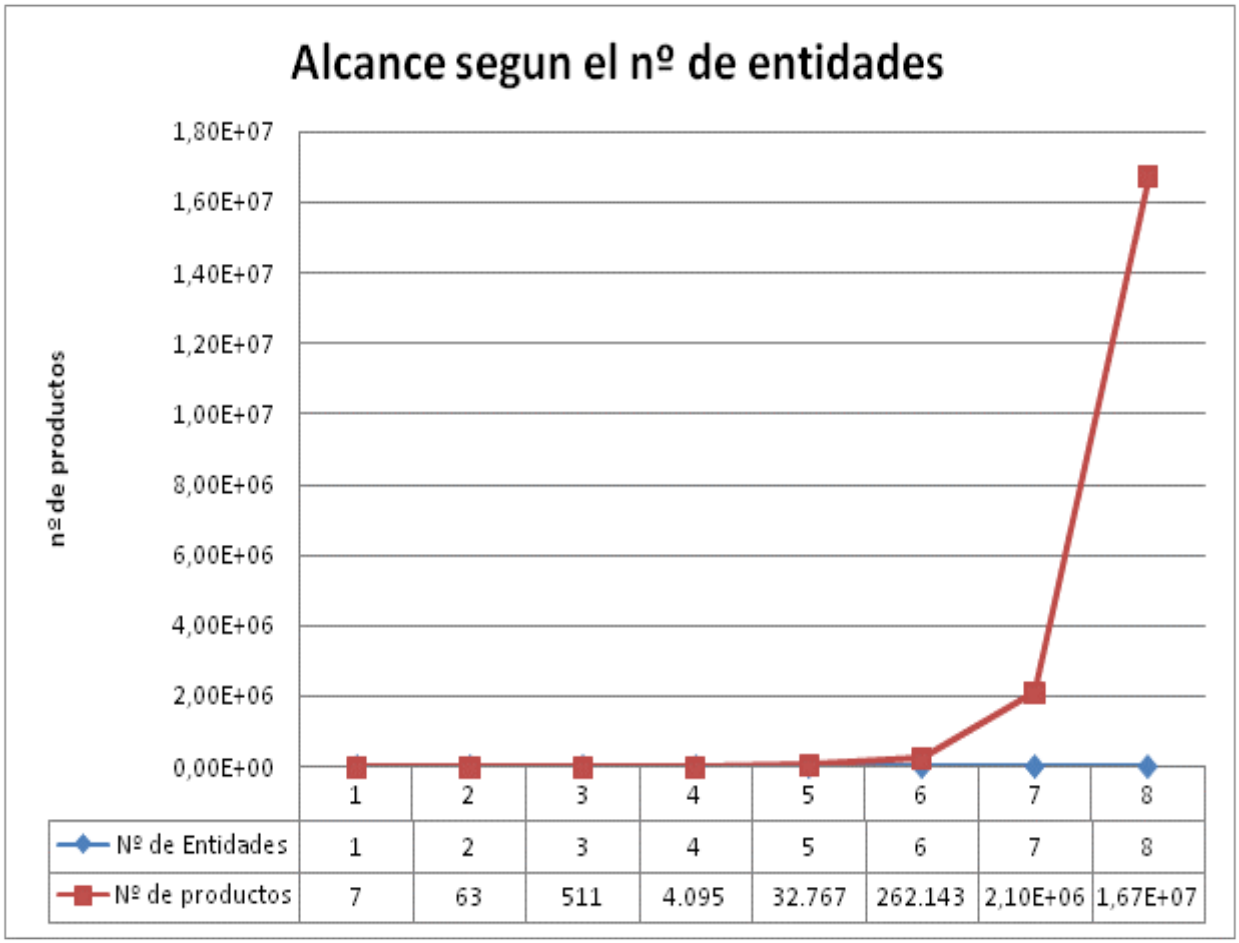
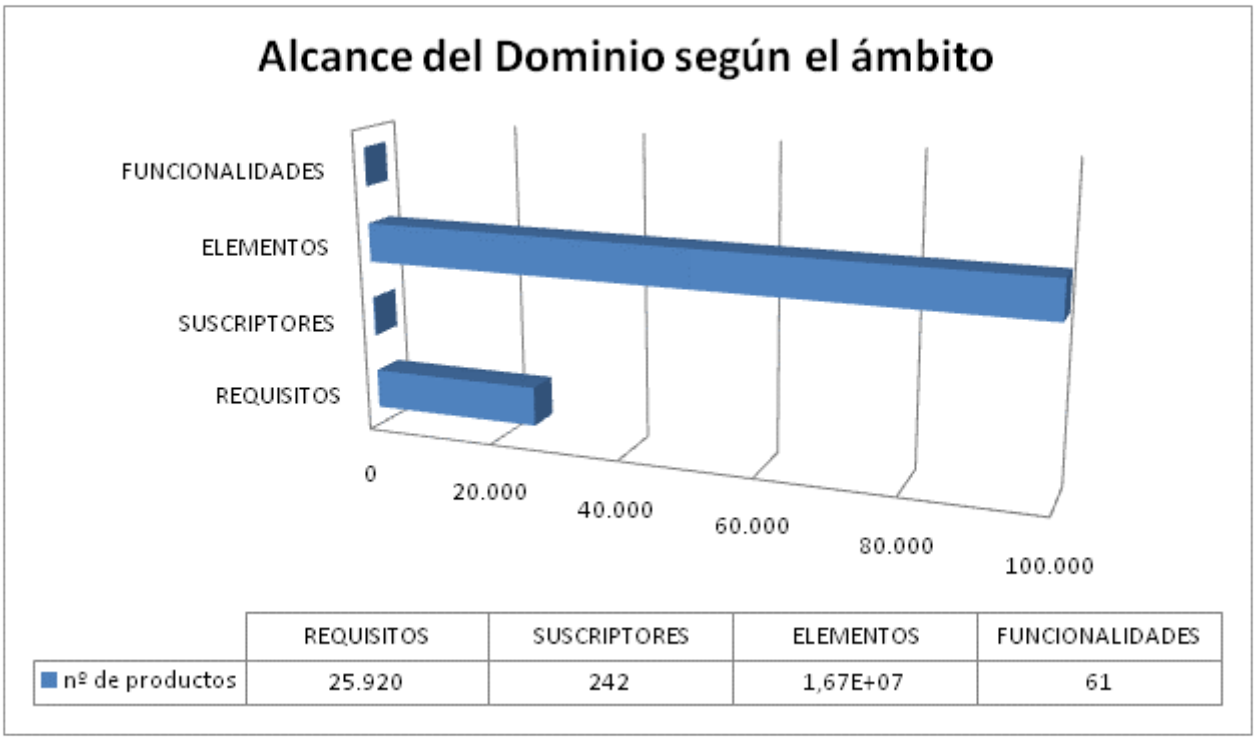


Figura 7-9 nº de productos según el nº de Entidades de la CMDB.

A modo de resumen, **representamos la métrica principal de nuestro dominio (el número de productos) para los principales ámbitos,** en la figura 7-11. Como podemos observar el alcance de nuestro dominio tiene una mayor dependencia con los elementos. **A mayor complejidad de nuestra CMDB, mayor alcance tendrá nuestro dominio.**



	REQUISITOS	SUSCRIPTORES	ELEMENTOS	FUNCIONALIDADES
nº de productos	25.920	242	1,67E+07	61

Figura 7-10 nº de productos según los ámbitos de nuestro Dominio.

Si realizamos un análisis de todas las métricas, podríamos obtener métricas integradas y más completas de nuestro dominio.

Incluyendo la variabilidad de todos los ámbitos, las métricas completas de nuestro dominio nos indican que, para el caso más complejo, considerando todas las entidades y los elementos relacionados, los cinco suscriptores, los requisitos y las funcionalidades, el nº posible de productos se eleva a la cifra de **$1{,}36*10^{30}$.**

Considerando únicamente un suscriptor esta cifra bajaría y sería de $3,73*10^{10}$ y considerando únicamente una entidad la cifra baja hasta los $4,36*10^{10}$ productos, lo que nos indica el gran impacto de la CMDB para determinar el alcance del dominio.

En el capítulo 10, dedicado al análisis de costes y la rentabilidad de la línea de productos, serán analizadas todas estas métricas y la relación entre la CMDB y el número de productos de nuestra SPL.

8. Implementación de un Servicio de Notificación de Cambios en una CMDB

"No trabajé duro para hacer
Ruby perfecto para todo el mundo,
porque todos somos diferentes.
Intenté hacer Ruby perfecto para mí,
así que puede que a ti no te lo parezca;
probablemente, el mejor lenguaje
para Guido van Rossum es Python"
Yukihiro Matsumoto, "Matz", creador de Ruby

Siguiendo el proceso metodológico propuesto por EODAM, una vez construida la Base de Datos (en nuestro caso, la CMDB) y obtenidos algunos entregables de la fase de flexibilización del ejemplar, como el modelo ampliado del dominio y el modelo completo, es necesario desarrollar un ejemplar y flexibilizarlo.

Una vez llevada a cabo la implementación de la flexibilización del ejemplar, el siguiente paso será la obtención de los productos, la última fase de EODAM.

En la sección posterior del capítulo se describe detalladamente el desarrollo de un ejemplar del dominio.

Para la obtención de los productos se ha construido un Framework de desarrollo que incluye diversas utilidades como transformadores, detectores de inconsistencias, generadores, herramientas para la elaboración de informes, etc. En la última sección se detalla el funcionamiento de este Framework.

8.1.Desarrollo del Ejemplar

Para **desarrollar el ejemplar**, nos centraremos en la **selección de requisitos** del ejemplar realizada en el capítulo quinto. El ejemplar deberá cubrir los requisitos y las funcionalidades identificados en esa fase. Además, desde nuestro modelo completo del dominio, identificaremos aquellos elementos que seleccionaremos para nuestro ejemplar y los interfaces de la base de datos, en nuestro caso los subscriptores al servicio de notificación de cambios.

Seleccionamos, para nuestro ejemplar, dos suscriptores al servicio de notificación de cambios: el sistema financiero de nuestra organización y el ERP de soporte a la gestión de niveles de servicio y contratos.

El sistema financiero estará interesado en cualquier tipo de cambio que se produzca en las entidades de contratos y en las inserciones de nuevos proveedores. Este sistema deberá recibir toda la información de grano fino de dicho cambio, lo cual significa que el sistema conocerá toda la información del cambio, a nivel de registro de la CMDB.

El sistema ERP estará interesado únicamente en los cambios a nivel de personal (entidad “people” de la CMDB) y en todos los accesos a la base de datos: cada usuario o sistema que acceda a la CMDB deberá registrar su acceso y su desconexión.

Además**, los requisitos a cubrir por nuestro ejemplar** serán los ya identificados en el capítulo quinto y que resumimos a continuación:

- Servicio con Persistencia y en Diferido.
- Mensajes Sin Agrupación.
- Búsquedas Sin Criterio.
- Navegación Modo Normal, sin afectar a los mensajes.
- Navegación Siguiente Mensaje.
- Desencolado con Visibilidad Inmediata.
- Encolado con Visibilidad Inmediata.
- Mensajes Con Prioridad.
- Granularidad de Grano Medio.
- Suscriptores Múltiples.
- Esperas con valor. El mensaje espera a encolarse un tiempo predeterminado.
- Gestión General del Tiempo con Expiración y Retraso.

Para el desarrollo de este ejemplar hacemos uso del **mecanismo de Gestión Avanzada de Colas**. El funcionamiento, muy resumido, de este sistema es de la siguiente manera:

I. Se crean y configuran las Colas, que serán la que registren los Cambios.

II. Mediante un mecanismo de Disparadores, las operaciones seleccionadas sobre la CMDB registran el correspondiente mensaje en las Colas de notificaciones, en función de los requisitos seleccionados de granularidad de la información, gestión de tiempos, etc.

III. Mediante un mecanismo de escuchas, los subscriptores al servicio de notificación de cambios podrán recibir los mensajes de la Cola, con los requisitos que establezcamos de tiempos, prioridades, etc.

La Implementación, también muy resumida, de este Sistema **consiste en:**

1) **Configuración de la CMDB**. Se prepara la base de datos de forma que pueda trabajar con las colas de notificación.

2) **Creación y Configuración de las Colas**.

3) **Creación y Configuración de los Mensajes** que proporcionarán las Colas.

4) Creación del **Mecanismo de Disparadores**. Se crea un Disparador que complete el Mensaje con la información necesaria.

5) Creación del **Mecanismo de Escuchas.** Se implementa un Mecanismo que permanezca a la escucha para recibir e interpretar el Mensaje.

Las funcionalidades necesarias a desarrollar ya fueron expuestas en el capítulo quinto. A continuación mostraremos las funcionalidades desarrolladas para el ejemplar. Lo primero que desarrollaremos son las funcionalidades de **Creación y Configuración de Colas** (FCC y FCFC). Mostramos el código de ambas funcionalidades.

```
-- DESARROLLOS DE DOMINIOS DE BASES DE DATOS MEDIANTE EL USO DE UNA
-- METODOLOGIA ORIENTADA A EJEMPLARES. J.R.Coz.Fdez.
-- APLICACION AL DOMINIO DEL CONTROL DE CAMBIOS SOBRE UNA CMDB
-- Dpto. Ing. de SW y Sist. Informáticos. E.T.S.I. Informática, UNED
-- Sección de Código: Creación de Colas
CONNECT sys/<password>;
-- Instalación de Paquetes
@PUPBLD.SQL;/
@C:\orant\RDBMS80\ADMIN\DBMSAQAD.SQL;/
```

```
@C:\orant\RDBMS80\ADMIN\DBMSAQ.PLB;/
-- Llamada a Creación de Mensaje
@C:\Colas\MENSAJE_USUARIO_AQ;/
-- Llamada a Configuración de Colas
@C:\Colas\CONF_COLA.sql;/
CONNECT aq/<password>;


-- Sección de Código: Configuración de Colas
CONNECT sys/<password>;
EXECUTE DBMS_AQADM.CREATE_QUEUE_TABLE
        (queue_table => 'aq.MultiConsumerMsgs_qtab',
        multiple_consumers => TRUE,
        queue_payload_type => 'aq.Message_typ');
-- Opción Gestión del Tiempo con Retencion
-- El Valor de las Retenciones (100) se toma del DSL.
EXECUTE DBMS_AQADM.CREATE_QUEUE
        (queue_name => 'aq.msg_queue_multiple',
        queue_table => 'aq.MultiConsumerMsgs_qtab',
        retention_time => 100);
EXECUTE DBMS_AQADM.START_QUEUE
        (queue_name => 'aq.msg_queue_multiple');
-- Creación del Contenedor de Mensajes
CONNECT sys/<password>;
@C:\ORASW\CL817\RDBMS\ADMIN\CATnoque.sql;/
@C:\ORASW\CL817\RDBMS\ADMIN\CATqueue.sql;/
CREATE USER aq IDENTIFIED BY aq;
GRANT CONNECT, RESOURCE TO aq;
GRANT EXECUTE ON dbms_aq TO aq;
GRANT EXECUTE ON dbms_aqadm TO aq;
GRANT Aq_administrator_role TO aq;
CREATE type aq.Message_typ as object (message VARCHAR2(2000));
```

Posteriormente, desarrollamos las funcionalidades correspondientes a los **procedimientos de Encolado y Desencolado** (FE y FD):

```
-- Sección de Código: Encolado
CONNECT aq/<password>;
CREATE OR REPLACE PROCEDURE
P_ENCOLADO
(message_in IN VARCHAR2,expiracion IN number,retraso IN number,
prioridad IN NUMBER) as
        enqueue_options     dbms_aq.enqueue_options_t;
        dequeue_options     dbms_aq.dequeue_options_t;
        message_properties  dbms_aq.message_properties_t;
        recipients          dbms_aq.aq$_recipient_list_t;
        message_handle      RAW(16);
        message             aq.Message_typ;
no_messages      exception;
        pragma exception_init (no_messages, -25228);
BEGIN
message := Message_typ(message_in);
message_properties.delay := retraso;
message_properties.expiration := expiracion;
message_properties.priority := prioridad;
enqueue_options.visibility := dbms_aq.IMMEDIATE;
dbms_aq.enqueue(queue_name => 'aq.msg_queue_multiple',
enqueue_options    => enqueue_options,
message_properties => message_properties,
payload            => message,
msgid              => message_handle);
EXCEPTION
WHEN OTHERS THEN
message := Message_typ('NO');
END;

-- Sección de Código: Desencolado
CONNECT sys/<password>;
CREATE OR REPLACE PROCEDURE
P_DESENCOLADO
(user_in IN VARCHAR2,entity_out OUT VARCHAR2,espera IN NUMBER) as
        dequeue_options     dbms_aq.dequeue_options_t;
        message_properties  dbms_aq.message_properties_t;
        message_handle      RAW(16);
        message             aq.Message_typ;
        no_messages        exception;
        pragma exception_init (no_messages, -25228);
```

```
BEGIN
-- Esperas con Valor
dequeue_options.wait := espera;
dequeue_options.consumer_name := user_in;
-- En modo Navegación se ignora la visibilidad
-- Modo no por defecto (Borrado): Navegación, en modo normal
dequeue_options.dequeue_mode := dbms_aq.BROWSE;
dbms_aq.dequeue(queue_name => 'priority_msg_queue',
dequeue_options    => dequeue_options,
message_properties => message_properties,
payload            => message,
msgid              => message_handle);
entity_out := message.message;
COMMIT;
EXCEPTION
WHEN OTHERS THEN
entity_out := 'NO';
COMMIT;
END;
```

Una vez desarrollados los procedimientos de encolado y desencolado podemos desarrollar la funcionalidad de **creación de disparadores** (FCD), según los requisitos establecidos para nuestro ejemplar y detallados en la sección anterior:

```
-- Sección de Código: Creación de Disparadores
-- CONEXION
CREATE OR REPLACE TRIGGER CONEXION_ENCOLADO AFTER LOGON ON DATABASE
DECLARE
        PRAGMA AUTONOMOUS_TRANSACTION;
        USER_ID VARCHAR(100);
        DATABASE_ID VARCHAR(100);
        EXPIRACION NUMBER;
        RETRASO NUMBER;
        PRIORIDAD NUMBER;
BEGIN
        SELECT USER INTO USERID FROM DUAL;
   EXPIRACION = 6;
   RETRASO = 11;
   PRIORIDAD = 1;
```

```
        AQ.P_ENCOLADO('BASE DE DATOS: 'DATABASE_ID||' ESQUEMA:
NULL'||' USER: 'USERID||'TABLA: NULL'||' OPERACION:CONEXION A LA BASE
DE DATOS'  || ' PK1: NULL@NULL',EXPIRACION,RETRASO,PRIORIDAD);
END;
-- DESCONEXION
CREATE OR REPLACE TRIGGER DESCONEXION_ENCOLADO AFTER LOGOFF ON
DATABASE
DECLARE
        PRAGMA AUTONOMOUS_TRANSACTION;
        USER_ID VARCHAR(100);
        DATABASE_ID VARCHAR(100);
        EXPIRACION NUMBER;
        RETRASO NUMBER;
        PRIORIDAD NUMBER;
BEGIN
        SELECT USER INTO USERID FROM DUAL;
   EXPIRACION = 7;
   RETRASO = 12;
   PRIORIDAD = 2;
         AQ.P_ENCOLADO('BASE DE DATOS: 'DATABASE_ID||' ESQUEMA:
NULL'||' USER: 'USERID||'TABLA: NULL'||' OPERACION:DESCONEXION DE LA
BASE DE DATOS'  || ' PK1: NULL@NULL',EXPIRACION,RETRASO,PRIORIDAD);
END;
-- PEOPLE_ENCOLADO: Todas las operaciones serán notificadas
-- Grano Fino: necesitamos toda la información del registro que ha
-- cambiado
  CREATE OR REPLACE TRIGGER T_PEOPLE_ENCOLADO AFTER INSERT OR UPDATE
OR DELETE ON PEOPLE FOR EACH ROW
DECLARE
        PRAGMA AUTONOMOUS_TRANSACTION;
        USER_ID VARCHAR(100);
        DATABASE_ID VARCHAR(100);
        ESQUEMA_ID VARCHAR(100);
        EXPIRACION NUMBER;
        RETRASO NUMBER;
        PRIORIDAD NUMBER;
BEGIN
    SELECT USER INTO USERID FROM DUAL;
  IF INSERTING THEN
     EXPIRACION = 23; RETRASO = 30; PRIORIDAD = 16;
```

```
     AQ.P_ENCOLADO('BASE DE DATOS: 'DATABASE_ID||' ESQUEMA:
'ESQUEMA_ID||' USUARIO: 'USERID||'TABLA: PEOPLE'||' OPERACION:
INSERCION DE REGISTRO'  || ' PK1: User_ID@' ||
:new.User_ID,EXPIRACION,RETRASO,PRIORIDAD);
    END IF;
    IF UPDATING THEN
      EXPIRACION = 28; RETRASO = 35; PRIORIDAD = 21;
      AQ.P_ENCLADO('BASE DE DATOS: 'DATABASE_ID||' ESQUEMA:
'ESQUEMA_ID||' USUARIO: 'USERID||'TABLA: PEOPLE'||' OPERACION:
ACTUALIZACION DE REGISTRO'  || ' PK1: User_ID@' ||
:new.User_ID,EXPIRACION,RETRASO,PRIORIDAD);
    END IF;
    IF DELETING THEN
      EXPIRACION = 26; RETRASO = 33; PRIORIDAD = 19;
      AQ.P_ENCOLADO('BASE DE DATOS: 'DATABASE_ID||' ESQUEMA:
'ESQUEMA_ID||' USUARIO: 'USERID||'TABLA: PEOPLE'||' OPERACION: BORRADO
DE REGISTRO'  || ' PK1: User_ID@' ||
:old.User_ID,EXPIRACION,RETRASO,PRIORIDAD);
    END IF;
  END;
-- THIRD_COMPANIES_ENCOLADO: Todas las operaciones serán notificadas
-- Grano Fino: necesitamos toda la información del registro que ha
-- cambiado
  CREATE OR REPLACE TRIGGER T_THIRD_COMPANIES_ENCOLADO AFTER INSERT OR
UPDATE OR DELETE ON THIRD_COMPANIES FOR EACH ROW
DECLARE
        PRAGMA AUTONOMOUS_TRANSACTION;
        USER_ID VARCHAR(100);
        DATABASE_ID VARCHAR(100);
        ESQUEMA_ID VARCHAR(100);
        EXPIRACION NUMBER;
        RETRASO NUMBER;
        PRIORIDAD NUMBER;
BEGIN
    SELECT USER INTO USERID FROM DUAL;
    IF INSERTING THEN
      EXPIRACION = 24;RETRASO = 31;PRIORIDAD = 17;
      AQ.P_ENCOLADO('BASE DE DATOS: 'DATABASE_ID||' ESQUEMA:
'ESQUEMA_ID||' USUARIO: 'USERID||'TABLA: THIRD_COMPANIES'||'
OPERACION: INSERCION DE REGISTRO'  || ' PK1: Company_ID@' ||
:new.Company_ID,EXPIRACION,RETRASO,PRIORIDAD);
```

```
    END IF;
    IF UPDATING THEN
      EXPIRACION = 29; RETRASO = 36; PRIORIDAD = 22;
      AQ.P_ENCOLADO('BASE DE DATOS: 'DATABASE_ID||' ESQUEMA:
'ESQUEMA_ID||' USUARIO: 'USERID||'TABLA: THIRD_COMPANIES'||'
OPERACION: ACTUALIZACION DE REGISTRO'  || ' PK1: Company_ID@' ||
:new.Company_ID,EXPIRACION,RETRASO,PRIORIDAD);
    END IF;
    IF DELETING THEN
      EXPIRACION = 27;RETRASO = 34;PRIORIDAD = 20;
      AQ.P_ENCOLADO('BASE DE DATOS: 'DATABASE_ID||' ESQUEMA:
'ESQUEMA_ID||' USUARIO: 'USERID||'TABLA: THIRD_COMPANIES'||'
OPERACION: BORRADO DE REGISTRO'  || ' PK1: Company_ID@' ||
:old.Company_ID,EXPIRACION,RETRASO,PRIORIDAD);
    END IF;
  END;
-- CONTRACTS_ENCOLADO: Solo nuevas inserciones son notificadas
-- Grano Fino: necesitamos toda la información del registro que se ha
-- insertado
  CREATE OR REPLACE TRIGGER T_CONTRACTS_ENCOLADO AFTER INSERT OR
UPDATE OR DELETE ON CONTRACTS FOR EACH ROW
DECLARE
        PRAGMA AUTONOMOUS_TRANSACTION;
        USER_ID VARCHAR(100);
        DATABASE_ID VARCHAR(100);
        ESQUEMA_ID VARCHAR(100);
        EXPIRACION NUMBER;
        RETRASO NUMBER;
        PRIORIDAD NUMBER;
BEGIN
    SELECT USER INTO USERID FROM DUAL;
    IF INSERTING THEN
      EXPIRACION = 25;RETRASO = 32;PRIORIDAD = 18;
      AQ.P_ENCOLADO('BASE DE DATOS: 'DATABASE_ID||' ESQUEMA:
'ESQUEMA_ID||' USUARIO: 'USERID||'TABLA: CONTRACTS'||' OPERACION:
INSERCION DE REGISTRO'  || ' PK1: Contract_ID@' || :new.Contract_ID ||
' PK2: Company_ID@' || :new.Company_ID || ' PK2: Machine_ID@' ||
:new.Machine_ID,EXPIRACION,RETRASO,PRIORIDAD);
    END IF;
  END;
```

Hemos establecido, para nuestro ejemplar, una serie de prioridades para cada cambio y unos retrasos y expiraciones para los correspondientes mensajes a notificar. Como analizaremos posteriormente, estos datos formarán parte de nuestro Lenguaje Específico del Dominio (DSL).

Desarrollamos, a continuación, las funcionalidades de **Inicio y Fin de Operaciones Masivas** (FIOM y FFOM) que nos permitirán notificar a los dos subscriptores (el sistema financiero y el ERP) que se está produciendo una modificación masiva en la CMDB.

```
-- Sección de Código: Inicio de Operaciones Masivas
CONNECT aq/<password>;
CREATE OR REPLACE PROCEDURE P_MASIVO_COMIENZO
DECLARE
        PRAGMA AUTONOMOUS_TRANSACTION;
        USER_ID VARCHAR(100);
        DATABASE_ID VARCHAR(100);
        ESQUEMA_ID VARCHAR(100);
        EXPIRACION NUMBER;
        RETRASO NUMBER;
        PRIORIDAD NUMBBER;
BEGIN
  SELECT USER INTO USER_ID FROM DUAL;
  DELETE from aq.MultiConsumerMsgs_qtab where state = 3;
     -- La expiración, retraso y prioridad se obtienen desde el DSL
   EXPIRACION = 10;
   RETRASO = 15;
   PRIORIDAD = 5;
  AQ.P_ENCOLADO('BASE DE DATOS: 'DATABASE_ID||' ESQUEMA:
'ESQUEMA_ID||' USER: 'USER_ID||'TABLA: VARIAS'||' OPERACION: COMIENZO
DE OPERACIONES MASIVAS'  || ' PK1:
NULL@NULL',EXPIRACION,RETRASO,PRIORIDAD);
  dbms_aqadm.stop_queue (queue_name => 'aq.msg_queue_multiple');
END;
```

```
-- Sección de Código: Fin de Operaciones Masivas
CONNECT aq/<password>;
CREATE OR REPLACE PROCEDURE P_MASIVO_FINAL
DECLARE
        PRAGMA AUTONOMOUS_TRANSACTION;
        USER_ID VARCHAR(100);
        DATABASE_ID VARCHAR(100);
        ESQUEMA_ID VARCHAR(100);
        EXPIRACION NUMBER;
        RETRASO NUMBER;
        PRIORIDAD NUMBBER;
BEGIN
  SELECT USER INTO USER_ID FROM DUAL;
  dbms_aqadm.start_queue (queue_name => 'aq.msg_queue_multiple');
   -- La expiracion, retraso y prioridad se obtienen desde el DSL
   EXPIRACION = 10;
   RETRASO = 15;
   PRIORIDAD = 5;
  -- Los datos del Mensaje a encolar se tomarán de un DSL: Base de
datos y esquema
  AQ.P_ENCOLADO('BASE DE DATOS: 'DATABASE_ID||' ESQUEMA:
'ESQUEMA_ID||' USER: 'USER_ID||'TABLA: VARIAS'||' OPERACION: FIN DE
OPERACIONES MASIVAS'  || ' PK1:
NULL@NULL',EXPIRACION,RETRASO,PRIORIDAD);
END;
```

Nos interesará, también, desarrollar la **habilitación, deshabilitación y borrado de disparadores** (FHD, FDD y FBD), para poder, en cualquier momento, deshabilitar las notificaciones y volverlas a poder habilitar. Además, incluiremos el **borrado de las operaciones masivas** (FBOM). Nos parece importante desarrollar también las funcionalidades de borrado para que ante cualquier actualización o cambio general de nuestra CMDB podamos reiniciar la misma a su estado inicial.

```
-- Sección de Código: Habilitación de Disparadores
CONNECT sys/<password>;
ALTER TRIGGER ARRANQUE_ENCOLADO ENABLE;/
ALTER DROP TRIGGER APAGADO_ENCOLADO ENABLE;/
ALTER DROP TRIGGER CONEXION_ENCOLADO ENABLE;/
```

```
ALTER DROP TRIGGER DESCONEXION_ENCOLADO ENABLE;/
ALTER DROP TRIGGER CONEXION_FALLIDA_ENCOLADO ENABLE;/
-- Entities With Notifications
ALTER DROP TRIGGER T_PEOPLE_ENCOLADO ENABLE;/
ALTER DROP TRIGGER T_THIRD_COMPANIES_ENCOLADO ENABLE;/
ALTER DROP TRIGGER T_THIRD_CONTRACTS_ENCOLADO ENABLE;/


-- Sección de Código: Deshabilitación de Disparadores
CONNECT sys/<password>;
ALTER TRIGGER ARRANQUE_ENCOLADO DISABLE;/
ALTER DROP TRIGGER APAGADO_ENCOLADO DISABLE;/
ALTER DROP TRIGGER CONEXION_ENCOLADO DISABLE;/
ALTER DROP TRIGGER DESCONEXION_ENCOLADO DISABLE;/
ALTER DROP TRIGGER CONEXION_FALLIDA_ENCOLADO DISABLE;/
-- Entities With Notifications
ALTER DROP TRIGGER T_PEOPLE_ENCOLADO DISABLE;/
ALTER DROP TRIGGER T_THIRD_COMPANIES_ENCOLADO DISABLE;/
ALTER DROP TRIGGER T_THIRD_CONTRACTS_ENCOLADO DISABLE;/


-- Sección de Código: Borrado de Disparadores
CONNECT sys/<password>;
DROP TRIGGER ARRANQUE_ENCOLADO;/
DROP TRIGGER APAGADO_ENCOLADO;/
DROP TRIGGER CONEXION_ENCOLADO;/
DROP TRIGGER DESCONEXION_ENCOLADO;/
DROP TRIGGER CONEXION_FALLIDA_ENCOLADO;/
-- Entities With Notifications
DROP TRIGGER T_PEOPLE_ENCOLADO;/
DROP TRIGGER T_THIRD_COMPANIES_ENCOLADO;
DROP TRIGGER T_THIRD_CONTRACTS_ENCOLADO;/

-- Sección de Código: Borrado de Operaciones Masivas
CONNECT aq/<password>;
DROP PROCEDURE P_MASIVO_FINAL;/
DROP PROCEDURE P_MASIVO_COMIENZO FINAL;/
```

Por último, desarrollamos las **funcionalidades relacionadas con los subscriptores** (FCS) **y los permisos** (FAP):

```
-- Sección de Código: Creación de Subscriptores
CONNECT aq/<password>;
DECLARE
subscriber sys.aq$_agent;
BEGIN
-- USUARIO: SYSTEM.
subscriber := sys.aq$_agent('SYSTEM', NULL, NULL);
dbms_aqadm.add_subscriber(queue_name =>
'priority_msg_queue',subscriber => subscriber);
-- USUARIO: AQ.
subscriber := sys.aq$_agent('AQ', NULL, NULL);
dbms_aqadm.add_subscriber(queue_name =>
'priority_msg_queue',subscriber => subscriber);
-- Sistema Financiero
subscriber := sys.aq$_agent('Financial_Sys', NULL, NULL);
dbms_aqadm.add_subscriber(queue_name =>
'priority_msg_queue',subscriber => subscriber);
-- Sistema ERP
subscriber := sys.aq$_agent('ERP_Sys', NULL, NULL);
dbms_aqadm.add_subscriber(queue_name =>
'priority_msg_queue',subscriber => subscriber);
END;

-- Sección de Código: Asignación de Permisos
-- Nota: usuarios de Sistema.
CONNECT sys/<password>;
GRANT EXECUTE ON P_DESENCOLADO TO SYSTEM;
GRANT EXECUTE ON P_ENCOLADO TO SYSTEM;
GRANT EXECUTE ON P_MASIVO_COMIENZO TO SYSTEM;
GRANT EXECUTE ON P_MASIVO_FINAL TO SYSTEM;
GRANT EXECUTE ON P_MASIVO_COMIENZO TO AQ;
GRANT EXECUTE ON P_MASIVO_FINAL TO AQ;
EXECUTE
dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','SYSTEM',FALSE);
EXECUTE
dbms_aqadm.grant_system_privilege('DEQUEUE_ANY','SYSTEM',FALSE);
GRANT EXECUTE ON P_DESENCOLADO TO AQ;
GRANT EXECUTE ON P_ENCOLADO TO AQ;
CONNECT aq/<password>;
EXECUTE
dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','SYSTEM',FALSE);
```

```
EXECUTE
dbms_aqadm.grant_system_privilege('DEQUEUE_ANY','SYSTEM',FALSE);
CONNECT sys/<password>;
-- Subscriptores al Servicio: Sistema Financiero y ERP
GRANT EXECUTE ON P_DESENCOLADO TO Financial_Sys;
GRANT EXECUTE ON P_ENCOLADO TO Financial_Sys;
GRANT EXECUTE ON P_DESENCOLADO TO ERP_Sys;
GRANT EXECUTE ON P_ENCOLADO TO ERP_Sys;
GRANT EXECUTE ON P_MASIVO_COMIENZO TO Financial_Sys;
GRANT EXECUTE ON P_MASIVO_FINAL TO Financial_Sys;
GRANT EXECUTE ON P_MASIVO_COMIENZO TO ERP_Sys;
GRANT EXECUTE ON P_MASIVO_FINAL TO ERP_Sys;
-- Los subscriptores tienen su propio usuario de acceso a la CMDB
CONNECT Financial_Sys/<password>;
EXECUTE
dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','SYSTEM',FALSE);
EXECUTE
dbms_aqadm.grant_system_privilege('DEQUEUE_ANY','SYSTEM',FALSE);
-- Los subscriptores tienen su propio usuario de acceso a la CMDB
CONNECT ERP_Sys/<password>;
EXECUTE
dbms_aqadm.grant_system_privilege('ENQUEUE_ANY','SYSTEM',FALSE);
EXECUTE
dbms_aqadm.grant_system_privilege('DEQUEUE_ANY','SYSTEM',FALSE);
```

Podemos incluir, como última funcionalidad, la **eliminación de la Cola** (FEC).

```
-- Sección de Código: Eliminación de Colas
CONNECT aq/<password>;
-- Borrado Cola
EXECUTE dbms_aqadm.stop_queue (queue_name => 'msggroup_queue');
EXECUTE dbms_aqadm.drop_queue (queue_name => 'msggroup_queue');
EXECUTE dbms_aqadm.drop_queue_table (queue_table => 'aq.msggroup');
-- Borrado Mensaje
drop type aq.Message_typ; CONNECT sys/<password>;
-- Borrado Usuario AQ
drop user aq;/
-- Borrado Procedimientos de Encolado y Desencolado
DROP PROCEDURE P_ENCOLADO;/ DROP PROCEDURE P_DESENCOLADO;/
```

8.2.Obtención de los Productos

Ya tenemos desarrollado un ejemplar de nuestro dominio. Estamos ya en condiciones de continuar con la fase de obtención de los productos, la última fase de la metodología EODAM.

Esta fase tiene tres actividades: **el análisis y modelado completo del dominio**, que ya hemos realizado, **el desarrollo de un Lenguaje Específico del Dominio (DSL)**, que nos servirá para parametrizar nuestros productos, **y el desarrollo de los generadores** que, desde el ejemplar y nuestro DSL, obtendrán el resto de productos de nuestro dominio.

Para el desarrollo de los generadores y de nuestro DSL haremos uso del lenguaje Exemplar Flexibilitation Language (EFL) [Heradio et al. 2008], que analizamos a continuación.

8.2.1. Sobre el lenguaje EFL

EFL (Exemplar Flexibilization Language) [Heradio et al. 2008] es un lenguaje de transformaciones creado para dar soporte a la metodología EDD [Heradio 2007] y **será utilizado como soporte a la metodología EODAM.**

La **implementación actual de EFL** corresponde a un lenguaje específico del dominio **embebido en Ruby**. Ruby es un lenguaje interpretado y con tipado dinámico que posee una sintaxis flexible y concisa, además es un lenguaje apropiado para la construcción de transformadores porque es totalmente Orientado a Objetos (OO), su sintaxis es sorprendentemente concisa y tiene una gran capacidad para el tratamiento de textos y ficheros.

De hecho, incorpora las expresiones regulares [Friedl 2002] como tipo básico del lenguaje. Ofrece potentes contenedores (listas y tablas) e iteradores, y existen implementaciones del intérprete de Ruby para distintos sistemas operativos, lo que asegura un alto grado de portabilidad.

Además, Ruby [Ruby 2010] dispone de la capacidad de extensión suficiente para encapsular ciertos patrones de diseño (ofreciendo una interfaz que sólo expone los parámetros variables y ocultando la codificación de la parte permanente del patrón y de la gestión de la parte variable). A modo de ejemplo, en [Thomas and Hunt 2001] aparecen encapsulados los patrones *visitor, delegate, observer* y *singleton*. Existen otras referencias como [Fowler 2005] y [Eckel 2003] donde se indica cómo utilizar Ruby para construir transformadores.

EFL es un lenguaje simple, fácil de usar y con una gran potencia expresiva. **La unidad principal del lenguaje es el Generador.** Un Generador toma como entrada uno o varios archivos que forman parte del ejemplar, y una especificación que expresa la variabilidad deseada del nuevo producto.

Un Generador está formado por un conjunto de **reglas de Sustitución**. Una regla de Sustitución es la forma de indicar el tipo de cambio que se pretende realizar sobre el ejemplar. Estas reglas pueden ser sencillas o complejas, dependiendo del tipo de cambio. Las reglas de Sustitución se asocian a los ficheros del ejemplar a través de las **Producciones.**

Las Producciones se utilizan para agrupar un conjunto de Sustituciones que afectan a un fichero del ejemplar. La plataforma permite **detectar colisiones** potenciales, esto es, incoherencias debidas a que se aplican Sustituciones sobre una misma región del archivo del ejemplar. Actualmente, las reglas de Sustitución se expresan utilizando expresiones regulares. Por último, EFL define una colección de **operadores** (con diferente semántica) que se utilizan para combinar Generadores:

- la operación de secuenciación para indicar el orden de ejecución de los Generadores;

- la operación de suma para combinar las Producciones y Sustituciones de los Generadores;

- y la operación de superposición para actualizar las Sustituciones y Producciones de un Generador con las de otro Generador.

Como ya hemos mencionado, en la versión actual de EFL las Sustituciones están implementadas utilizando expresiones regulares. El motor de expresiones regulares trabaja sobre el fichero de entrada buscando los patrones especificados mediante la expresión regular. El fichero de entrada, desde el punto de vista del motor de expresiones regulares, es visto como un fichero plano formado por un conjunto de caracteres.

Cuando se utilizan las expresiones regulares para localizar el cambio, no se necesita conocer cuál es la estructura del fichero de entrada, ya que todo él se percibe de la misma forma. Este hecho es importante, ya que el programador no necesita conocer o disponer de más información estructural, y por tanto su trabajo en la elaboración de las transformaciones es más sencillo.

Para el desarrollador es simple especificar el cambio que se quiere, ya que el lenguaje de las expresiones regulares es un lenguaje específico del dominio ampliamente utilizado; por ejemplo, en el mundo Unix se utiliza en las herramientas SED (Stream Editor) [SED 2007] y GREP (Global Regular Expression Print) [GREP 2009]. Utilizaremos las implementaciones de EFL disponibles para la construcción de nuestros generadores [EFL 2007].

8.2.2. Configuración de Productos

Como se analizará en el próximo apartado *"Framework de Desarrollo"*, para la construcción de los productos del dominio y del resto de funcionalidades necesarias se ha utilizado el lenguaje de propósito general Ruby.

Como ya hemos mencionado, utilizaremos EFL para el desarrollo de los generadores, y su implementación en Ruby. Además, Ruby es un lenguaje apto para el desarrollo de DSL's embebidos, tal y como se indica en [Heradio 2007].

El lenguaje Ruby dispone de un elemento denominado HASH, un array asociativo que contiene elementos que se pueden acceder, no a través de índices numéricos secuenciales, sino a través de claves que pueden tener cualquier tipo de valor. Los hashes se pueden crear mediante pares de elementos dentro de llaves ({ }). Se usa la clave para encontrar algo en un hash de la misma forma que se utiliza el índice para encontrar algo en un ARRAY**. La sintaxis concreta de nuestro DSL será especificada en formato HASH.**

La semántica se construirá mediante las transformaciones programadas en Ruby y que hacen uso de EFL. A continuación, se muestra un ejemplo de DSL para nuestro dominio:

```
'AREA' => 'SPD',
'AGRUPACION_MENSAJES' => 'MSA',
'BUSQUEDAS' => 'BCC',
'NAVEGACION' => 'NSM',
'VISIBILIDAD'=> {
        'Encolado' => 'EVI',
        'Desencolado' => 'DVI'
},
```

```
'PRIORIDAD' => 'MCP',
'GRANULARIDAD' => 'GF',
'SUBSCRIPTORES' => 'SM',
'ESPERAS' => 'DECN'
'GESTION_TIEMPO' => {
        'Retenciones' => 'GTCR',
        'General' => 'GTER'
}
```

Este DSL representa los requisitos seleccionados para nuestro ejemplar del dominio y lo identificaremos como el DSL de los requisitos. Es decir, se tratará del **DSL de requisitos del Ejemplar del Dominio.**

Si, por ejemplo, quisieramos Configurar nuestro Producto con otros requisitos, como por ejemplo, Búsquedas Sin Criterio, Encolado y Desencolado con Visbilidad Transaccional, Granularidad de Grano Medio, Mensajes Sin Prioridad, Gestión del Tiempo Sencilla y con un solo subscriptor, nuestro DSL sería:

```
'AREA' => 'SPD',
'AGRUPACION_MENSAJES' => 'MSA',
'BUSQUEDAS' => 'BSC',
'NAVEGACION' => 'NSM',
'VISIBILIDAD'=> {
        'Encolado' => 'EVT',
        'Desencolado' => 'DVT'
},
'PRIORIDAD' => 'MSP',
'GRANULARIDAD' => 'GM',
'SUBSCRIPTORES' => 'SU',
'ESPERAS' => 'DECN'
'GESTION_TIEMPO' => {
        'Retenciones' => 'GTCR',
        'General' => 'GTS'}
```

Para ampliar el DSL de nuestro ejemplar deberemos incluir el DSL de funcionalidades, donde determinaremos que funcionalidades opcionales van a ser desarrolladas. Obviamente, no se incluirán las funcionalidades obligatorias. A continuación, se muestra el **DSL de funcionalidades del Ejemplar del Dominio:**

```
'HABILITACION_DISP' => true,
'DESHABILITACION_DISP' => true,
'BORRADO_DISP' => true,
'OPER_MASIVAS' => true,
'BORRADO_MASIVAS' => true
```

Esto significará que desarrollaremos todas las funcionalidades optativas para nuestro producto. Para completar el DSL debemos añadir los elementos. A continuación, se muestra el **DSL de elementos del Ejemplar del Dominio:**

```
'BRET' => '100',
'BOLON' => true,
'BOLOFF' => true,
'BOS' => false,
'BOC' => false,
'BPV' => [
        ['BOLON'=> '1'],
        ['BOLOFF'=> '2'],
        ['BOS'=> ''],
        ['BOC'=> ''],
        ['BOM'=> '5']],
'BEXP' =>[
        ['BOLON'=> '6'],
        ['BOLOFF'=> '7'],
        ['BOS'=> ''],
        ['BOC'=> ''],
        ['BOM'=> '10']],
'BERET' =>[
        ['BOLON'=> '11'],
        ['BOLOFF'=> '12'],
        ['BOS'=> ''],
```

```
            ['BOC'=> ''],
            ['BOM'=> '15']],
    'TID'=>['PEOPLE', 'THIRD_COMPANIES', 'CONTRACTS'],
    'TPV' => [
            ['Inserciones'=> ['16','17','18']],
            ['Borrados'=> ['19','20','']],
            ['Actualizaciones'=> ['21','22','']]],
    'TEXP' => [
            ['Inserciones'=> ['23','24','25']],
            ['Borrados'=> ['26','27','']],
            ['Actualizaciones'=> ['28','29','']]],
    'TRET' => [
            ['Inserciones'=> ['30','31','32']],
            ['Borrados'=> ['33','34','']],
            ['Actualizaciones'=> ['35','36','']]],
    'UID' => ['Financial_Sys', 'ERP_Sys']
```

Como podemos observar, se han incluido las entidades identificadas para nuestro ejemplar, los subscriptores del dominio y se han establecido parámetros sobre las prioridades, expiraciones y retrasos para cada una de las operaciones. Además, se han incluido las operaciones de conexión y desconexión de los usuarios, pero no se han incluido las operaciones de arranque y parada de la base de datos (BOS y BOC). Si observamos el código de nuestro ejemplar expuesto en las secciones anteriores, comprobaremos la correspondencia entre el DSL y el código desarrollado.

8.2.3. El Framework de desarrollo

Siguiendo la metodología EODAM, una vez realizado el análisis completo del dominio y construido nuestro DSL, estamos en condiciones de desarrollar los Generadores del dominio. Los Generadores, desde nuestro DSL y un Ejemplar del Dominio, obtendrán el resto de productos de la línea. **Con el objetivo de integrar la fase de desarrollo, análisis y diseño se construye un Framework de desarrollo**, cuyo funcionamiento resumido se muestra en la figura siguiente.

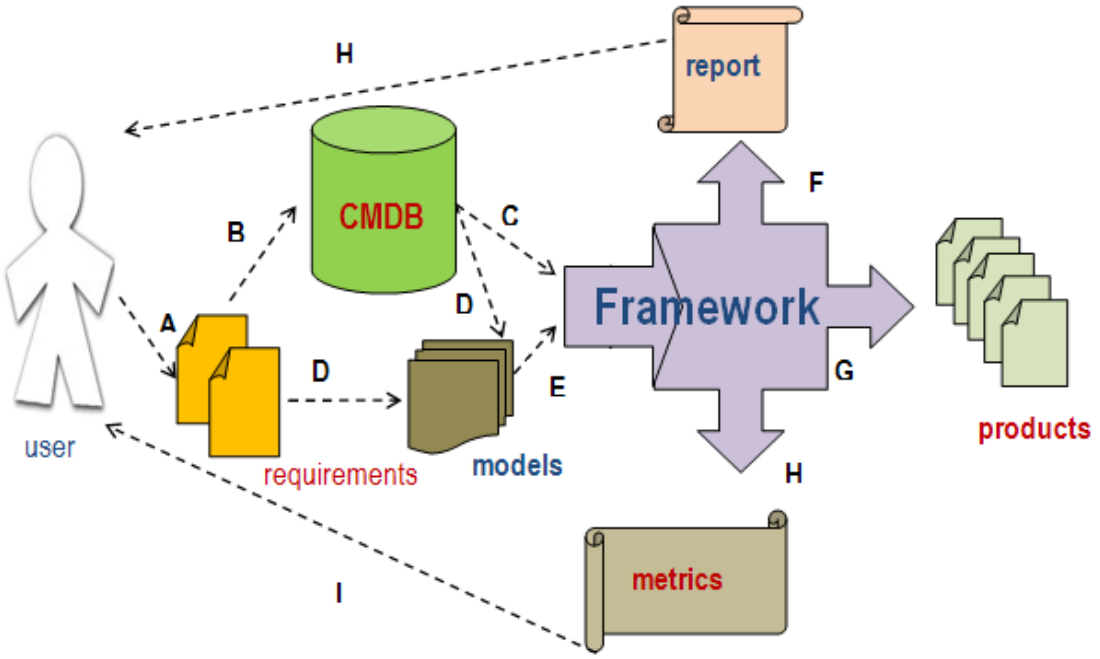


Figura 8-1 Framework de Desarrollo

A) En primer lugar, los usuarios establecen los requisitos del sistema.

B) Según los requisitos establecidos, se construye la CMDB.

C) La CMDB es una entrada para el Framework de Desarrollo (FD).

D) Desde los requisitos y la CMDB se construyen los modelos del dominio (el modelo ampliado, el integrado, el de elementos, el completo, etc.)

E) Los modelos constituyen una entrada para el FD.

F) El FD, con los datos de esquema de la CMDB y los modelos, los transforma a un DSL mediante un dispositivo denominado ***"transformador DSL"*** que se detallará a continuación. Además, mediante otro dispositivo denominado ***"detector de inconsistencias"***, el FD generará un informe para los usuarios con las inconsistencias detectadas, como se detallará a continuación.

G) El FD, desde el DSL y un ejemplar desarrollado, que hace uso también del propio FD, genera el resto de productos mediante los ***"generadores".***

H) El informe generado por el FD llega a los usuarios en un formato amigable.

I) El FD obtiene todas las métricas del dominio, que pueden ser consultadas por los usuarios, mediante las ***"herramientas de soporte a NFTE"***, ya analizadas en el capítulo dos.

8.2.3.1. Transformador DSL

El Transformador DSL (TDSL) es una parte del Framework de Desarrollo (FD) que da soporte a la metodología EODAM, donde además se incluyen el *Detector de Inconsistencias*, los *Generadores* y las *herramientas de soporte a NFTE*.

El TDSL parte de los modelos que incluyen la información sobre los requisitos y de la CMDB para obtener y verificar el DSL final, que actúa como entrada a los generadores para obtener el resto de productos del dominio.

Parte de la información del DSL debe de ser diseñada por el rol que establezca los requisitos y parte de la misma está intrínsecamente relacionada con la estructura interna de la propia Base de Datos, en nuestro caso de la CMDB.

Por ejemplo, en el caso de la selección de las prioridades de cada una de las entidades de la CMDB, la decisión sobre que prioridades tiene cada una de ellas la realizará el rol que decida este requisito. De la misma forma, en el caso de identificar a los subscriptores de las notificaciones, deberá de determinarlo el correspondiente rol. Existe, también, otra información de carácter interno a la base de datos, como el nº de tablas, de columnas de cada tabla, de las claves, del nombre de los usuarios, de la base de datos, etc., que es necesario considerar a la hora de desarrollar los productos. Esta información está contenida, en parte, en el DSL.

La meta información sobre la base de datos (o CMDB) objetivo está contenida en la propia base de datos, que dispone de meta tablas que almacenan toda esta información. Se considera esencial realizar el análisis de esta información de forma automática, de tal forma que, por ejemplo, si se modifica el nombre de una columna de una tabla, esta información sea modificada en la flexibilización de los ejemplares y se generen los productos correctos con información actualizada.

A continuación se muestra, en resumen, el funcionamiento del TDSL:

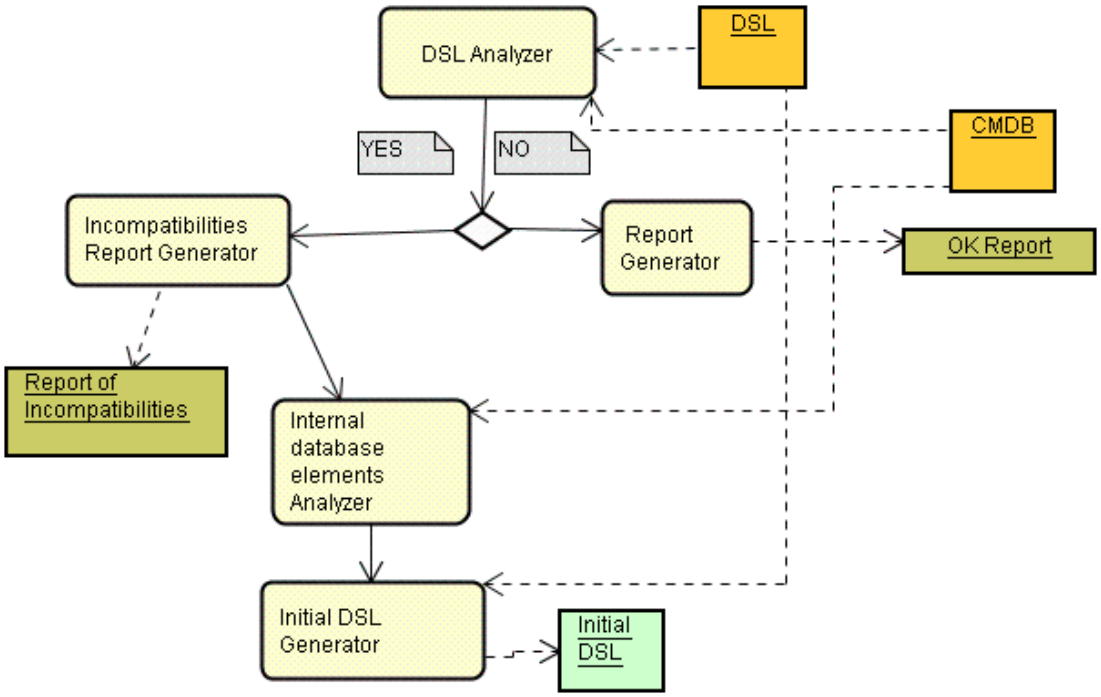


Figura 8-2 Transformador DSL

Por un lado, el transformador DSL toma el DSL de entrada y mediante un analizador (*DSL Analyzer*) comprueba su consistencia con la base de datos objetivo. Para ello, se conecta con la CMDB y obtiene la meta información necesaria.

En caso de que no haya inconsistencias se genera un informe (*"Ok Report"*) que le sirve al usuario para saber que no ha habido cambios y que el DSL es correcto. Si, por ejemplo, el DSL de elementos incluye una entidad que ya no existe en la CMDB o un usuario que tampoco existe o ha cambiado de nombre, el TDSL generará un informe de error para el usuario con el soporte de un modulo denominado *"Incompatibilities Report Generator"*.

En caso de que el DSL de entrada no sea correcto, el transformador DSL, a través de un modulo denominado "*Internal Database Elements Analyzer*", obtendrá toda la información necesaria de los requisitos y la estructura de la CMDB y mediante otro

módulo denominado *"Initial DSL Generator"* nos generará un DSL sin rellenar para facilitar al usuario la labor de especificación de requisitos de forma amigable. Parte de esta información debe de ser diseñada por el rol que establezca los requisitos y parte de la misma está intrínsecamente relacionada con la estructura interna de la propia Base de Datos. A continuación, se muestra un ejemplo de DSL inicial:

```
'BRET' => 'XXX',
'BOLON' => true,
'BOLOFF' => true,
'BOS' => false,
'BOC' => false,
'BPV' => [
        ['BOLON'=> 'X'],
        ['BOLOFF'=> 'X'],
        ['BOS'=> ''],
        ['BOC'=> ''],
        ['BOM'=> 'X']
        ],
'BEXP' =>[
        ['BOLON'=> 'X'],
        ['BOLOFF'=> 'X'],
        ['BOS'=> ''],
        ['BOC'=> ''],
        ['BOM'=> 'X']
         ],
'BERET' =>[
        ['BOLON'=> 'X'],
        ['BOLOFF'=> 'X'],
        ['BOS'=> ''],
        ['BOC'=> ''],
        ['BOM'=> 'X']
         ],
'TID'=>['PEOPLE', 'THIRD_COMPANIES', 'CONTRACTS'],
'TPV' => [
        ['Inserciones'=> ['X','X','X']],
        ['Borrados'=> ['X','X','X']],
        ['Actualizaciones'=> ['X','X','X']],
        ],
'TEXP' => [
```

```
            ['Inserciones'=> ['X','X','X']],
            ['Borrados'=> ['X','X','X']],
            ['Actualizaciones'=> ['X','X','X']]
            ],
        'TRET' => [
            ['Inserciones'=> ['X','X','X']],
            ['Borrados'=> ['X','X','X']],
            ['Actualizaciones'=> ['X','X','X']]
            ],
        'UID' => ['Financial_Sys', 'ERP_Sys']
```

Todos los módulos de esta sección se han desarrollado utilizando Ruby, para facilitar la integración de todos los módulos y evitar trabajar con múltiples entornos de desarrollo.

8.2.3.2.Detector de Inconsistencias

Previamente al desarrollo de cualquier producto, es necesario detectar, a partir del DSL, las inconsistencias entre los requisitos seleccionados. De lo contrario, si el DSL es erróneo, podría producirse la generación de productos incorrectos. Este es el objetivo del *Dectector de Inconsistencias*, detectar todas las combinaciones no permitidas de requisitos.

A continuación se muestra, en resumen, el funcionamiento de este módulo.

Desde el DSL, mediante un analizador, en el caso de que se detecte alguna inconsistencia se generará un informe de error para el usuario con el soporte de un modulo denominado *"Incompatibilities Report Generator"* y en el caso de que no haya inconsistencias se generará un informe (*"Ok Report"*), que le sirve al usuario para saber que el DSL es correcto y, por tanto, pueden ser generados los productos desde los generadores.

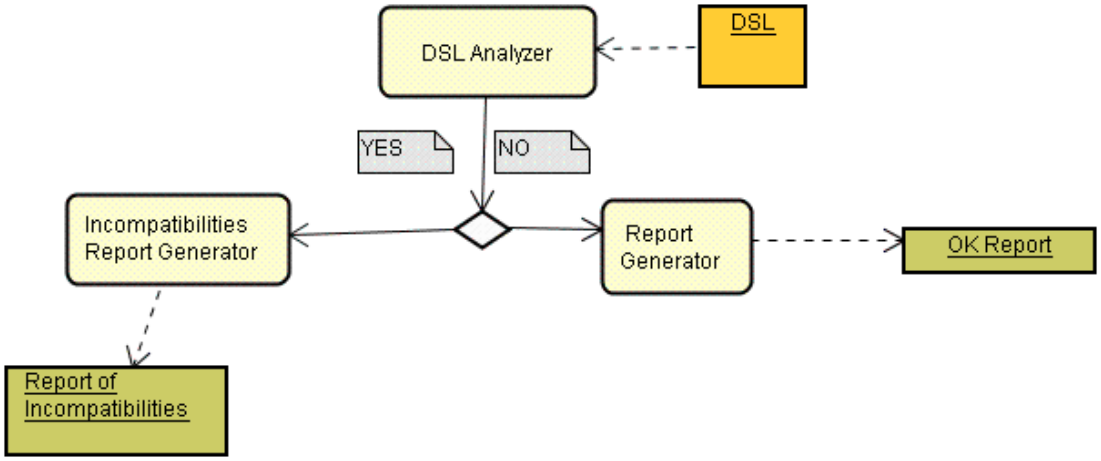


Figura 8-3 Detector de Inconsistencias

8.2.3.3.Generadores

El componente aún no descrito del Framework son los Generadores. Los Generadores obtienen desde el Ejemplar del Dominio y el DSL el resto de Productos del dominio. Los Generadores hacen uso del lenguaje EFL. Este componente **es un compilador como el de la figura siguiente, compuesto por un analizador y un generador.** El analizador crea una representación interna del código fuente DSL y se la pasa al generador, que obtiene el producto final correspondiente.

EFL facilita la escritura del generador, que en realidad es un nuevo compilador (subcompilador) que transforma el ejemplar en el producto especificado en el código DSL. El subcompilador se compone de un analizador (subanalizador) que crea una representación interna del ejemplar y de uno o varios generadores que actúan de forma coordinada para modificar el ejemplar.

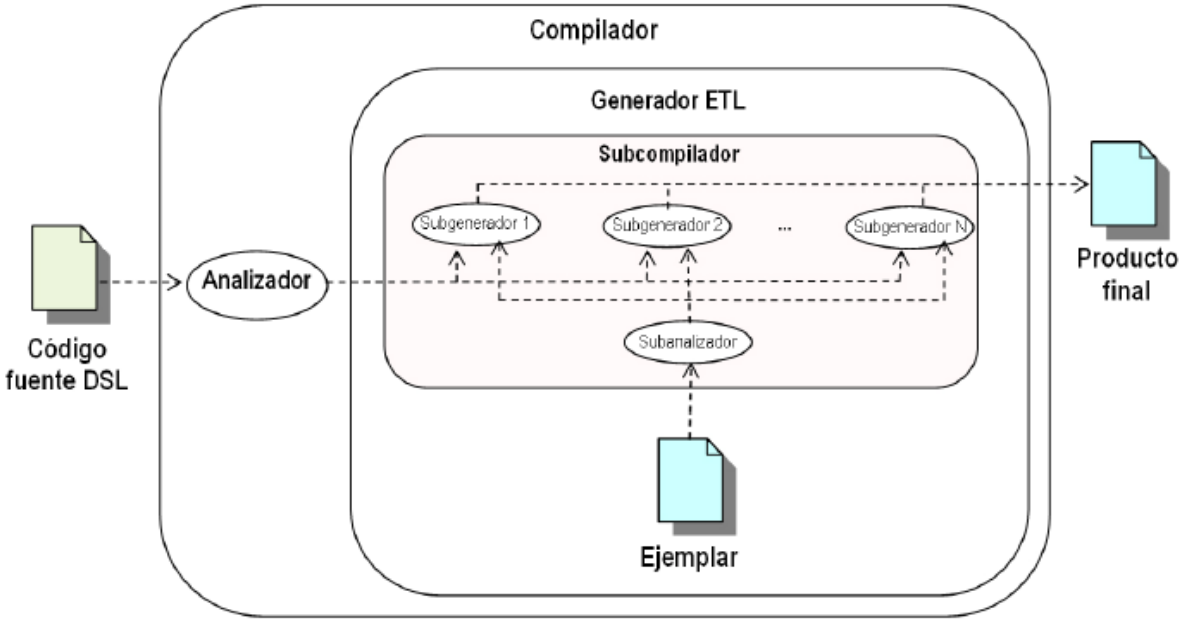


Figura 8-4 Generadores en EFL.

Para desarrollar un generador EFL, se dispone de las siguientes primitivas:

1. **Sustitución**. Expresa el intercambio de un patrón de código del ejemplar por nuevo código. Cuando la implementación del ejemplar sea modular y esté repartida entre varios ficheros, con frecuencia será necesario expresar una misma sustitución sobre más de un fichero. Para facilitar la reutilización de las sustituciones y evitar así su repetición, las sustituciones son independientes de los ficheros del ejemplar y de los ficheros del producto final. Una sustitución se liga a los ficheros del ejemplar y del producto final mediante la primitiva de "producción", que se explica a continuación.

2. **Producción**. Expresa la aplicación simultánea de un conjunto de sustituciones sobre un fichero del ejemplar para obtener un fichero del producto final. En el caso de que el conjunto sea vacío, la producción se limitará a copiar el fichero del ejemplar en el fichero del producto final. Puede suceder que varias de las sustituciones de una producción se solapen si indican cambios distintos sobre el mismo fragmento de un fichero del ejemplar. Para prevenir errores en el programa objeto, las colisiones entre sustituciones se detectarán automáticamente.

3. **Generación**. Expresa la ejecución simultánea de un conjunto de producciones. Esta primitiva facilita la detección automática de colisiones entre las sustituciones de distintas producciones.

Además, en nuestro caso, desarrollamos un generador para cada funcionalidad-tipo de nuestro dominio, tal y como se muestra en la figura siguiente.

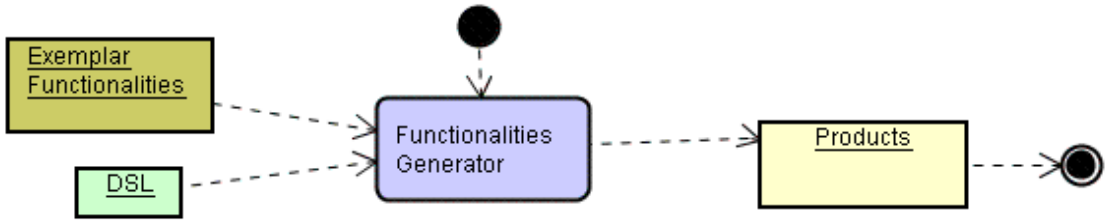


Figura 8-5 Generadores de Funcionalidades.

Cada *"Funcionality Generator"* obtiene las funcionalidades de cada producto final, desde la funcionalidad del ejemplar y el DSL. Las funcionalidades desarrolladas se adjuntan, a continuación, en la tabla siguiente, donde se identifica cada generador con un código, con propósitos de gestión de la configuración.

GENERADORES DE FUNCIONALIDADES	
FUNCIONALIDAD	**CODIGO**
Creación de Disparadores	G-FCD
Habilitación de Disparadores	G-FHD
Deshabilitación de Disparadores	G-FDD
Borrado de Disparadores	G-FBD
Inicio de Operaciones Masivas	G-FIOM
Fin de Operaciones Masivas	G-FFOM
Borrado de Operaciones Masivas	G-FBOM
Creación de Colas	G-FCC
Configuración de Colas	G-FCFC
Encolado	G-FE
Desencolado	G-FD
Eliminación de las Colas	G-FEC
Creación de Suscriptores	G-FCS
Asignación de Permisos	G-FAP

Tabla 8-1 Generadores de Funcionalidades desarrollados.

9. Construcción de un Framework de pruebas

"Da igual.
Prueba otra vez.
Fracasa otra vez.
Fracasa mejor."
Samuel Beckett.
.

Una vez obtenidos los productos de nuestro dominio, el siguiente paso será realizar las pruebas necesarias para validar nuestros productos. El objeto de este capítulo es detallar las pruebas que se han desarrollado.

En primer lugar, el capítulo describe los aspectos más relevantes de las pruebas en el campo de la ingeniería del software. Posteriormente, se analiza el rol que juegan las pruebas en la metodología EODAM.

Finalmente, el capítulo concluye con una descripción de las pruebas que se han llevado a cabo.

Por tratarse de una línea de productos con un alcance de miles de productos, se ha realizado un esfuerzo para automatizar la fase de pruebas, creando un framework para el desarrollo de las pruebas, que es descrito en esta última sección.

9.1.Las pruebas en la Ingeniería del Software

La realización de pruebas en la ingeniería del software es uno de los procesos más importantes. **Existen diversos estándares que nos permiten normalizar el desarrollo de las pruebas** como el estándar para *Verificación del Software y Planes de Validación* (IEEE 1012-1986), el estándar para *Documentación de Pruebas de Software* (IEEE 829-1983), el estándar para *Pruebas de Unidad de Software* (IEEE 1008-1987] o la guía PSS-05-10 de *Verificación y Validación de la ESA* (European Space Agency).

Además**, todas las metodologías tratan de forma específica la realización de pruebas** como Rational Unified Process (RUP) [Kruchten 2004] o Metrica3 [METRICA3 2010].

En general, **las pruebas se clasifican en cinco grandes grupos**: **Pruebas Unitarias, Pruebas de Integración, Pruebas de Transición, Pruebas de Sistema y Pruebas de Aceptación.**

Si tratamos el proceso de ingeniería del software de forma genérica y dividimos su proceso en grandes bloques: la especificación de la necesidad, la planificación, la presupuestación, el análisis y el diseño, la construcción o el desarrollo, la implantación o puesta en producción, la aprobación y el sistema en uso, y ubicamos cada una de estos grupos de pruebas, obtendríamos una figura como la que se muestra a continuación:

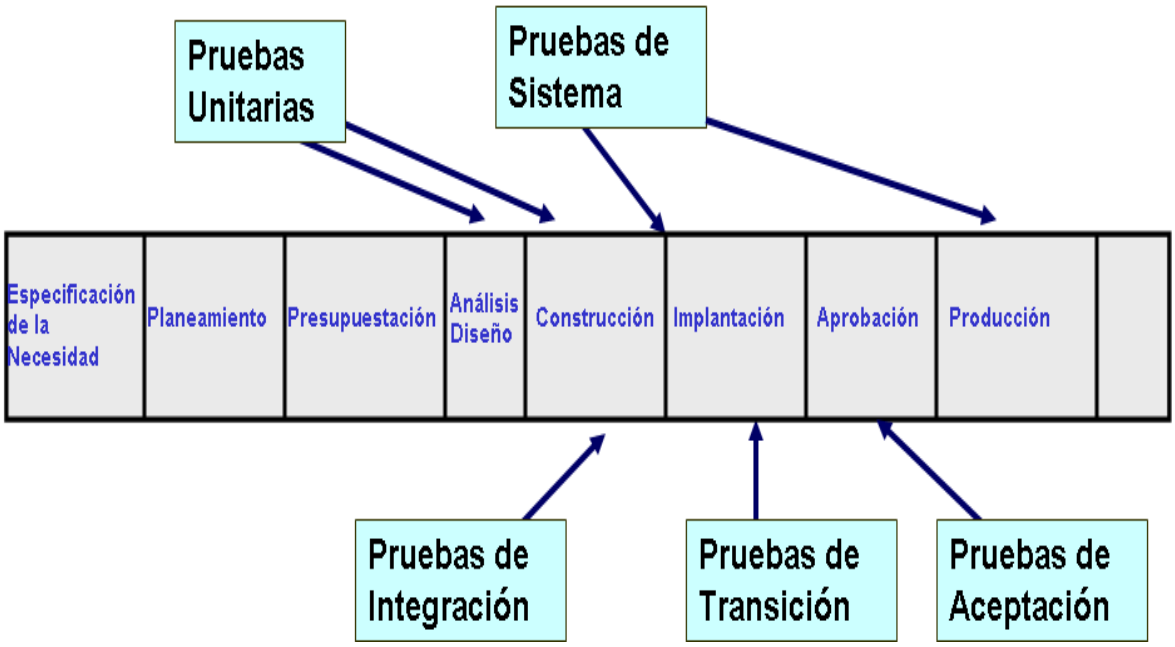


Figura 9-1 Las pruebas en el Ciclo de Vida del Software.

Por su relevancia, damos algunas pinceladas sobre cada uno de estos tipos de pruebas.

9.1.1. Pruebas Unitarias

Las pruebas unitarias son aquellas **pruebas de diseño y desarrollo de cada uno de los módulos** desarrollados y diseñados a medida para el Sistema. Cada unidad de software (base de datos, interfaz de usuario, lógica de negocio, etc.) deberá especificar sus propias pruebas unitarias.

La primera fase de las pruebas unitarias es la **Planificación de las Pruebas**, que incluye el análisis de roles, responsabilidades, fechas, etc. Es lo que se suele conocer con el nombre de *UTP (Unit Test Plan).*

A continuación, se **identifican todas las pruebas** que se van a realizar (a alto nivel), lo que se conoce como *UTD (Unit Test Description).* Hay tres tipos de pruebas que nos interesan para nuestro análisis:

- **De Caja Negra.** Aquellas pruebas donde solo se identifican las entradas y las salidas. No se incluyen los procesos intermedios.

- **De Caja Blanca.** Aquellas pruebas donde se incluyen los procesos intermedios: la parte del código por el que pasan las entradas de información (únicamente los módulos, no el contenido de ellos: entradas y salidas de cada uno de los módulos desarrollados). Las pruebas de denominadas de **Caja de Cristal** son un caso particular, donde se recorre la totalidad del código, como en el caso de la pruebas de calidad del código.

- **De Rendimiento**. Aquellas pruebas donde se miden los tiempos de cada función en procesar las salidas.

Posteriormente, **se identifican todos los casos de pruebas** que se van a realizar (a bajo nivel), que se conoce como *UTCD (Unit Test Case Description)* **y cómo se van a realizar las pruebas** (los procedimientos), que se denomina *UTPD (Unit Test Procedure Description).*

Finalmente, **se llevan a cabo las pruebas y se documentan los resultados** de las mismas en el UTR *(Unit Test Report).*

9.1.2. Pruebas de Integración

Constituyen las pruebas de integración de todos los módulos del sistema. El objetivo principal de estas pruebas es **determinar el grado de integración de los diferentes componentes del sistema entre sí y la evaluación de la integración del sistema en la plataforma** en la que debe operar.

Las fases a realizar son muy similares a las pruebas unitarias, incluyendo un *ITP (Integration Test Plan)* con la planificación, un *ITD (Integration Test Description)* con la identificación de las pruebas y un *ITCD (Integration Test Case Description)* con los casos de prueba, un *ITCD (Integration Test Case Description)* con los procedimientos y un *ITR (Integration Test Report)* con los resultados. Además, se suele incluir la misma tipología de pruebas a realizar que en las pruebas unitarias.

9.1.3. Pruebas de Transición

El objetivo de las pruebas de transición es **verificar el paso al entorno de producción del sistema desde el entorno de desarrollo o preproducción** (o ambos, dependiendo del caso). El objetivo es probar que sobre el nuevo entorno de hardware y software el sistema opera con corrección.

Estas pruebas deben verificar el cumplimiento de los requisitos adicionales expuestos por el organismo responsable de la explotación del sistema. En realidad, las pruebas de transición constituyen una tipología específica de las pruebas de integración.

De la misma forma que en las pruebas unitarias y de transición las fases son idénticas, incluyendo el *TTP (Transition Test Plan), el TTD (Transition Test Description),* el *TTCD (Transition Test Case Description),* el *TTPD (Transition Test Procedure Description)* y el *TTR (Transition Test Report).*

La **tipología** de este tipo de pruebas suele incluir:

- Pruebas de instalación / desinstalación del Sistema.
- Pruebas de arranque y parada de todas y cada una de las capas del sistema: base de datos, servidores de aplicaciones, lógica de negocio, etc.

- Pruebas de copias de seguridad y recuperación del Sistema.

- Pruebas de seguridad y control de accesos al sistema, incluyendo todos los roles del sistema.

9.1.4. Pruebas de Sistema

Son las pruebas principales en un Sistema de Información. **Su objetivo es verificar el cumplimiento de los requisitos software**. En multitud de sistemas únicamente se contemplan este tipo de pruebas, junto con las pruebas de aceptación.

De la misma forma que en las pruebas unitarias y de integración - transición las fases son idénticas, incluyendo el *STP (Software Test Plan)*, el *STD (Software Test Description)*, el *STD (Software Test Description)*, *el STPD (Software Test Procedure Description)* y el *STR (Software Test Report).*

Podemos clasificar las pruebas de sistema de la siguiente manera:

- **Pruebas de Seguridad.** El objetivo principal de las mismas es verificar que no hay daños a personas debidas al funcionamiento del Sistema. También, engloban temas jurídicos y normativas de Seguridad como INFOSEC [DEF73 2002], COMMON CRITERIA [CC 2010], ITSEC [IS 2006], cumplimiento de la Ley Orgánica de Protección de Datos [LOPD 2010], pruebas específicas en entornos de sistemas que, por su funcionalidad, afecten a las personas como los sistemas para sanidad, por ejemplo, o sistemas que manejen información clasificada o secreta.

- **Pruebas de Funcionalidad.** El objetivo de las mismas es **verificar el cumplimiento de los requisitos funcionales del Sistema.** Estas pruebas deben abarcar todos los requisitos funcionales del sistema. **Son las pruebas más comunes.**

- **Pruebas de Rendimiento.** El objetivo de las mismas es medir los tiempos medios de respuesta para cada una de las funcionalidades del sistema.

- **Pruebas de Interfaces.** Estas pruebas tienen como objetivo verificar que los interfaces que interactúan con el Sistema operan según los requisitos establecidos. Se han de verificar los interfaces de salida, es decir, aquellas salidas desde el sistema bajo estudio hacia otros sistemas externos y los interfaces de entrada, es decir, aquellas entradas desde otros sistemas externos al sistema bajo estudio.

- **Pruebas de Recursos.** Estas pruebas tienen como objetivo verificar los recursos adicionales del sistema: impresoras, CPU, sistemas de almacenamiento, tarjetas, etc. Ejemplos típicos: tamaño de los discos y unidades de almacenamiento frente a volumen de información que maneja el sistema, % de uso de la CPU, tamaño de la memoria RAM de los dispositivos (memoria principal, y memorias secundarias, como tarjetas gráficas u otros dispositivos), que las impresoras realizan correctamente su función, etc.

- **Pruebas de Copia y Restauración.** Estas pruebas incluyen: la verificación de las paradas, arranques y reinicio de todas y cada una de las capas del sistema (BBDD, Servidor de Aplicaciones, cliente, etc.) y las pruebas de cumplimiento de las copias de salvaguarda del sistema y su posterior recuperación. En función de los requisitos del sistema, cada una de las copias volcará parte de la información del sistema.

- **Pruebas de Portabilidad.** Incluyen la posibilidad de instalar el sistema en otro entorno de HW y SW de Base. Se han de contemplar todas las posibles plataformas futuras sobre las que se va implantar el sistema. Incluyen las **pruebas de autonomía**, para garantizar el cumplimiento de los requisitos de despliegue en unidades autónomas del sistema. Ejemplos: tiempo máximo de supervivencia del sistema autónomo, tiempo de repliegue (paso del principal al autónomo), de despliegue (instalación del sistema autónomo), capacidades del sistema autónomo, etc.

- **Pruebas de Stress.** El objetivo de las mismas es verificar el cumplimiento de las funcionalidades del sistema entre los límites máximos y mínimos establecidos por los requisitos del mismo. Ejemplos: *concurrencia* (número máximo de usuarios que acceden al sistema), *procesamiento* (número máximo de procesos por unidad de tiempo), tiempo máximo de parada y arranque, tiempo máximo de recuperación, tiempo máximo de instalación y configuración, etc.

- **Pruebas de Regresión.** El objetivo de estas pruebas es verificar que ante un cambio en el sistema, éste opera con corrección. Estas pruebas pueden implicar una repetición de todas y cada una de las pruebas de sistema, integración, unitarias, etc. Deben estar perfectamente dimensionadas para que el coste de su desarrollo no sea excesivo.

- **Pruebas de Mantenimiento.** El objetivo de estas pruebas es verificar el grado de cumplimiento del mantenimiento y soporte del sistema. Ejemplos: TMAF (tiempo medio de arreglo de fallos), número de incidencias por unidad de tiempo, tiempo medio de respuesta a las incidencias, etc.

- **Pruebas de Operación / Usabilidad.** Estas pruebas incluyen garantizar la correcta visualización y el entendimiento de todas y cada una de las pantallas gráficas del sistema, el correcto acceso a las ayudas del sistema y las pruebas

que garanticen la accesibilidad a la información, incluyendo pruebas de usuarios del sistema con discapacidades.

- **Pruebas de Entregables.** El objetivo de estas pruebas es verificar aquellos entregables externos al propio sistema en sí. Principalmente incluye la documentación (manuales de usuario, manuales de administración, documentación de análisis y diseño...) y los dispositivos externos (si un sistema tiene impresoras, escáneres, monitores..), etc.

- **Pruebas de Auditoría.** El objetivo de estas pruebas es garantizar los requisitos de auditoría del sistema. Se suele incluir la gestión de LOG's y los requisitos de monitorización del Sistema (procesos y usuarios).

- **Pruebas de Gestión de Roles y Usuarios.** El objetivo de estas pruebas es verificar el funcionamiento de la gestión de roles y usuarios del sistema, y el control de acceso al mismo. Además, se suelen incluir las capacidades de administración de usuarios del sistema: bloqueo de usuarios y procesos, altas, bajas, modificaciones, creación de nuevos roles, etc.

- **Pruebas de Fiabilidad.** Aquellas pruebas cuyo objetivo es garantizar que el sistema es fiable (no produce fallos). Los fallos suelen ser clasificados. Incluimos una clasificación, a modo de ejemplo:

    - *Muy Críticos*. El Sistema se queda parado y no puede continuar su operación.
    - *Críticos*. Alguna de las funciones principales del sistema no funciona con corrección o cierta información relevante contenida en el mismo no es correcta.
    - *Medianamente Críticos*. El Sistema funciona pero parte de la información no relevante en él contenida es corrupta o bien ciertas funciones no esenciales del mismo no funcionan con corrección. El Sistema puede continuar su operación.

- *No Críticos.* El Sistema puede continuar su operación y la información es correcta, pero se producen fallos (mensajes no esperados del sistema, tiempos de espera prolongados en la respuesta, en el arranque, etc.).

9.1.5. Pruebas de Aceptación

Son aquellas pruebas cuyo objetivo es **validar los requisitos de usuario del sistema**. El responsable funcional del sistema acepta el sistema una vez se hayan completado, y se haya revisado el último entregable relativo a las mismas (ATR, *Acceptance Test Report).*

Estas pruebas puedan abarcar todas las funciones realizadas por el usuario del sistema, o un conjunto relevante de las mismas, y corresponden a una visión de alto nivel (desde el punto de vista del usuario del sistema) de las pruebas de funcionalidad.

Las fases suelen incluir el el ATP (*Acceptance Test Plan*), el ATD. (*Acceptance Test Description*), el ATCD (*Acceptance Test Case Description*), el ATPD (*Acceptance Test Procedure Description*), que detalla todos los pasos que debe de realizar el usuario sobre el sistema y el ATR (*Acceptance Test Report*).

Estas son las únicas pruebas obligatorias en la mayor parte de los estándares y normativas ad hoc de las organizaciones.

9.1.6. Otros tipos de pruebas

Tanto en las pruebas de sistema como en las pruebas de aceptación pueden tener lugar otra tipología de pruebas:

- **Pruebas Alfa y Beta.** Aquellas pruebas de sistema y aceptación que son realizadas en dos etapas. En una primera etapa (pruebas alfa) las pruebas de sistema se realizan por los usuarios dentro de la organización que desarrolla el software y en una segunda etapa (pruebas beta) las pruebas de aceptación son realizadas, generalmente, por un número limitado de usuarios externos.

- **Pruebas Piloto.** La prueba piloto es una prueba preliminar de sistema o de aceptación que se enfoca en aspectos específicos y determinados del sistema. Su objetivo es probar un alcance limitado del sistema.

- **Pruebas en Paralelo.** Aquellas pruebas que se llevan a cabo mediante un proceso que introduce datos de prueba en dos sistemas: el sistema modificado y un sistema alternativo, que suele ser el sistema original, y comparar posteriormente los resultados. Este tipo de pruebas son realizadas comúnmente en la fase de mantenimiento de un sistema.

9.2.Las pruebas en EODAM

"Las pruebas son una fase cara y laboriosa del proceso de software" [Sommerville 2006]. En el caso de las pruebas unitarias, su formalización constituye paso decisivo para la reutilización del software. Más si cabe, en el caso de la construcción de líneas de productos (SPL).

Varias líneas de investigación respaldan que la reutilización de componentes posibilita, en gran medida, la reutilización de las pruebas de dichos componentes [Edwards 2001] [Lonngren 1998] [Michael 1997].

Algunos autores consideran que la reutilización de pruebas puede incrementarse notablemente en dominios específicos, aprovechando la cercanía entre las aplicaciones de una familia [Dallal and Sorenson] [Mayrhauser et al 1994].

EODAM es una metodología ligera de construcción de líneas de productos software y pretende abordar la fase de pruebas de forma ágil, evitando el desarrollo de pruebas que supongan un sobrecoste adicional. En el desarrollo de las SPL el número de funciones a probar puede ser muy elevado.

Como ya hemos analizado con anterioridad, en nuestro caso, estamos ante el desarrollo de miles de productos y, por tanto, dimensionar las pruebas es una tarea importante para evitar costes que supongan un riesgo, en lo que a la rentabilidad de la solución se refiere. **En la metodología EODAM se incluye el desarrollo de pruebas dentro de unas fases identificadas** en color morado en la figura adjunta:

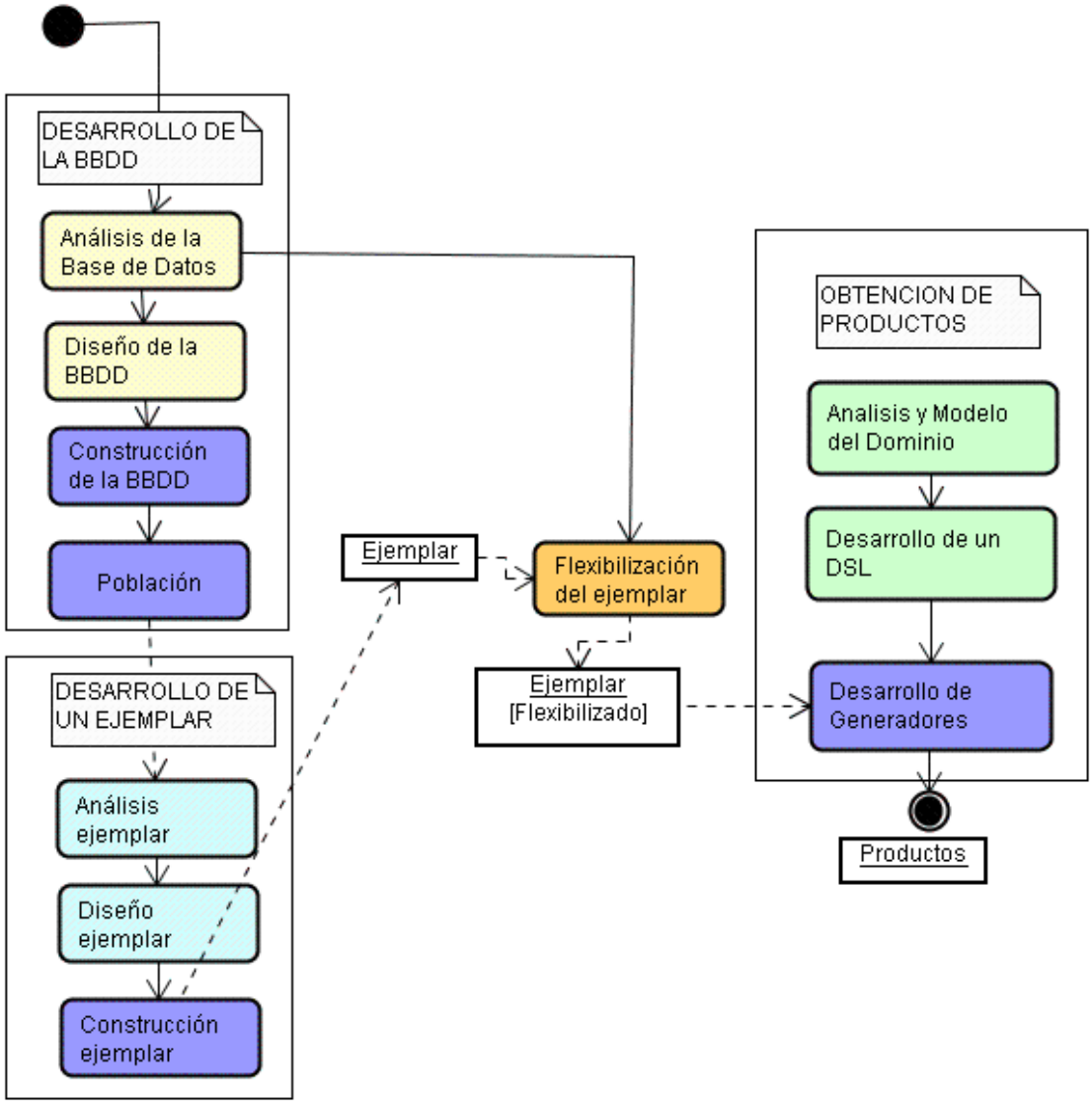


Figura 9-2 Las pruebas en EODAM.

Durante la construcción y población de la base de datos, en nuestro caso de la CMDB, se realizarán las pruebas necesarias para garantizar el correcto funcionamiento de la misma.

En multitud de casos la utilización de EODAM se realizará sobre una base de datos existente y, por tanto, no existirán las fases de desarrollo y pruebas de la base de datos. En el caso de no existir la CMDB, abogamos por la realización de las pruebas de sistema y aceptación, únicamente.

En lo que se refiere al desarrollo del ejemplar, **abogamos por la realización de todas las pruebas unitarias y de sistema** que nos permitan garantizar correctamente el funcionamiento del ejemplar del dominio, pues constituye, en nuestra metodología de desarrollo, el *"core"* de todo el proceso.

Finalmente, **la realización de pruebas de los Generadores** nos remitirá a las pruebas sobre los productos generados. En este punto del proceso, es vital dimensionar correctamente las pruebas y **señalar un conjunto mínimo de productos** sobre los cuales realizar las mismas.

Abogamos por la inclusión de un sistema que nos permita incluir en los propios generadores una automatización de las pruebas realizadas sobre el Ejemplar del Dominio, y que permita extender las mismas al resto de productos de la familia.

La automatización de pruebas ha supuesto en los últimos años un avance considerable, sobretodo en el caso de las pruebas unitarias y en metodologías ágiles como *Adaptive Software Development (ASD), Agile Unified Process (AUP), Crystal, Essential Unified Process (EssUP), Feature Driven Development (FDD), Lean Software Development (LSD), Open Unified Process (OpenUP) o la Programación Extrema (XP)* [ASD 2000] [Allen 2008] [LSD 2003] [XP 2003].

Han sido desarrollados multitud de marcos de trabajo para la automatización de pruebas como *JUnit* para Java [J.U. 2006], *NUnit* para C# [N.U. 2010], *CppUnit* para C++ [CPP.U. 2010], *RubyUnit* para Ruby [RU.U. 2010] o *HttpUnit* para aplicaciones web [HTTP.U. 2010], y existen algunas propuestas de aumento de la abstracción de marcos para pruebas de unidades mediante un enfoques generativos como [Huang and Chen 2005].

9.3.El Framework construido para las Pruebas

Para nuestro caso de estudio, **construimos un Framework de pruebas.** En primer lugar, realizamos las pruebas de nuestro Ejemplar, siguiendo un procedimiento que detallamos a continuación y que se refleja en la figura siguiente.

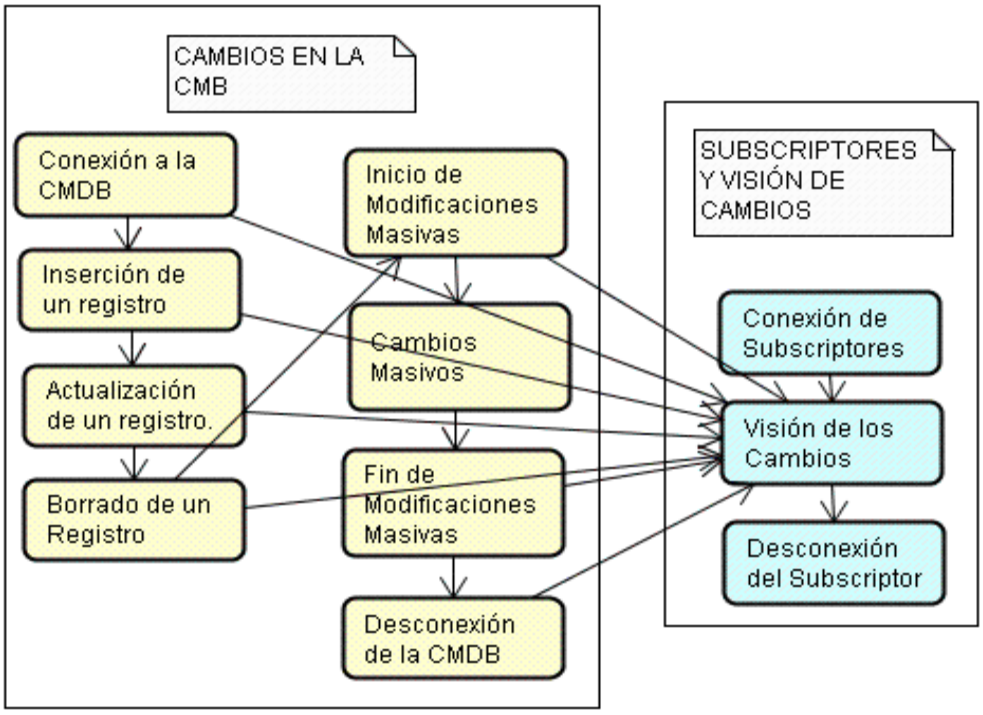


Figura 9-3 Procedimiento de Pruebas.

Realizamos las diferentes operaciones que suponen cambios o interactúan con la CMDB y desde los diferentes subscriptores se observan los cambios producidos.

Un usuario se conecta a la CMDB, inserta un registro nuevo, actualiza un registro y borra un registro. Posteriormente, se lleva a cabo un cambio masivo en la CMDB (siguiendo el proceso: inicio, cambio y fin). Finalmente, se desconecta de la CMDB.

Los subscriptores se conectan al servicio de notificación de los cambios y van recibiendo los diferentes cambios que se producen en la CMDB. Finalmente, se desconectan.

A continuación mostramos, a modo de ejemplo, un **caso de pruebas desarrollado para nuestro Ejemplar del Dominio.**

```
-- Connection
connect <user_admin>/<password>;
-- Changes
INSERT INTO PEOPLE ( First_Name, Last_Name, Role, Mob, Tlf, Email,
Description ) VALUES ('Jose Ramon', 'Coz Fernandez', 'Administrador',
'0034666666666', '0034911111111', 'jrcozf@gmail.com', 'Administrador
de las Bases de Datos')
commit;
UPDATE PEOPLE SET Tlf='0034911111111' WHERE User_ID='000001'
commit;
DELETE PEOPLE WHERE User_ID='000001'
commit;
INSERT INTO THIRD_COMPANIES ( Company_Name, Company_Address,
Company_Tlf, Description ) VALUES (
'UNED', 'Juan del Rosal', '911111111', 'Universidad a Distancia. Dpto.
Ingeniería del SW y Sistemas.12')
commit;
UPDATE THIRD_COMPANIES SET Company_Tlf='0034911111111' WHERE
Company_ID='000001'
commit;
DELETE THIRD_COMPANIES WHERE Company_ID='000001'
-- Massive Test.
EXECUTE AQ.P_MASIVO_COMIENZO;
INSERT INTO PEOPLE ( First_Name, Last_Name, Role, Mob, Tlf, Email,
Description ) VALUES ('Jose Ramon', 'Coz Fernandez', 'Administrador',
'0034666666666', '0034911111111', 'jrcozf@gmail.com', 'Administrador
de las Bases de Datos') commit;
UPDATE PEOPLE SET Tlf='0034911111111' WHERE User_ID='000001'
commit;
DELETE PEOPLE WHERE User_ID='000001'
commit;
```

```
INSERT INTO THIRD_COMPANIES ( Company_Name, Company_Address,
Company_Tlf, Description ) VALUES (
'UNED', 'Juan del Rosal', '911111111', 'Universidad a Distancia. Dpto.
Ingeniería del SW y Sistemas.12')/
UPDATE THIRD_COMPANIES SET Company_Tlf='0034911111111' WHERE
Company_ID='000001'
commit;
DELETE THIRD_COMPANIES WHERE Company_ID='000001'
EXECUTE AQ.P_MASIVO_FINAL;
---------------------
-- Subscribers and Notifications
---------------------
connect 'Financial_Sys'/<password>;
-- Views of the Changes…
SET SERVEROUTPUT ON;
DECLARE
message VARCHAR2(200);
BEGIN
AQ.P_DEQUEUE_MESSAGE ('AQ',message);
DBMS_OUTPUT.PUT_LINE(message);
AQ.P_DEQUEUE_MESSAGE ('AQ',message);
DBMS_OUTPUT.PUT_LINE(message);
AQ.P_DEQUEUE_MESSAGE ('AQ',message);
DBMS_OUTPUT.PUT_LINE(message);
AQ.P_DEQUEUE_MESSAGE ('AQ',message);
DBMS_OUTPUT.PUT_LINE(message);
...
END;
/
connect 'ERP_Sys'/<password>;
-- Views of the Changes…
...
```

Una vez realizadas las pruebas del Ejemplar, **observamos que la información necesaria para automatizarlas está relacionada con los elementos** (las entidades, o CI's de nuestra CMDB y sus atributos clave) y los **subscriptores** a la Base de Datos.

Además existen otros factores, como los requisitos establecidos en nuestro DSL y la información que se encuentra almacenada en la CMDB.

Desde el DSL y el Ejemplar de Pruebas para el dominio podemos generar el resto de pruebas para nuestros productos, que era el objetivo que perseguíamos.

Con respecto a la información que se encuentra almacenada en la CMDB, podemos tomar de ejemplo cualquier registro de la misma para realizar las correspondientes pruebas.

10. Estudio económico y análisis de rentabilidad

"(...) descubrí que había que diseñar
de nuevo incluso los gráficos tradicionales
de la economía si queríamos que la
sombría ciencia de la economía
se convirtiera en la apasionante
disciplina que realmente era."
Paul Samuelson

En este capítulo se detalla un estudio económico de la línea de productos construida para dar soporte a nuestro dominio, un servicio de comunicación de los cambios que se producen en una CMDB al resto de procesos de negocio relacionados con la gestión de las TIC.

El capítulo comienza con un breve resumen sobre los modelos de costes y la rentabilidad en las líneas de productos software.

En la siguiente sección se presenta el modelo económico COMPLIMO-EODAM, un modelo matemático para obtener el coste y la rentabilidad de un desarrollo realizado bajo el paraguas de la metodología EODAM. Posteriormente, en base al modelo presentado, se obtiene la rentabilidad del desarrollo llevado a cabo para nuestro dominio.

Finalmente, el capítulo presenta una valoración general de la propuesta de desarrollo del dominio bajo estudio, donde se analizan diversas métricas y factores que influyen en los costes y la rentabilidad.

10.1. Los Modelos de Costes y la Rentabilidad en las SPL

La viabilidad económica del desarrollo *líneas de productos software* (SPL) depende, en gran parte, del alcance y del coste de proyectar las *características* del domino bajo estudio sobre el código. En la mayor parte de los casos, este desarrollo se lleva a cabo por primera vez y no se conoce la complejidad y esfuerzo que conllevan las tareas a realizar, y tampoco se conoce el coste que puede suponer.

Esta situación convierte la tarea de estimación en una tarea compleja. Se podría efectuar una estimación asistida por guías que comuniquen la experiencia de otras personas que han aplicado la metodología y/o estimar en grados de magnitud del tamaño del proyecto.

En el caso de la metodología EODAM, otra dificultad que se antepone a un conocimiento más exacto del coste de creación de la *línea de productos* es la necesidad de conocer en profundidad el ejemplar.

No obstante, a medida que avanzamos en la identificación de los cambios concretos que se tienen que efectuar sobre el ejemplar, nos podemos hacer una idea más acertada del coste real. En definitiva, la viabilidad económica de la línea se debería de plantear como una actividad que se efectúa en varias etapas de todo el proceso de construcción de la *línea de productos*.

Por ejemplo, en una primera etapa, la actividad de viabilidad se podría desarrollar en la fase de identificación y caracterización de la *línea de productos*. En esta etapa no sería necesario considerar aspectos relacionados con la implementación.

En una segunda etapa, se continuaría con el análisis de la viabilidad, que se realizaría una vez definidos todos los cambios que se tienen que efectuar sobre los ejemplares. Esta segunda parte del estudio de viabilidad podría abarcar la realización de un modelo de costes basado en los Generadores a desarrollar, y tomaría como referencia el coste de la construcción de otros generadores en el pasado. El coste de implementar un generador dependerá del tipo de cambios que efectúa el generador.

Este enfoque está más centrado sobre el aspecto de costes de la construcción de la línea de productos, que sobre el aspecto de la rentabilidad. Su objetivo sería determinar cuánto cuesta construir la línea de productos, pero no lo rentable o beneficioso que puede ser su construcción. Para conocer si es nuestra propuesta es rentable tendríamos que disponer de datos objetivos sobre el coste de construcción de un único producto y sobre el coste de construir un único producto utilizando la línea de productos. También, deberíamos de tener en cuenta cual es el coste de construir la línea de productos.

El modelo que nos permitiese obtener la rentabilidad sería sumamente ventajoso para la toma de decisiones en el campo de la ingeniería de dominio y las líneas de productos software. Además, este modelo formaría parte del *Bussines Case* de nuestro proyecto. Como ya hemos mencionado con anterioridad, el objetivo de este capítulo es exponer este modelo. En nuestra propuesta, tanto la variabilidad del espacio problema como de la solución, son expresables a través de un lenguaje específico del dominio (DSL) que hereda de unos modelos de características bien definidos y normalizados. **Será importante considerar**, en nuestro modelo, **el coste de construir el lenguaje específico del dominio**, ya que en algunos casos este coste puede ser considerable.

Otro aspecto a considerar en nuestro modelo será el coste del desarrollo del propio Ejemplar, que no es más que un producto de la familia, **y de la construcción de los Generadores**. Dentro el coste de los Generadores, se deberán considerar los analizadores del DSL, los generadores de los informes necesarios y los generadores de pruebas, tal y como se gestionan desde los Frameworks construidos, y que fueron detallados en los capítulos ocho y nueve.

El coste de construcción de la CMDB no será relevante, en lo que se refiere a la rentabilidad de nuestra solución, pues el enfoque utilizado no representa ningún sobrecoste sobre un desarrollo tradicional de una CMDB.

A continuación, se detalla el modelo económico que tiene en consideración todos estos aspectos y **que nos permite obtener la rentabilidad de nuestra propuesta**.

10.2. Modelo adaptado a EODAM

El modelo matemático de costes que nos permite obtener los beneficios proporcionados por nuestra solución **está basado en el estándar COPLIMO** [Boehm et al. 2004], una extensión del modelo COCOMO II [Boehm et al. 2000]. **Denominaremos a nuestro Modelo: COPLIMO-EODAM.**

La primera ecuación de nuestro modelo es:

$$\mathbf{PLS(N) = PMNR(N) - PMR(N)}$$

Ecuación 10-1. Ecuación Inicial del Modelo COPLIMO-EODAM.

Donde:

- PLS(N) es el ahorro de construir nuestra línea de productos software (SPL), que está compuesta de N productos (PLS, *Product Line Savings*),

- PMR(N) es el coste en personas / mes (*PM, person / months*) de construir N productos con la línea de productos (SPL) y

- PMNR(N) es el coste, en personas / mes, de construir N productos sin hacer uso de la línea de productos (SPL).

El valor de PMNR(N) se estima utilizado el estándar COCOMO II, según la siguiente ecuación :

$$\mathbf{PM = A\ x\ Size}^{\mathbf{E}}\ \boldsymbol{x}\ \prod_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{EM_i}$$

Ecuación 10-2. Ecuación de Costes según COCOMOII.

Donde:

- A es una *constante organizacional*,

- E es el *Parámetro de Escala* (*"scaling parameter")* y

- EM_i son los *Multiplicadores de Esfuerzo* ("*Effort Multipliers")*, que incluyen: la complejidad de los productos, la reutilización, el tamaño de la base de datos, la volatilidad de la plataforma, el almacenamiento, el tiempo de ejecución, la capacidad de los programadores, la continuidad del personal, la experiencia en el uso de los lenguajes, las herramientas y otros aspectos que influyen en el coste de la línea de productos.

El *Parámetro de Escala*, E, refleja el esfuerzo para desarrollar grandes proyectos software debido al crecimiento entre las comunicaciones del personal y la sobrecarga del proceso de integración, entre otros aspectos.

Este parámetro se obtiene a través de la siguiente ecuación:

$$\mathbf{E = B + 0.01\ x \sum_{i=1}^{n} SF_i}$$

Ecuación 10-3. Parámetro de Escala.

Donde:

- B es una constante denominada *"Scaling base-exponent for effort"* y
- SF_i son los *Factores de Escala* (*"scale factors"),* que incluyen varios aspectos que influyen en el coste del desarrollo como la resolución de riesgos, la cohesión del equipo de trabajo, la madurez del proceso software, la arquitectura, la flexibilidad del desarrollo y otros.

El coste, en personas / mes, de construir N productos sin hacer uso de la línea de productos vendrá dado por la expresión siguiente:

$$\mathbf{PMNR(N) = N\ x\ A\ x\ Size^E\ x \prod_{i=1}^{n} EM_i}$$

Ecuación 10-4. Coste del desarrollo, sin usar la SPL.

Según el estándar COMPLIMO, el coste en personas / mes de construir N productos con la línea de productos (SPL) se calcula por las siguientes ecuaciones:

$$\mathbf{PMR(N) = PMNR(1)\ x\ \left(PFRAC + RCWR\ x\ (RFRAC + AFRAC)\right)}$$

Ecuación 10-5. Coste del desarrollo, usando la SPL, para N=1.

$$\mathbf{PMR(N) = PMR(1)} \\ \mathbf{+ (N - 1)\ x\ PMNR(1)\ x\ \left(PFRAC + RFRAC\ x \frac{AA}{100}\right.} \\ \mathbf{\left. + AFRAC\ x\ AAM\right)}$$

Ecuación 10-6. Coste de desarrollo, usando la SPL, para N>1.

Donde:

- PFRAC, RFRAC y AFRAC son únicos, y constituyen las partes de reutilización de caja negra y caja blanca de nuestros productos,
- RCWR es el coste relativo de codificar la reutilización (*Relative Cost of Writing for Reuse*) y
- AA es el *parámetro de Evaluación y Asimilación* (*Assessment and Assimilation*), y representa el esfuerzo necesario para evaluar el componente candidato reutilizable y elegir el más apropiado, más el esfuerzo por asimilar el código del componente y la documentación en el nuevo producto.

El parámetro RCWR es un multiplicador para estimar el esfuerzo que es necesario llevar a cabo para hacer reutilizable el software, a través de la línea de productos. Se obtiene a partir de la siguiente ecuación:

$$\mathbf{RWCR = RUSE\ x\ DOCU\ x\ RELY}$$

Ecuación 10-7. Coste del desarrollo de la reutilización

Donde RUSE es el desarrollo para la reutilización, DOCU es el grado de documentación y RELY representa la fiabilidad del software.

Para nuestro caso de estudio el coste, en personas / mes, de construir N productos, sin hacer uso de la línea de productos, vendrá dado por la expresión siguiente:

$$\mathbf{PMNR(N) = N\ x\ A\ x\ esize^E\ x \prod_{i=1}^{n} eEM_i}$$

Ecuación 10-8. Coste del desarrollo de nuestro caso, sin usar la SPL.

Donde *esize* es el tamaño del Ejemplar y eEM son los multiplicadores del ejemplar.

Y el coste para construir N productos, haciendo uso de nuestra línea de productos, vendrá dado por la siguiente expresión:

$$\mathbf{PMR(N)} = \begin{cases} \mathbf{PMNR(1) + \sum_{i=1}^{G} [A\ x\ gsize^E\ x \prod_{i=1}^{n} gEM_i\ ],\ n = 1} \\ \mathbf{PMR(1) + (N-1)x\ \frac{AA}{100\ x\ (\sum_{i=1}^{G} [A\ x\ gsize^E\ x \prod_{i=1}^{n} gEM_i])},\ n > 1} \end{cases}$$

Ecuación 10-9. Coste del desarrollo de nuestro caso, utilizando la SPL.

Donde *gsize* es el tamaño de los Generadores, G es el número de Generadores y eEM son los Multiplicadores de Esfuerzo de los Generadores.

Pero **nuestro interés se centra en estimar el ROI** (Return On Investment) de nuestra propuesta.

El ROI se obtendrá mediante la ecuación:

$$\mathbf{ROI(N) = \frac{cost\ savings}{cost\ investiment} = \frac{PLS(N)}{|PLS(1)|}}$$

Ecuación 10-10. Rentabilidad de nuestra línea de productos.

10.3. Estudio detallado de Costes

Una vez presentado el modelo, realizaremos un estudio detallado de los costes con el objetivo último de estimar la rentabilidad. Para ello, en primer lugar, obtendremos los Factores de Escala, posteriormente los Multiplicadores de Esfuerzo y, finalmente, el resto de los parámetros del modelo.

10.3.1. Factores de Escala

Los Factores de Escala (*"scale factors"),* incluyen varios aspectos que influyen en el coste del desarrollo:

- **Precedentedness** (PREC), captura la experiencia previa en cuanto al desarrollo de proyectos similares.

- **Development Flexibility** (FLEX), mide la flexibilidad del proceso de desarrollo en relación a los requisitos establecidos.

- **Architecture / Risk Resolution** (RESL), pondera el nivel de riesgo asociado al proyecto y el porcentaje de respuesta que es capaz de lograr la organización ante la ocurrencia de un riesgo.

- **Team Cohesion** (TEAM), mide el tipo de interacción entre el equipo de proyecto y su impacto.

- **Process Maturity** (EPML), representa el grado de madurez de la organización en relación con las prácticas clave del Modelo de Madurez CMM (Common Maturity Model) [CMM 1993].

El valor de estos factores, según COCOMOII, se adjunta en la siguiente tabla:

CMDB NOTIFICATION SERVICE ECONOMIC MODEL SCALE FACTORS						
SCALE FACTORS	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedeted	largely unprecedeted	somewhat unprecedeted	generally familiar	largely familiar	thoroughly familiar
SF_j	6,2	4,96	3,72	2,48	1,24	0
FLEX	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
SF_j	5,07	4,05	3,04	2,03	1,01	0
RESL	little (20%)	some (40%)	often (60%)	generaly (75%)	mostly (90%)	full (100%)
SF_j	7,07	5,65	4,24	2,83	1,41	0
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
SF_j	5,48	4,38	3,29	2,19	1,1	0
EPML	SW-CMM Level 1. Lower	SW-CMM Level 1. Upper	SW-CMM Level 2	SW-CMM Level 3	SW-CMM Level 4	SW-CMM Level 5
SF_j	7,8	6,24	4,68	3,12	1,56	0

Tabla 10-1 Factores de Escala según COCOMOII.

Obtenemos, para nuestro ejemplar y los generadores, los siguientes valores, detallados en la tabla adjunta:

CMDB NOTIFICATION SERVICE ECONOMIC MODEL SCALE FACTORS RESUME			
SCALE FACTOR	DESCRIPTION	EXEMPLAR	GENERATOR
PREC	Precedentedness	largely familiar	largely familiar
SF_j		1,24	1,24
FLEX	Development Flexibility	some conformity	some conformity
SF_j		1,01	1,01
RESL	Architecture / Risk Resolution	mostly (90%)	mostly (90%)
SF_j		1,41	1,41
TEAM	Team Cohesion	highly cooperative	highly cooperative
SF_j		1,1	1,1
EPML	Process Maturity	SW-CMM Level 4	SW-CMM Level 4
SF_j		1,56	1,56
	$\sum SF_j$	**6,32**	**6,32**

Tabla 10-2 Factores de Escala para el Ejemplar y los Generadores.

10.3.2. Multiplicadores de Esfuerzo

A continuación analizamos los Multiplicadores de Esfuerzo (“*Effort Multipliers”).*

El primer multiplicador, **RELY**, contempla la influencia de la confiabilidad del software; el segundo multiplicador, **DATA**, considera el tamaño de la base de datos, en nuestro caso, la CMDB; **TIME** refleja el esfuerzo adicional requerido como producto de las restricciones de tiempo de ejecución; **STOR** considera el almacenamiento principal; **PVOL** la volatilidad de la plataforma; **ACAP** y **PCAP** las capacidades de los analistas y los programadores, respectivamente; **CPLX** la complejidad de los productos; **RUSE** el esfuerzo adicional requerido para construir componentes que deberán de ser reutilizados, tanto en el proyecto como en proyectos futuros; **DOCU** la documentación requerida y **PCON** la tasa de rotación del personal.

La experiencia del personal con la plataforma, la aplicación, los lenguajes y las herramientas se consideran con los multiplicadores **PLEX, APLEX, LPLEX y TOOL.**

Por último, **SITE** tiene en consideración el lugar donde se desarrolla y **SCED** las restricciones de tiempo para el desarrollo.

A continuación adjuntamos los posibles valores de estos multiplicadores:

RELY Cost Driver						
RELY Descriptors	slight invconvenience	low, easily recoverable lisses	moderate, easily recoverable lisses	high financial loss	risk to human life	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0,82	0,92	1	1,1	1,26	n/a

Tabla 10-3 RELY Cost Driver.

DATA Cost Driver						
DATA Descriptors		Testing DB bytes/Pgm SLOC<10	10≤D/P<100	100≤D/P<1000	D/P≥1000	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0,9	1	1,14	1,28	n/a

Tabla 10-4 DATA Cost Driver.

TIME Cost Driver						
TIME Descriptors			≤50% use of available execution time	70% use of available execution time	85% use of available execution time	95% use of available execution time
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1	1,05	1,17	1,46

Tabla 10-5 TIME Cost Driver.

STOR Cost Driver						
STOR Descriptors			≤50% use of available storage	70% use of available storage	85% use of available storage	95% use of available storage
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1	1,05	1,17	1,46

Tabla 10-6 STOR Cost Driver.

PVOL Cost Driver						
PVOL Descriptors		Major change every 12 mo.; Minor change every 1 mo.	Major change every 6 mo.; Minor change every 2 mo.	Major change every 1 mo.; Minor change every 1 wk.	Major change every 2 wk.; Minor change every 2 days.	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0,87	1	1,15	1,3	n/a

Tabla 10-7 PVOL Cost Driver.

ACAP Cost Driver						
ACAP Descriptors	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1,42	1,19	1	0,85	0,71	n/a

Tabla 10-8 ACAP Cost Driver.

CPLX Cost Driver						
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0,73	0,87	1	1,17	1,34	1,74

Tabla 10-9 CPLX Cost Driver.

RUSE Cost Driver						
RUSE Descriptors		None	Across project	Across program	Across product line	Across multiple product line
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0,95	1	1,07	1,15	1,24

Tabla 10-10 RUSE Cost Driver.

DOCU Cost Driver						
DOCU Descriptors	Many life-cycle needs uncovered	Some life-cycle needs uncovered	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very excessive for life-cycle needs	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0,81	0,91	1	1,11	1,23	n/a

Tabla 10-11 DOCU Cost Driver.

PCAP Cost Driver						
PCAP Descriptors	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1,34	1,15	1	0,88	0,76	n/a

Tabla 10-12 PCAP Cost Driver.

PCON Cost Driver						
PCON Descriptors	48% / year	24% / year	12% / year	6% / year	3% / year	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1,29	1,12	1	0,9	0,81	n/a

Tabla 10-13 PCON Cost Driver.

PLEX Cost Driver						
PLEX Descriptors	≤2 months	6 months	1 year	3 years	6 years	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1,19	1,09	1	0,91	0,85	n/a

Tabla 10-14 PLEX Cost Driver.

APEX Cost Driver						
LTEX Descriptors	≤2 months	6 months	1 year	3 years	6 years	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1,22	1,1	1	0,88	0,81	n/a

Tabla 10-15 APEX Cost Driver.

LTEX Cost Driver						
LTEX Descriptors	≤2 months	6 months	1 year	3 years	6 years	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1,2	1,09	1	0,91	0,84	n/a

Tabla 10-16 LTEX Cost Driver.

TOOL Cost Driver						
TOOL Descriptors	edit, code, debug	simple,frontend, backend, CASE, little integration	basic life-cycle tools moderately integration	strong, mature life-cycle tools, moderately integration	strong, mature, proactive life-cycle tools, reuse	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1,17	1,09	1	0,9	0,78	n/a

Tabla 10-17 TOOL Cost Driver.

SITE Cost Driver						
SITE Descriptors	International, phone, email	multiy city & company, phone, FAX	multiy city or company, narrow band, email	same city or metro area, wideband comunication	same building , wideband com, video conference	Fully collocated, interactive multimedia
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1,22	1,09	1	0,93	0,86	0,8

Tabla 10-18 SITE Cost Driver.

SCED Cost Driver						
SCED Descriptors	75% of nominal	85% of nominal	100% of nominal	130% of nominal	160% of nominal	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1,43	1,14	1	1	1	n/a

Tabla 10-19 SCED Cost Driver.

Según nuestro modelo, hacemos unas aproximaciones y obtenemos los datos que se muestran en las dos tablas siguientes, con los valores de los efectos multiplicadores para el desarrollo del Ejemplar del dominio y de los Generadores.

Como podemos observar, los valores obtenidos para el desarrollo del ejemplar del dominio y de los Generadores son los mismos, pues en ambos casos se trata de un proyecto nuevo, donde la documentación, el lugar de desarrollo, la experiencia, la plataforma, el almacenamiento, la base de datos y otros aspectos como las restricciones de tiempo o las capacidades del personal coinciden.

EFFORT MULTIPLIER	DESCRIPTION	EXEMPLAR	GENERATOR
DOCU	Documentation Match to Life-Cycle Needs	Right-sized to life-cycle needs	Right-sized to life-cycle needs
EM_j		1	1
PCAP	Programmer Capability	55th percentile	55th percentile
EM_j		1	1
PCON	Personnel Continuity	12% / year	12% / year
EM_j		1	1
APEX	Applications Experience	1 year	1 year
EM_j		1	1
PLEX	Plataform Experience	1 year	1 year
EM_j		1	1
LTEX	Language and Tool Experience	1 year	1 year
EM_j		1	1
TOOL	Use of Software Tools	edit, code, debug	edit, code, debug
EM_j		1,17	1,17
SITE	Multisite Development	multiy city or company, narrow band, email	multiy city or company, narrow band, email
EM_j		1	1
SCED	Required Development Schedule	160% of nominal	160% of nominal
EM_j		1	1
	∏Emj (P2)	**1,17**	**1,17**

Tabla 10-20 Efectos Multiplicadores 1 de 2.

EFFORT MULTIPLIER	DESCRIPTION	EXEMPLAR	GENERATOR
RELY	Required Software Reliability	high financial loss	high financial loss
EM_j		1,1	1,1
DATA	Database Size	D/P≥1000	D/P≥1000
EMj		1,28	1,28
TIME	Execution Time Constraint	70% use of available execution time	70% use of available execution time
EMj		1,05	1,05
STOR	Main Storage Constraint	85% use of available storage	85% use of available storage
EMj		1,17	1,17
PVOL	Plataform Volatility	Major change every 6 mo.; Minor change every 2 mo.	Major change every 6 mo.; Minor change every 2 mo.
EMj		1	1
ACAP	Analyst Capability	55th percentile	55th percentile
EMj		1	1
CPLX	Product Complexity	Nominal	Nominal
EMj		1	1
RUSE	Developed for Rusability	Across product line	Across product line
EMj		1,15	1,15
	∏Emj (P1)	**1,99**	**1,99**
	∏Emj (P1*P2)	**2,33**	**2,33**

Tabla 10-21 Efectos Multiplicadores 2 de 2.

10.3.3. Parámetros del Modelo

Una vez obtenidos los factores de escala y los efectos multiplicadores, procedemos a obtener el resto de parámetros del Modelo.

En primer lugar, tratamos el parámetro de Evaluación y Asimilación (AA, Assessment and Assimilation). Los valores de este parámetro se obtienen según la tabla siguiente:

CMDB NOTIFICATION SERVICE ECONOMIC MODEL ASSESMENT AND ASSIMILATION.	
AA Increment	**DESCRIPTION**
0	None
2	Basic module search and documentation
4	Some module Test and Evaluation (T&E), documentation
6	Considerable module Test and Evaluation (T&E), documentation
8	Extensive module Test and Evaluation (T&E), documentation

Tabla 10-22 Parámetro de Evaluación y Asimilación.

Todos los parámetros que se obtienen son detallados en la tabla siguiente, donde se incluye: la constante organizacional (A), el Parámetro de Escala *("scaling parameter"*), los tamaños de los Generadores y el Ejemplar, las agregaciones correspondientes de los factores de escala y los efectos multiplicadores.

PARAMETER	DESCRIPTION	VALUE
A	Effort coefficient	2,94
B	Scaling base-exponent for effort	0,91
∑SFj(gen)	Scale Factors for the Generators	6,32
∏Emj (gen)	Product of 17 Effort Multipliers for the Generators	2,33
E (gen)	Scaling exponent for effort (Generators)	0,97
Size (ex)	Size of the Exemplar in PM (person /months)	0,25
Size (gen)	Size of the Generators in PM (person /months)	2,00
AA	Assessment and Assimilation	4,00

Tabla 10-23 Parámetros del Modelo.

10.3.4. Modelo numérico

Con los parámetros obtenidos y, a partir del Modelo matemático, tenemos que, el coste, en personas / mes, de construir N productos, sin hacer uso de la línea de productos vendrá dado por la expresión siguiente:

$$\mathbf{PMNR(N) = N\ x\ (A\ x\ esize^{E}\ x \prod_{i=1}^{n} eEM_i) = N\ x\ 1,78}$$

Ecuación 10-11. Coste numérico de construir N productos, sin hacer uso de la SPL.

Haciendo uso de nuestra línea de productos, el coste para construir N productos es:

$$\mathbf{PMR(N) = PMR(1) + (N-1)x\ \frac{AA}{100\ x\ (\sum_{i=1}^{G}[A\ x\ gsize^{B}\ x \prod_{i=1}^{n} gEM_i])}}$$

Ecuación 10-12. Coste de construir N productos, utilizando la SPL.

Como:

$$\mathbf{PMR(1) = PMNR(1) + \sum_{i=1}^{G} A\ x\ gsize^{B}\ x \prod_{i=1}^{n} gEM_i\ = 14,21}$$

Ecuación 10-13. Coste numérico de construir 1 producto, utilizando la SPL.

Por tanto, tenemos que:

$$\mathbf{PMR(N) = 13,67 + N\ x\ 0,54}$$

Ecuación 10-14. Coste numérico de construir N productos, utilizando la SPL.

Si recordamos la Ecuación Inicial del Modelo COPLIMO-EODAM (Ecuación 9-1), y a partir de las ecuaciones 9-11 y 9-13, obtenemos que:

$$|\mathbf{PLS(1)}| = |\mathbf{PMNR(1)} - \mathbf{PMR(1)}| = \mathbf{12,43}$$

Ecuación 10-15. Ecuación Inicial del Modelo para N=1, en valor absoluto.

Y para N productos:

$$\mathbf{PLS(N)} = \mathbf{PMNR(N)} - \mathbf{PMR(N)} = \mathbf{1,24\ x\ N - 13,67}$$

Ecuación 10-16. Ecuación Inicial del Modelo, en valores numéricos.

Pero nuestro interés, como ya hemos mencionado con anterioridad, **se centra en obtener la rentabilidad** (el ROI, Return On Investment).

Desde la ecuación 9-10 y las ecuaciones 9-15 y 9-16 tenemos una expresión de la rentabilidad, en función del número de productos de nuestra SPL:

$$\mathbf{ROI(N)} = \frac{\mathbf{PLS(N)}}{|\mathbf{PLS(1)}|} = \frac{\mathbf{1,24}\ \boldsymbol{x}\ \mathbf{N - 13,67}}{\mathbf{12,43}}$$

Ecuación 10-17. Rentabilidad de nuestra SPL en función del nº de productos.

10.4. Valoración Final

Para valorar si es rentable nuestra SPL, podemos calcular el número mínimo de productos a obtener para lograr dicha rentabilidad. Despejando desde la ecuación 9-17 obtenemos:

$$\mathbf{ROI(N)} \geq \mathbf{0} \Rightarrow N > 11$$

Ecuación 10-18. Condición de Rentabilidad de nuestra SPL.

Esto significa que **si tenemos que construir más de 11 productos nuestra SPL será productiva.**

Como ya hemos analizado con anterioridad, el número de productos que podemos obtener con nuestra SPL depende de varios factores como los requisitos, los elementos, la dimensión de la base de datos, etc. Hemos obtenido en el capítulo siete unas métricas de nuestro dominio que ponían de manifiesto la dimensión del mismo y que tenía un **alcance de miles de productos, solamente considerando el ámbito de los requisitos del dominio.**

Con el Framework de Desarrollo construido somos capaces de generar cualquier producto del dominio, con las limitaciones expuestas en lo que se refiere al número de requisitos analizados, el número de funcionalidades y otros factores como los elementos del dominio. **Es más que evidente la potencialidad y rentabilidad de la solución ofrecida por nuestra SPL.**

Durante la fase de Flexibilización del Ejemplar, incluimos una fase denominada *"Análisis de Rentabilidad"*, tal y como se destaca en la figura adjunta:

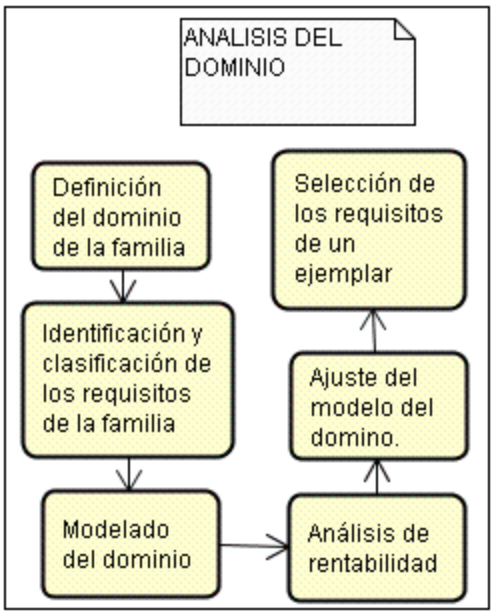


Figura 10-1 Análisis de Rentabilidad en EODAM.

En este punto del proceso habíamos obtenido un modelo inicial del dominio, pero para llevar a cabo el análisis de rentabilidad hemos comprobado que precisamos conocer ciertos aspectos que en este punto no conocíamos, como el modelo completo del dominio o el coste del desarrollo de un ejemplar o de los generadores.

No obstante, consideramos importante mantener este proceso en esta fase de la metodología EODAM, porque esto nos permite conocer el alcance inicial del dominio bajo estudio. Por ejemplo, si en este punto del proceso comprobamos que el dominio tiene un alcance muy limitado, podríamos reconsiderar construir nuestra SPL.

Como ya fue analizado en el capítulo siete, en la exposición de las métricas del dominio, **el número final de productos de nuestro dominio varía en función de la dimensión de nuestra CMDB y de los interfaces que con ella interactúan**.

La rentabilidad de nuestra SPL también tendrá una dependencia de estos factores. Si realizamos un ajuste a una curva exponencial, la relación entre el número de entidades y el número de productos se refleja con la siguiente expresión matemática:

$$N = 0,93 * e^{2,091*X}$$

Ecuación 10-19. Nº de productos en función del número de entidades.

Donde N es el nº de productos de nuestro dominio y X el número de entidades de la CMDB bajo la notificación de cambios. Es decir, que **el nº de productos posibles crece de forma exponencial con el nº de entidades consideradas**.

Para que N sea mayor de 11 y nuestra línea de productos sea rentable basta con que X sea igual o mayor de 2. Es decir, **con solamente dos entidades nuestra línea de productos ya sería productiva**, **con independencia de los requisitos, las funcionalidades o los suscriptores**, que son el resto de ámbitos que influyen en el alcance de nuestro dominio.

A continuación se muestra la gráfica ya analizada en el capítulo siete, con la curva de la ecuación 10-19 ajustada.

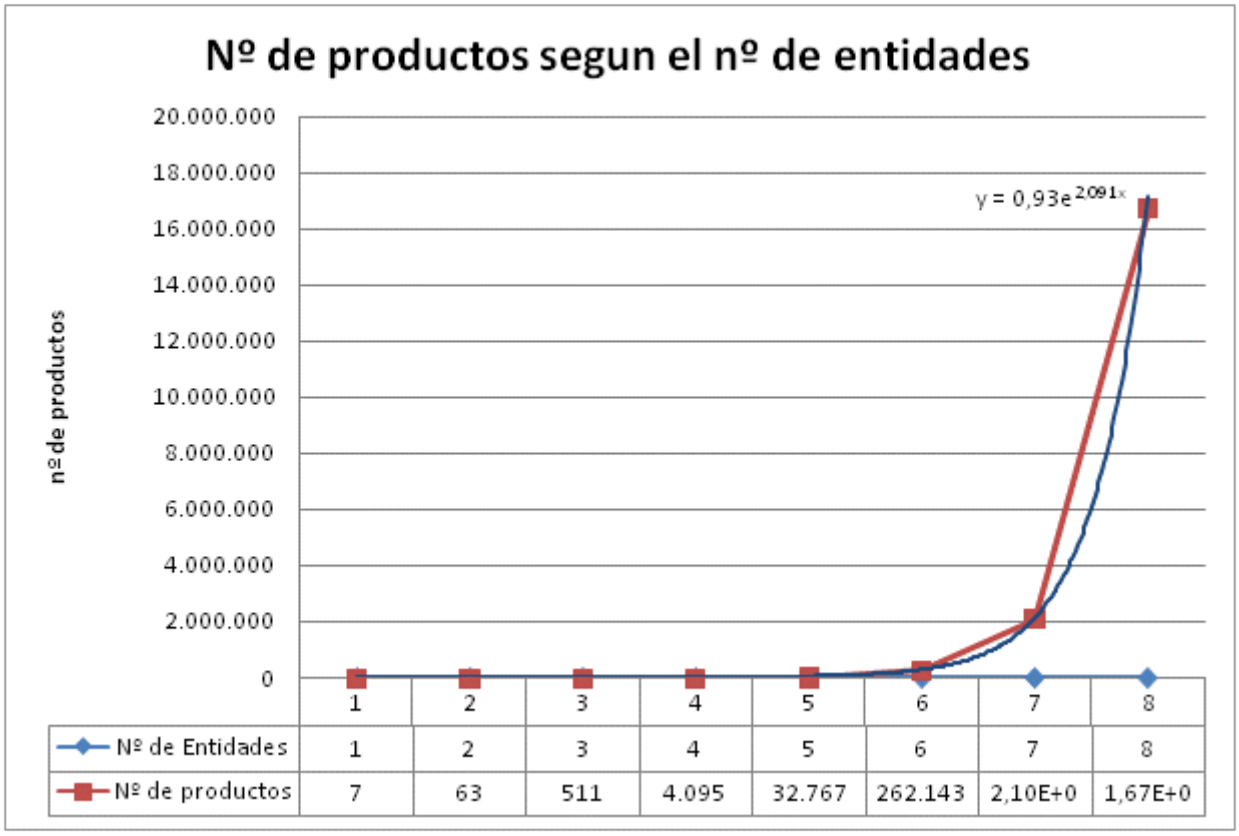


Figura 10-2 Gráfica del nº de productos según el nº de entidades.

Con respecto al nº de suscriptores, la relación también es de tipo exponencial, tal y como se muestra en la gráfica 10-3. En este caso la ecuación que se obtiene es:

$$N = 0,67 * e^{1,189*X}$$

Ecuación 10-20. Nº de productos en función del número de suscriptores.

Donde N es el nº de productos de nuestro dominio y X el número de suscriptores al servicio de notificación de cambios. Es decir, que **el nº de productos posibles crece de forma exponencial con el nº de suscriptores considerados.**

En este caso, solo considerando el ámbito de los suscriptores, para que N sea mayor o igual que 11 basta con que X sea mayor o igual que 3. **Con solo tres suscriptores ya sería productiva nuestra línea de productos, independientemente del resto de ámbitos de influencia** (requisitos, funcionalidades o elementos).

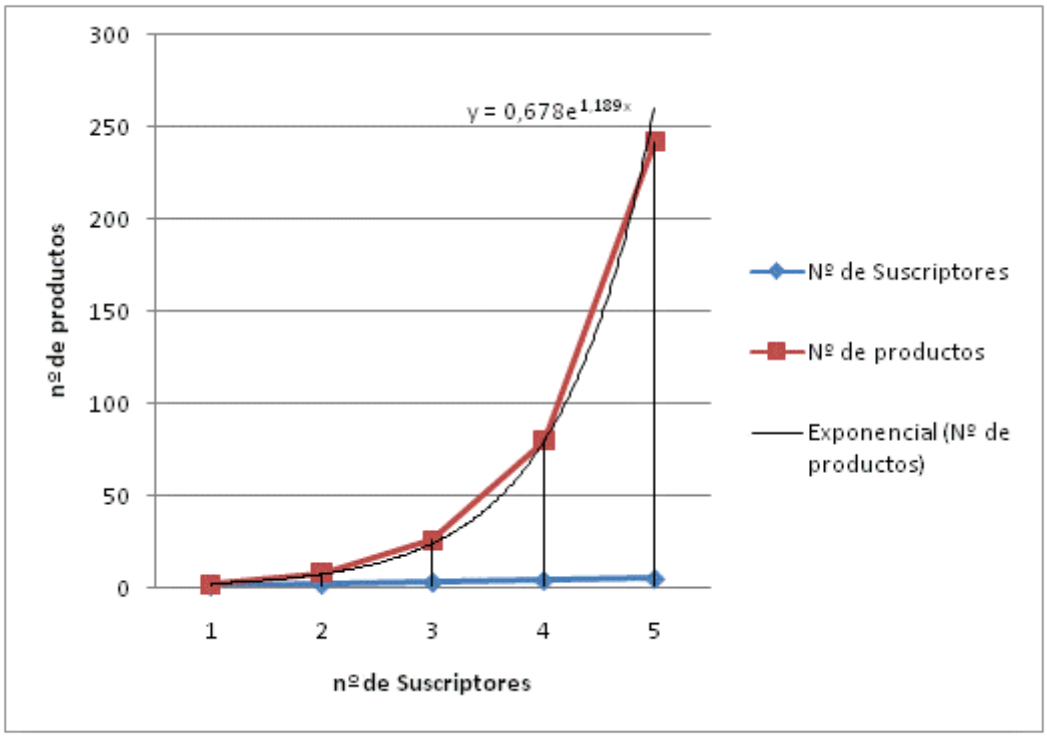


Figura 10-3 Gráfica del nº de productos según el nº de suscriptores.

Todas estas métricas nos demuestran la rentabilidad y el gran alcance de la línea de productos construida.

11.Conclusiones y Futuros Trabajos

"Entre dos explicaciones,
elige la más clara;
entre dos formas,
la más elemental;
entre dos expresiones,
la más breve."
(Eugeni d' Ors)

11.1. Resumen y Conclusiones

Durante el desarrollo de la Tesis se plantearon una serie de objetivos:

Objetivo 1. *Diseñar una metodología de construcción de líneas de productos software para desarrollos de bases de datos.*

Objetivo 2. *Integrar la metodología de desarrollo con las buenas prácticas de gestión de proyectos.*

Objetivo 3. *Adaptar el proceso metodológico para la construcción de bases de datos de gestión de la configuración (CMDB).*

Objetivo 4. *Proponer una normativa para el diseño de una CMDB.*

Objetivo 5. *Estudiar el Dominio de la notificación de cambios en bases de datos y su aplicación a una CMDB, siguiendo las buenas prácticas de ITIL.*

Objetivo 6. *Analizar con detalle el dominio, determinando sus requisitos y todos los elementos de análisis necesarios. Este dominio constituirá un caso de estudio, con un alcance determinado, sobre el cual hacer uso de la metodología y normativa planteadas.*

Objetivo 7. *Realizar una propuesta para documentar la variabilidad en la Ingeniería de Dominio, realizando la correspondiente adaptación a nuestro caso de estudio.*

Objetivo 8. *Construir un Framework de desarrollo y pruebas para la construcción de líneas de productos y aplicarlo al caso de estudio.*

Objetivo 9. *Desarrollar un modelo de costes que nos permita medir la rentabilidad de la solución propuesta.*

Para cumplir el **primer objetivo**, se ha expuesto una metodología original, denominada EODAM (*Exemplar Oriented Database Methodology*), que ofrece un proceso de desarrollo software de líneas de productos para bases de datos, orientado a la construcción de productos a partir de un ejemplar del dominio.

Con respecto al **segundo objetivo**, la metodología se ha integrado con las buenas prácticas de gestión de proyectos y se ha propuesto una adaptación de este proceso para construir desarrollos de Bases de Datos de Gestión de la Configuración (CMDB) (**tercer objetivo**).

En lo que se refiere al **cuarto objetivo**, se ha realizado una propuesta de estandarización para una CMDB, incluyendo normas para el diseño conceptual, lógico y físico. Esta propuesta puede ser extendida a cualquier otra tipología de base de datos, realizando la correspondiente adaptación.

El dominio específico de la notificación de cambios en bases de datos y su aplicación a una CMDB ha sido estudiado en detalle, analizando los requisitos principales, estudiando los procesos que intervienen en el dominio e identificando los interfaces que interactúan con una CMDB. Estos trabajos han cubierto los **objetivos quinto y sexto.**

Para el **séptimo objetivo**, se ha realizado un estudio pormenorizado de los diferentes métodos y herramientas disponibles en varias líneas de investigación sobre la documentación de la variabilidad en la Ingeniería de Dominio, detallando un conjunto de técnicas y herramientas novedosas que ofrecen una serie de mejoras sustanciales, y permiten obtener métricas y funcionalidades de gran relevancia para el desarrollo de líneas de productos.

El cumplimiento del **octavo objetivo** ha supuesto el desarrollo de un caso de estudio específico, una CMDB con requisitos de notificación de cambios. Para realizar este desarrollo se han estudiado las diferentes propuestas existentes en el mercado para el diseño y construcción de una CMDB. Con el objeto de desarrollar los productos del dominio bajo estudio, se ha construido un Framework de Desarrollo y de Pruebas que nos permiten obtener, de forma automática, los productos del dominio y las métricas más relevantes del mismo.

Finalmente, con respecto al **último objetivo**, la tesis se ofrece un modelo económico para la línea de productos presentada. Este modelo tiene como objetivo realizar un estudio de rentabilidad de la solución ofrecida. El modelo puede ser extendido a otros enfoques de programación generativa y/o de desarrollo de líneas de productos software y a otros casos de estudio bajo la metodología EODAM.

Este modelo económico pone de manifiesto la gran rentabilidad de la solución ofrecida por la propuesta de desarrollo basada en EODAM y con el soporte de los Frameworks de desarrollo llevados a cabo.

11.2. Trabajos Futuros

Pueden considerarse varias líneas de trabajo futuro:

- **Extender el dominio bajo estudio, incorporando nuevos requisitos**. Para ello podríamos **realizar una extensión del DSL** (Lenguaje de Especificación del Dominio) o **construir un transformador más general**.

- **Extender la portabilidad de la solución**, incluyendo diversas bases de datos relevantes del mercado y productos de soporte a la CMDB. Esta actividad podría incluir **ampliar el DSL** para ampliar la solución a más productos. Podría existir un DSL común para las diferentes BBDD y, dependiendo de la CMBD seleccionada, habilitar o no determinadas opciones. En este caso, la extensión del DSL sería más bien una evolución en cuanto a nuevos requisitos, como en el caso anterior. También podríamos **construir un transformador más general** para incluir más productos.

- **Aplicar el proceso EODAM sobre un dominio diferente.** Por ejemplo, realizar una **adaptación a otra tipología de base de datos**, como por ejemplo, una base de datos de gestión documental.

- **Construir un entorno de desarrollo integrado (IDE)** que integre las herramientas de soporte a la gestión de la variabilidad, los algoritmos de generación de productos, la obtención del DSL y las pruebas.

- **Ampliar la Normativa expuesta para otras Bases de Datos y desarrollar aplicaciones que permitan la detección automática del cumplimiento de la Normativa.**

- Implementar, como **mejora del Framework de Desarrollo**, la **generación de documentación** que permita elaborar informes sobre las sustituciones, producciones y generaciones realizadas por los transformadores.

- **Ampliar el dominio bajo estudio a otros escenarios donde se utilizan los mismos mecanismos que en la notificación de cambios**, como la replicación de datos, que utiliza la gestión avanzada de colas.

- **Desarrollar mejoras sobre EFL**, incluyendo una **gestión avanzada de errores** en la detección de colisiones y una **mejora en las prestaciones a la hora de implementar las sustituciones**.

- **Adaptar el Modelo de Costes a otros escenarios** de construcción de líneas de productos software.

- Además de las publicaciones que se han realizado durante la Tesis, con los resultados obtenidos y las conclusiones se plantea como otro trabajo futuro adicional la **publicación de algunos artículos.**

Bibliografía

[A. García et al. 2006] Alessandro Garcia, Uirá Kulesza , Cláudio Sant'Anna, Christina Chavez and Carlos J. P. de Lucena. *Aspects in Agent-Oriented Software Engineering: Lessons Learned.* Agent-Oriented Software Engineering VI. pp 231-247. ISBN 978-3-540-34097-3. 2006.

[Adriana et al. 2000] *Orientación a objetos y orientación a agentes: una propuesta de unificación.* Adriana Giret, Luca Cernuzzi y Oscar Pastor. http://www.dsic.upv.es/~einsfran/papers/22-CLEIOO-Method.pdf

[Aho et al. 1990] Aho, A. V.; Sethi, R.; Ullman, J. D. *Compiladores, Principios, técnicas y herramientas.* Addison-Wesley Iberoamericana, 1990.

[Allen 2008] Mark K. Allen. *The Elysian Fields of Information Technology. People Path to Technological Value*. pp 225-230. 2008

[Alloy 2010] Página Web de la *herramienta Alloy Analyzer.* Link actualizado a Junio 2010. http://alloy.mit.edu/alloy4/

[ANTLR 2010] *ANTLR, ANother Tool for Language Recognition.* 2010 http://www.antlr.org/

[AQ Oracle Java] *Oracle Advanced Queuing and Java.* Link actualizado a Junio 2010.http://www.akadia.com/services/ora_advanced_queueing.html

[AQ Oracle] *Introduction to Oracle Advanced Queuing.* Link actualizado a Junio 2010.http://download.oracle.com/docs/cd/B10500_01/appdev.920/a96587/qintro.htm

[AQ SQLS] *Creación de Colas en SQL SERVER.* Link actualizado a Junio 2010. http://msdn.microsoft.com/en-us/library/ms190495.aspx

[ASD 2000] James A. Highsmith. *Adaptive software development: a collaborative approach to managing complex systems.* Dorset House Pub., 2000.

[Atkinson et al. 2002] C. Atkinson et al., *Component-based product line engineering with UML*. Addison-Wesley, London, New York, 2002.

[B. Gonzalez et al. 2004] Bruno González-Baixauli, Miguel A. Laguna, Julio Cesar Sampaio do Prado Leite. *Análisis de Variabilidad con Modelos de Objetivos.* Workshop em Engenharia de Requisitos 2004.

[B. Gonzalez et al. 2004] Bruno González-Baixauli, Miguel A. Laguna, Yania Crespo. *Product Line Requirements based on Goals, Features and Use cases.* International Workshop on Requirements Reuse in System Family Engineering (IWREQFAM), pp 4-7 – june 2004.

[Bachmann and Clements 2005] Felix Bachmann, Paul C. Clements. *Variability in Software Product Lines. Technical Report.* CMU/SEI-2005-TR-012. September 2005.

[Baskarada 2009] Saša Baškarada. *IQM-CMM: Information Quality Management Capability Maturity Model*. pp 27, 67, 68. 2009.

[Batini 2004] Carlo Batini. *Diseño conceptual de bases de datos.* pp 65- 99 Ediciones Díaz de Santos, 2004

[Batory 2005] D. S. Batory. *Feature Models, Grammars, and Propositional Formulas.* Software Product Line Conference (SPLC). September 2005.

[Batory et al. 2006] D. Batory, D. Benavides and A. Ruiz-Cortés. *Communications of the ACM (Special Section on Software Produc Lines). Automated Analyses of Feature Models: Challenges Ahead.* 2006.

[Bayer et al. 1999] Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., and DeBaud, J. 1999. *PuLSE: a methodology to develop software product lines.* In Proceedings of the 1999 Symposium on Software Reusability (Los Angeles, California, United States, May 21 - 23, 1999). SSR '99. ACM Press, New York, NY, 122-131.

[Benavides 2010] Benavides, D.. Página web de la *herramienta FAMA Tool Suite* para la gestión de la variabilidad. Link actualizado a Junio 2010. http://www.isa.us.es/fama/

[Benavides et al. 2006] Benavides, D.; Ruiz-Cortes A.; Trinidad, P.; Segura, S. *A survey on the automated analyses of feature models. J*ornadas de Ingenieria del Software y Bases de Datos (JISBD06), 2006

[Benavides 2007] Benavides, D. *On the automated analysis of software product lines using feature models. a framework for developing automated tool support.* PhD Dissertation. University of Seville, Spain, June 2007.

[Benavides et al. 2007] D. Benavides, S. Segura, P. Trinidad and A.Ruiz-Cortés. First International Workshop on Variability Modelling of Software-intensive Systems (VAMOS). *FAMA: Tooling a Framework for the Automated Analysis of Feature Models.* January 2007.

[Berre and Parrain 2008] Le Berre, D.; Parrain, A. *On SAT Technologies for dependency management and beyond.* First Workshop on Analyses of Software Product Lines (ASPL 2008).

[Bertran et al. 2010] Bertrand David, Olivier Champalle, Guillaume Masserey and René Chalon. *From Task Model to Wearable Computer Configuration. Task Models and Diagrams for User Interface Design.* pp 261-266. Springer Berlin / Heidelberg 2010.

[Biglever 2009] *BigLever Software, Inc. Gears.* Link actualizado a Junio 2010. http://www.biglever.com/index.html

[Blokdijk and Menken 2008] Gerard Blokdijk, Ivanka Menken. *It Service Continuity Management and Disaster Recovery Best Practice Handbook about ITIL* Theo. Lulu.com, 2008.

[BMC CMDB 2010] *BMC Atrium CMDB.* Links actualizados a Junio 2010. http://www.bmc.com/products/brand/bmc-atrium-0726.html http://www.bmc.com/products/product-listing/atrium-cmdb.html

[BMCAD 2010] *BMC Discovery Solution*. BMC Software, 2010. http://documents.bmc.com/products/documents/12/29/101229/101229.pdf

[Boehm et al. 2000] Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D. J., Steece, B.: *Software Cost Estimation with COCOMO II*; Prentice Hall, 2000.

[Boehm et al. 2004] Boehm, B., Brown, A. W., Madachy, R., Yang, Y. *A software product line life cycle cost estimation model; International Symposium on Empirical Software Engineering;* 156-164; 2004.

[Bon ITSM 2008] Jan Van Bon. *IT Service Management Global Best Practices,* Volume 1. Page 339. Van Haren Publishing, 2008

[Booch et al. 1998] Grady Booch (Author), James Rumbaugh (Author), Ivar Jacobson. *The Unified Modeling Language User Guide* (Hardcover). ISBN 978-0201571684, 1998.

[Booch et al. 2005] Grady Booch, James Rumbaugh, Ivar Jacobson. *The Unified Modeling Language user guide.* Addison-Wesley, 2005.

[Bosch 2005] Jan Bosch. *Software variability: process and management. Software Process: Improvement and Practice.* October 2005.

[Bresciani et al. 2004] Bresciani P., Giorgini P., Giunchiglia F., Mylopoulos J., Perini A. *TROPOS: An agent-oriented software development methodology.* Journal of Autonomous Agents and Multi-Agent Systems 8(3): 203–236. 2004.

[Broek et al. 2008] Broek, P.; Galvao, I.; Noppen, J. *Elimination of Constraints from Feature Trees.* First Workshop on Analyses of Software Product Lines. ASPL 2008.

[Captain Feature 2010] Página web de la *herramienta Captain Feature* para la gestión de la variabilidad. Link actualizado a Junio 2010. https://sourceforge.net/projects/captainfeature/

[CC 2010] *Common Criteria.* Link actualizado a Septiemrbe 2010. http://www.commoncriteriaportal.org

[Celko 1994] Celko, J. *The Great Key Debate, in DBMS*, Vol. 7, No. 10. September 1994.

[Chastek et al. 2002] Chastek, Gary; Donohoe, Patrick; & McGregor, John D. *Product Line Production Planning for the Home Integration System Example* (CMU/SEI-2002-TN-029, ADA405846). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.

[Chen et al. 2005] Kun Chen, Wei Zhang, Haiyan Zhao, Hong Mei. *An Approach to Constructing Feature Models Based on Requirements Clustering* 13th IEEE International Conference on Requirements Engineering (RE'05) pp. 31-40. 2005.

[CIM 2010] *Common Intormation Model.* Versión 2.26.0. Julio 2010. http://www.dmtf.org/standards/cim

[CIM Sc. Sp. 2010] *Common Intormation Model.*
Esquema del Modelo:
http://dmtf.org/standards/cim/cim_schema_v2260
Especificaciones del Modelo:
http://www.dmtf.org/sites/default/files/standards/documents/DSP0219_1.0.0.pdf

[Clark et al. 2007] D. Clark et al. *The Federated CMDB Vision: A Joint White Paper* from BMC, CA, Fujitsu, HP, IBM, and Microsoft, Version 1.0. Technical report; 2007.

[Cleaveland 2001] Cleaveland, J. C. *Program Generators with XML and Java*. Section 2.9. A Balancing Act.. Prentice Hall, 2001.

[Clements and Northrop 2001] Clements, P., Northrop, L.: *Software Product Lines: Practices and Patterns;* Addison-Wesley; 2001.

[Clements and Northrop 2002] Clements, P., Northrop, L.: *Software Product Lines*; Addison-Wesley; 2002.

[Clements et al. 2002] Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; Sta®ord, J. *Documenting Software Architectures: Views and Beyond.* SEI Series in Software Engineering. Addison-Wesley Professional. October 6, 2002.

[Clements et al. 2005] Clements, P., McGregor, J., Cohen, S. *The structured intuitive model for product line economics*; Technical report; CMU/SEI-2005-TR-003; 2005.

[CMM 1993] Mark C. Paulk, Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis and Marilyn Bush. *Key Practices of the Capability Maturity ModelSM, Version 1.1*. Technical Report CMU/SEI-93-TR-025 ESC-TR-93-178, February 1993.

[Cobit 4.1 2007] *The IT Governance Institute. Cobit 4.1.* Publisher: ISACA, 2007.

[Colin 2010] Colin Bently. *Prince2: A Practical Handbook. Project Mandate. pp 272-276.* Butterworth-Heinemann, 2009

[Colmerauer and Roussel 1992] Alain Colmerauer Philippe Roussel. *La naissance de Prolog.* juillet 1992. Link actualizado a Marzo 2010. http://alain.colmerauer.free.fr/ArchivesPublications/HistoireProlog/24juillet92.pdf

[Cota 1994] Cota A. 1994 *"Ingeniería de Software". Soluciones Avanzadas.* pp. 5-13. Julio de 1994.

[Covadonga 1987] María Covadonga Fernández Baizán. *El modelo racional de datos: De los fundamentos a los modelos deductivos.* Página 131. Ediciones Díaz de Santos, 1987

[Coz et al. 2008] Coz, J.R.; Heradio, R.; Cerrada, J.A.; Lopez, J.C. *A generative approach to improve the abstraction level to build applications based on the notification of changes in databases.* 10th International Conference on Enterprise Information Systems. Barcelona, Spain, June 12 – 16, 2008.

[CPP.U. 2010] *CppUnit. Unit testing framework for C++*. Link actualizado a Junio 2010. http://cppunit.sourceforge.net

[Czarnecki 2000] *Generative Programming - Methods, Tools, and Applications* by Krzysztof Czarnecki and Ulrich W. Eisenecker Addison-Wesley, June 2000

[Czarnecki et al. 2005] K. Czarnecki, S. Helsen, and U. Eisenecker. *Formalizing Cardinality-based Feature Models and their Specialization.* Software Process Improvement and Practice, 2005 10(1).

[Czarnecki and Antkiewicz 2005] Czarnecki, K.; Antkiewicz, M. *Mapping Features to Models: A Template Approach Based on Superimposed Variants.* GPCE 2005, pp. 422-437. 2005.

[Czarnecki et al. 2000] Czarnecki, K.; Eisenecker, U. *Generative Programming: Methods, Tools, and Applications.* Addison-Wesley, 2000.

[Czarnecki et al. 2005, 2] K. Czarnecki, S. Helsen and U. Eisenecker. *Staged Configuration Through Specialization and Multi-Level Configuration of Feature Model*, Software Process Improvement and Practice, 10(2), 2005.

[Czarnecki et al. 2006] Czarnecki, K.; Pietroszek, K. *Verifying feature-based model templates against well-formedness OCL constraints.* GPCE 2006, pp: 211-220. 2006.

[Dallal and Sorenson, 2005] Dallal, J. A.; Sorenson, P. *Reusing class-based test cases for testing object-oriented framework interface classes.* Journal of Software Maintenance and Evolution: Research and Practice. Volume 17, Issue 3, Pages: 169-196. June 2005.

[Daniel et al. 2010] Daniel Sinnig , Maik Wurdel , Peter Forbrig , Patrice Chalin and Ferhat Khendek. *Practical Extensions for Task Models. Task Models and Diagrams for User Interface Design.* pp 42-55. Springer Berlin / Heidelberg 2010.

[Date 1991] Date, C. J. *Don't Encode Information into Primary Keys!,* Relational Database Writings 1989-1991. Reading, Mass: Addison-Wesley Publishing 1991.

[Date 2001] J. C. Date, *Introducción a los sistemas de bases de datos.* Página 112. Pearson Educación, 2001

[David Fdez. et al. 2009] Fernandez, D.; Heradio, R.; Cerrada, J. A. I*nferring Information from Feature Diagrams to Product Line Economic Models.* 13th International Software Product Line Conference, 2009. http://portal.acm.org/citation.cfm?id=1753235.1753242

[David y Heradio 2010] David Fernández Amorós y Rubén Heradio Gil. *Algorithm to calculate, from a Feature Diagram, the total number of products of a SPL, the SPL homogeneity and the commonality of the SPL requirements.* Link actualizado a Septiembre 2010. http://www.issi.uned.es/miembros/pagpersonales/ruben_heradio/spl.htm

[Davies and Woodcock 1996] Jim Davies and Jim Woodcock. *Using Z: Specification, Refinement and Proof.* Prentice Hall International Series in Computer Science. ISBN 0-13-948472-8. 1996

[DDM 2010] *HP key strategies for successful customer CMDB implementations white paper.* HP 2010. https://h10078.www1.hp.com/cda/hpdc/navigation.do?action=downloadPDF&caid=9554&cp=54_4000_100&zn=bto&filename=4AA1-5466ENW.pdf
HP UCMDB and DDM 9.00 English SW Evaluation https://h10078.www1.hp.com/cda/hpdc/navigation.do?action=downloadBinStart&caid=45795&cp=54_4000_100&zn=bto&filename=T8349DAE

[DEF73 2002] Orden DEF 73/2002. *Política de seguridad para la protección de la información almacenada, procesada o transmitida por sistemas de información y telecomunicaciones.* 2002.

[Deursen et al. 2001] Van Deursen, A.; Klint, P. *Domain specific language design requires feature descriptions.* Journal of Computing and Information Technology (2001).

[Deursen and Klint 2001] A. van Deursen and P. Klint. *Domain-specific language design requires feature descriptions.* Journal of Computing and Information Technology, 2001.

[Dinnus y Pohl 2005] Dinnus, C., Pohl, K.: *Software Product Line Engineering;* chapter Experiences with Software Product Line Engineering, 413- 434; Springer Berlin Heidelberg; 2005.

[DSLT 2010] Página web de la *herramienta DSL Tools* de Microsoft. http://code.msdn.microsoft.com/DslTools

[Eckel 2003] Eckel, B. *Thinking in Patterns. Revisión 0.9, 5-20-2003. Capítulo titulado Pattern refactoring*. http://www.mindview.net.

[Edwards 2001] Edwards, S. H. *A framework for practical, automated black-box testing of component-based software*. SW Testing, Verification and Reliability. Volume 11, Issue 2, Pages: 97-111. 2001.

[EFL 2007] *Ruby implementations of EFL.* 2007
RubyForge: http://rubyforge.org/projects/efl/ .
Ruby Application Archive: http://raa.ruby-lang.org/project/efl/ .

[Eriksson 2006] Magnus Eriksson. *An Approach to Software Product Line Use Case Modeling.* ISBN 91-7264-023-5. Print & Media, Umea University 2006.

[Eriksson et al. 2005] Eriksson, M.; Borstler, J.; Borg, K. *The PLUSS Approach - Domain Modeling with Features, Use Cases and Use Case Realizations.* Software Product Lines, 9th International Conference. pp. 33-44. SPLC 2005.

[Erwin 2009] *Guía de Implementación de la Modelización de Bases de Datos con Erwin.* r7.3.8 SP2 Edition. 2009.
http://support.ca.com/cadocs/0/e002953e.pdf

[ESPAPS-CAFE] Página Web del *proyecto ESAPS en el European Software Institute(ESI):* http://www.esi.es/esaps/overview.html
Página Web del proyecto CAFE en el European Software Institute(ESI): http://www.esi.es/Cafe/index.html

[Ewal and Hangs 2002] Ewald Geschwinde, Hans-Jürgen Schönig. *PostgreSQL developer's handbook.* Sams Publishing, 2002

[Feature Modeling 2010] Página web de la *herramienta Feature Modeling,* una herramienta complementaria a ECLIPSE para la gestión de la variabilidad. Link actualizado a Junio 2010. http://gsd.uwaterloo.ca/fmp

[FMSHAKER 2010] Héctor J. Pérez Morago, Rubén Heradio Gil. *Análisis Automático de Diagramas de Características.* Proyecto Fin de Carrera. ETSI. Informática. UNED. 2010.

[FODA 1990] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute (Carnegie Mellon), Pittsburgh, PA 15213. 1990

[FOMDD 2007] S. Trujillo. D.S. Batory, Oscar Díaz. *Feature Oriented Model Driven Development: A Case Study for Portlets.* International Conference on Software Engineering. Proceedings of the 29th international conference on Software Engineering. 2007.

[FOP 2007] S. Trujillo, D. Batory, O. Diaz. *Feature Oriented Model Driven Development: A Case Study for Portlets*, International Conference on Software Engineering, 2007.

[FORM 1998] Kang, K. C., Kim, S., Lee, J. y Kim, K. *FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures.* Annals of Software Engineering, 5:143-168. 1998.

[Fowler 2005] Fowler, M. *Language Workbenches: The Killer-App for Domain Specific Languages?* June 2005. http://www.martinfowler.com/articles/languageWorkbench.html

[Friedl 2002] Friedl, J. *Mastering Regular Expressions*. O'Reilly, 2002.

[Galindo et al. 2006] Galindo J., Urrutia A., Piattini M. *Fuzzy Databases: Modeling, Design and Implementation.* Idea Group Publishing Hershey, USA, 2006.

[Gartner 2006] Ron J. Colville. *Gartner RAS Core Research Note G00137125, Gartner on CMDB.* March 13, 2006.

[GEMS 2010] Página web de la *herramienta GEMS (Generic Eclipse Modeling System).* Link actualizado a Junio 2010. http://sourceforge.net/projects/gems

[Gerard 2008] Gerard Blokdijk. *CMDB and Configuration Management Process, Software Tools Creation and Maintenance, Planning, Implementation Guide.* pp. 63-68. Lulu.com, 2008.

[Ghosh et al. 2006] France, R. B.; Kim, D.-K.; Ghosh, S.; Song, E. *A UML-based pattern specification technique.* IEEE Transactions on Software Engineering, Volume 30, Issue 3, March 2004. Page(s):193 – 206.

[GMF 2010] Página web del *proyecto GMF(Graphical Modeling Framework).* Link actualizado a Junio 2010. http://www.eclipse.org/gmf/

[Gomaa 2004] Gomaa, H. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures.* Addison-Wesley 2004

[Graph 2010] *Herramientas Graph visualization.* Proyecto Open Source para la representación gráfica de diagramas. Link actualizado a Junio 2010. http://www.graphviz.org/

[Greenfield and Short 2004] Greenfield, J.; Short, K. *SW Factories: Assembling Applications with Patterns, Models, Framworks, and Tools.* Wiley, 2004.

[GREP 2009] *Proyecto GREP (Global Regular Expression Print).* Free Software Foundation, Inc. 2009. http://www.gnu.org/software/grep/

[Griss, M. L. 1996] Griss M. L. *Domain Engineering And Variability In The Reuse Driven Software.* (context), 1996.

[Guía ITIL 2008] *Gestión de servicios ti basado en ITIL: GUIA de bolsillo.* ISBN 9789087531065. Gestión de la Disponibilidad y de la Capacidad. pp 92 – 95. Van Haren Publishing, 2008.

[Gupta et al. 2008] Rajeev Gupta, J Hima Prasad and Mikesh Mohanla. *Automating ITSM Incident Management Process.* International Conference on Autonomic Computing, IEEE 2008.

[Gurp et al. 2001] Van Gurp, J.; Bosch, J.; Svahnberg, M. *On the notion of variability in software product lines.* Working IEEE/IFIP Conference on Software Architecture (WICSA 01), 2001.

[Haziri and Burgess 2009] Fitim Haziri, Mark Burgess. *Design of a CMDB with integrated knowledge management based on Topic Maps.* Network and System Administration Oslo University College. May 18, 2009.

[Hedeman et al. 2006] Bert Hedeman, Gabor Vis van Heemst, Hans Fredriksz. *Project Management Based on Prince2. Quality Review Technique.* pp 156 – 163. ISBN 9789077212837. Van Haren Publishing, 2006.

[Hemakumar 2008] Hemakumar, A. *Finding Contradictions in Feature Models.* First Workshop on Analyses of Software Product Lines. ASPL 2008.

[Henderson 2005] Henderson-Sellers, B., & Giorgini, P. *Agent-oriented methodologies.* Idea Group Publishing. 2005.

[Heradio 2007] Heradio Gil, Rubén. *Metodología de desarrollo de software basada en el paradigma generativo. Realización mediante la transformación de ejemplares*. Ph. D. Thesis, UNED, Spain, 2007. http://www.issi.uned.es/miembros/pagpersonales/ruben_heradio/rheradio_english.html

[Heradio et al. 2005] Heradio Gil, R., Estívariz J. F.., Abad, I., Cerrada Somolinos, J. A.. *Traducción de especificaciones a código ejecutable mediante transformadores de ejemplares.* V Jornadas sobre Programación y Lenguajes (PROLE'05), 185-191, Granada, 14-16. 2005

[Heradio et al. 2008] R. Heradio; J. A. Cerrada; J. C. Lopez; J. R. Coz. *Code Generation with the Exemplar Flexibilization Language.* Workshop on Generative Technologies (ETAPS'08). Budapest, Hungary, April 5, 2008.

[Heymans et al. 2008] Heymans, P.; Schobbens, P.; Trigaux, J.; Bontemps, Y.; Matulevicius, R.; Classen, A. *Evaluating formal properties of feature diagram languages.* Software, IET. Volume: 2, Issue: 3. pp: 281-302.a. June 2008.

[HP CMDB 2010] *HP Universal CMDB software.* Links actualizados a Junio 2010. https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-15-25%5E1059_4000_100 ; Demo: https://h30406.www3.hp.com/campaigns/2008/demo/ucmdb/index.html; Configuration Management System best practices: https://h10078.www1.hp.com/cda/hpdc/fetchPDF.do

[HSQLDB 2010] *Base de Datos Relacional y Open Source: Hyper SQL.* http://www.hsqldb.org/

[HTTP.U. 2010] *Framework for HTTP Test.* Link actualizado a Junio 2010. http://httpunit.sourceforge.net

[Huang and Chen 2005] Huang, C.; Chen, W. Y. *A Semi-Automatic Generator for Unit Testing Code Files Based on JUnit*. IEEE International Conference on Systems, Man and Cybernetics. Vol 1, 10-12 pp 140 - 145. 2005

[In et al. 2006] In, H. P., Baik, J., Kim, S., Yang, Y., Boehm, B. *A quality-based cost estimation model for the product line life cycle;* Communications of the ACM; 49, 12, 85-88; 2006.

[InfoWorld Oct. 2006] The CMDB will serve as the blueprint for how the entire IT infrastructure is structured. InfoWorld Vol. 28, No. 43. ISSN 0199-6649. Page 26. October 2006.

[IS 2006] *Information Technology Security Evaluation Criteria (ITSEC): Preliminary Harmonised Criteria.* Document COM(90) 314, Version 1.2. Commission of the European Communities. Retrieved 2006-06-02 http://www.ssi.gouv.fr/site_documents/ITSEC/ITSEC-uk.pdf

[ISO27002 2005] ISO/IEC 17799:2005. *Information technology - Security techniques - Code of practice for information security management.* 2005

[ISO-8859 1998] *ISO/IEC 8859-1:1998, Information technology — 8-bit single-byte coded graphic character sets — Part 1:* Latin alphabet No. 1. http://en.wikipedia.org/wiki/ISO/IEC_8859-1

[J. GARCIA R. 2008] *La gestión de la configuración y la gestión de Activos como una gestión del conocimiento.* Revista Española de Innovación, Calidad e Ingeniería del Software. Vol 4. Nº 1. 2008.

[J.U. 2010] *Junit. Resources for Test Driven Developmen in Java.* Link actualizado a Junio 2010.http://www.junit.org

[Jacobson 1998] Jacobson, I. 1998. "*Applying UML in The Unified Process*" Rational Software. Presentación disponible en http://www.rational.com/uml

[James and Paul 2002] James R Groff, Paul N. Weinberg. *SQL: The Complete Reference*, Second Edition. McGraw-Hill, 2002.

[Jan Van Bon SD 2008] Jan Van Bon. *Service Design Based on ITIL V3: A Management Guide Best Practice.* Van Haren Publishing Edition, 2008.

[Jan Van Bon SO 2008] Jan Van Bon. *Service Operation Based on ITIL V3: A Management Guide Best Practice.* Van Haren Publishing Edition, 2008.

[Jan Van Bon ST 2008] Jan Van Bon, Mike Pieper. *Service Transition Based on ITIL V3: A Management Guide Best Practice.* Mike Pieper Edition, 2008.

[Janota 2008] Janota, M. *Do SAT Solvers Make Good Configurators?.* First Workshop on Analyses of Software Product Lines, ASPL. 2008.

[John and Muthig 2002] I. John and D. Muthig. *Product Line Modeling with Generic Use Cases.* SPCL2 Workshop on Techniques for Exploiting Commonality Through Variability Management, August 2002, San Diego, California, 2002

[JUDE 2007] *Java and UML Developers' Environment.* http://jude.change-vision.com/jude-web/index.html

[Ken 1991] Ken S. Brathwaite. *Relational databases: concepts, design, and administration.* McGraw-Hill, 1991.

[Kevin et al. 1999] Kevin E. Kline, Lee Gould, Andrew Zanevsky. *Transact-SQL programming.* O'Reilly Media, Inc., 1999.

[Kleppe et al. 2003] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture-Practice and Promise.* Addison-Wesley, 2003.

[Kousik and Raman 2007] Kousik Sankar Ramasubramaniam and Raman Venkatachar. *Goal-Aligned Requirements Generation.* Advances in Conceptual Modeling – Foundations and Applications. pp 245-254. Springer Berlin / Heidelberg 2007.

[Kruchten 2004] Philippe Kruchten. *The Rational Unified Process.* ISBN: 0321197704. Editorial: Addison-Wesley Professional, 2004.

[Larry 2010] Larry Klosterboer. *Implementing ITIL Configuration Management* , Edition 2. Pearson Education, 2010

[Lee et all 2002] Kwanwoo Lee, Kyo Chul Kang, Jaejoon Lee: *Concepts and Guidelines of Feature Modeling for Product Line Software Engineering.* ICSR 2002: 62-77. 2002.

[Lewis 1994] Lewis G. 1994. *"What is Software Engineering?"* DataPro (4015). pp. 1-10. Feb 1994.

[LGPL 2007] *GNU Lesser General Public License.* June 2007. www.gnu.org/licenses/lgpl.html

[Linden 2002] F. van der Linden; Software Product Families in Europe: *The ESAPS and CAFÉ Projects*, IEEE Software, vol. 19, no. 4, July/August 2002, pp. 41–49

[Lonngren 1998] Lonngren, D.D. *Reducing the cost of test through reuse.* AUTOTESTCON '98. IEEE Systems Readiness Technology Conference. 24-27 Aug. 1998 Page(s):48 – 53. 1998.

[LOPD 1999] *Ley Orgánica 15/1999 de Protección de Datos de Carácter Personal de España.*

[López 2006] R.E. Lopez-Herrejon. *Understanding Feature Modularity, Ph.D. dissertation.* The Department of Computer Sciences, The University of Texas at Austin. September 2006.

[López et al. 2008] *Lopez, J.C.; Heradio, R.; Cerrada, J. A.; Coz J.R.* A first generation software product line for data acquisition systems in astronomy. *SPIE Symposium on Astronomical Telescopes and Instrumentation: Synergies between Ground and Space. Marseille, France, June 23-28, 2008.*

[LSD 2003] Mary Poppendieck, Tom Poppendieck. *Lean software development: an agile toolkit.* The Agile software development series. Addison-Wesley, 2003.

[Manish et al. 2004] Manish Bhide, Ajay Gupta, Mukul Josh and Mukesh Mohania. *Optimal Deployment of Triggers Detecting Events.* Database and expert systems applications. pp 66-74. 15th International Conference, DEXA 2004, Zaragoza, Spain, 2004.

[Mark and George 1999] Mark Levene, George Loizou. *A guided tour of relational databases and beyond.* Page 157. Springer, 1999.

[Martin and Finkelstein 1981] Martin, James and Clive Finkelstein. *Information Engineering, Technical Report, two volumes*, Lancs, UK : Savant Institute, Carnforth. Nov 1981.

[Masarat et al. 2009] Masarat Ayat, Mohammad Sharifi, Sulaimi Ibrahim Shamsul Sahibudin. *"CMDB Implementation Approaches and Considerations in SME/SITU's Companies".* Third Asia International Conference on Modeling & Simulation, 2009.

[Mayrhauser et al 1994] Von Mayrhauser, A.; Mraz, R.; Walls, J.; Ocken, P. *Domain based testing: increasing test case reuse.* IEEE International Conference on Computer Design: VLSI in Computers and Processors. ICCD '94. 10-12 Oct. 1994 Page(s):484 – 491. 1994.

[McGregor 2004] John D. McGregor: *Domain **, in Journal of Object Technology, vol. 3, no. 7, July-August 2004, pp. 71-81. http://www.jot.fm/issues/issue_2004_07/column6

[METAEDITM 2010] Página web de la *herramienta MetaEdit+ Modeler.* Link actualizado a Junio 2010. http://www.metacase.com/mep/

[METRICA3 2010] *Metodología de Planificación, Desarrrollo y Mantenimiento de Sistemas de Información. Metrica Versión 3.* Vigente en 2010. http://www.csi.map.es/csi/metrica3/index.html

[Metzger et al. 2007] Metzger, A.; Pohl, K.; Heymans, P.; Schobbens, P.; Saval, G. *Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis.* Requirements Engineering Conference, 2007. RE'07. 15th IEEE International, pp. 243-253. 2007.

[Michael 1997] Michael, C.C. *Reusing testing of reusable software components.* Proceedings of the 12th Annual Conference on Computer Assurance (COMPASS '97). 'Are We Making Progress Towards Computer Assurance?'. Page(s):97 – 104. June 1997.

[MODELWARE 2006] *Introduction to Model Engineering. 2006.* Se puede encontrar en la sección de material libre del proyecto ModelWare. http://atlanmod.emn.fr/ModelWare/

[MOF 2005] *Microsoft Operations Framework: Capacity Management Service Management Function.* Microsoft Corp, January 2005.

[MOF 2005] Meta Object Facility (MOF) 2.0 *XMI Mapping Specification, v2.1*.Formal/05-09-01. 2005.

[MOF 2006] *The Object Management Group. Meta Object Facility Core Specification version 2.0. OMG* document formal/2006-01-01.

[Morisio et al. 2002] Morisio, M.; Romano, D.; Stamelos, I. Quality, *productivity, and learning in framework-based development: an exploratory case study*. IEEE Transactions on Software Engineering, Volume 28, Issue 9, Sept. 2002. Page(s):876 –888.

[N.U. 2010] *NUnit 2.0.* Link actualizado a Junio 2010. http://www.nunit.org

[NFR 1999] *Non-Functional Requirements in Software Engineering.* Kluwer Academic Publishers, Boston Hardbound, ISBN 0-7923-8666-3 October 1999, 472 pp.

[NFTE Random 2010] David Fernández Amorós y Rubén Heradio Gil. *An Algorithm for the Random Generation of Feature Diagrams.* Página 6. Link actualizado a Septiembre 2010. http://www.issi.uned.es/miembros/pagpersonales/ruben_heradio/spl/user_manual.pdf

[OARW 2010] Página web del *proyecto Open Architectureware.* Link actualizado a Junio 2010. http://www.openarchitectureware.org/index.php

[OCMDB 2009] *One CMDB Documentation, an Open Source CMDB*. December 2009. http://www.onecmdb.org/wiki/index.php?title=Main_Page

[Oliver et al. 2005] Olivier Camp, Joaquim B. L. Filipe, Slimane Hammoudi and Mario Piattini. *Reasoning with Goals to Engineer Requirements.* Enterprise Information Sys. pp 12-20. Springer Netherlands 2005.

[OODA 1995] Maccario P.M. *The OODA (Object Oriented Domain Analysis) methodology*. Workshop "Application of Domain Analysis Techniques to Object Oriented System", 1995.

[Osiatis GV 2010]. *Curso sobre Fundamentos de ITIL. El Proceso de Gestión de Versiones (GV). Link de Julio 2010.* http://itil.osiatis.es/Curso_ITIL/Gestion_Servicios_TI/gestion_de_versiones/vision_general_gestion_de_versiones/vision_general_gestion_de_versiones.php

[Osman et al. 2008] Osman, A.; Phon-Amnuaisuk, S.; Kuan Ho, C. *Knowledge Based Method to Validate Feature Models.* First Workshop on Analyses of Software Product Lines. ASPL. 2008

[Perez y Lara 2006] Francisco Pérez y Juan de Lara. *Hacia la Definición de Lenguajes Específicos de Dominio con Sintaxis Gráfica y Textual.* III Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM'06). Taller dentro de las JISBD'06.

[Peter et al. 2000] Peter Domanski, Peter Domanski & Philip Irvine. *A Practical Guide to Relational Database Design.* Page 45. Diaxon Ltd, 2000.

[Petersen 2002] John V. Petersen. *Absolute beginner's guide to databases.* pp 13,12, 75-76, 286. Que Publishing, 2002.

[PIPES Oracle] *Resúmenes sobre el Mecanismo de PIPES en Oracle.* Links actualizados a Mayo 2010.
http://download.oracle.com/docs/cd/B10501_01/appdev.920/a96612/d_pipe.htm
http://www.orafaq.com/wiki/DBMS_PIPE
http://www.dbasupport.com/oracle/ora9i/DBMS_PIPES.shtml

[PIPES SQLS] *Resumen sobre el Mecanismo de PIPES en SQL - SERVER.* Link actualizado a Mayo 2010.
http://technet.microsoft.com/en-us/library/ms189321.aspx

[PMP 2003] Brent Knapp, Brent W. Knapp. *A Project Manager's Guide to Passing the Project Management Exam. Control and Limits*. pp 143. ISBN 9780972665674. 2003.

[Pohl et al. 2001] K. Pohl, G. Böckle, P. Clements, H. Obbink, and D. Rombach (eds.); *Proceedings Seminar Product Family Development*, University of Essen, Germany, 2001.

[Pohl et al. 2005] Pohl, K.; Bockle, G,; Linden, F. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer, 2005.

[PRINCE2 BC 2009] *Managing successful projects with PRINCE2*. Bussines Case. pp 19-22. The Stationery Office. 2009

[PRINCE2 GLF 2009] *Office of Government Commerce (OGC): Managing Successful Projects with PRINCE2TM (Spanish Translation). Gestión de los límites de una Fase.* pp 213-227. The Stationery Office, 2009.

[PRINCE2 PL 2009] *Office of Government Commerce (OGC): Managing Successful Projects with PRINCE2TM (Spanish Translation).Planes del Proyecto.* pp 69-86. The Stationery Office, 2009.

[PRINCE2 Tolerance 2009] *Managing successful projects with PRINCE2. Areas of Tolerance in Projects*. pp 102. The Stationery Office, 2009.

[QSEE1]. *Visión de la herramienta Superlite de QSEE Technologies.* http://www.leedsmet.ac.uk/qsee/index.htm

[QSEE2] *Características de los productos de QSEE Technologies.* http://www.leedsmet.ac.uk/qsee/multi-case.htm

[R. Díaz 2002] Rebeca Díaz Redondo. *Reutilización de requisitos funcionales de sistemas distribuidos utilizando técnicas de descripción formal.* ISBN 8469972901. Universidad de Vigo, 2002.

[Raian et al 2010] Raian Ali, Fabiano Dalpiaz, Paolo Giorgini. *A goal-based framework for contextual requirements modeling and analysis.* Requirements Eng Journal. Springer-Verlag London Limited 2010.

[Riebisch et al. 2002] Riebisch, M.; Bollert, K.; Streitferdt, D; Philippow, I. *Extending feature diagrams with UML multiplicities.* Sixth Conference on Integrated Design and Process Technology (IDPT' 2002), Pasadena, CA, June 2002.

[RSEB 1998] Griss, M., Favaro, J., and d' Alessandro, M. *Integrating feature modeling with the RSEB.* In Proceedings of the Fifth International Conference on Software Reuse, pages 76--85. IEEE Computer Society Press, 1998.

[RU.U. 2010] *Testing framework for Ruby.* Link actualizado a Junio 2010. http://homepage1.nifty.com/markey/ruby/rubyunit/index_e.html

[Ruby 2010] *Ruby Home Page.* Link actualizado a Julio 2010. http://www.ruby-lang.org/en/

[S.A. Oracle] *Resúmenes sobre el Mecanismo de Alertas y Señales en Oracle*. http://download.oracle.com/docs/cd/E11882_01/appdev.112/e10577/d_alert.htm http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14258/d_alert.htm

[S.A. SQLS] *Resumen sobre el Mecanismo de Alertas y Señales en SQL SERVER.* Link actualizado a Mayo 2010. http://technet.microsoft.com/es-es/library/ms180982.aspx

[SAFARI 2010] Página web del *proyecto SAFARI.* Link actualizado a Junio 2010. http://domino.research.ibm.com/comm/research_projects.nsf/pages/safari.index.html

[Sam R. 2008] Sam R. Alapati. *Expert Oracle Database 11g Administration* Apress Series. Page 514. Apress, 2008.

[Sami et al. 2005] Sami Beydeda, Matthias Book, Volker Gruhn. *Model-driven software development*. pp 1-19. Editor: Birkhäuser, 2005

[Schobbens et al. 2006] Schobbens, P.; Heymans, P.; Trigaux, J. *Feature Diagrams: A Survey and a Formal Semantics.* 14th IEEE International Conference on Requirements Engineering. 11-15, pp: 139-148.2006

[Schobbens et al. 2007] Schobbens, P.; Heymans, P.; Trigaux, J.; Bontemps, Y. *Generic semantics of feature diagrams.* Computer Networks 51(2): 456-479 (2007).

[SED 2007] *Proyecto SED (Stream Editor).* Free Software Foundation, Inc. 2007. http://www.gnu.org/software/sed//sed.html

[Segura 2008] Segura, S. *Automated Analysis of Feature Models using Atomic Sets.* First Workshop on Analyses of Software Product Lines. ASPL 2008.

[Sharifi et al. 2008] Sharifi, M.; Ayat, M.; Sahibudin, S. *Implementing ITIL-Based CMDB in the Organizations to Minimize or Remove Service Quality Gaps.* Publisher: Modeling & Simulation, 2008. AICMS 08. Second Asia International Conference on, Volume, Issue, 13-15. Page(s):734 – 737, IEEE, May 2008.

[Sharifi et al. Cobit 2008] Sharifi, M.; Ayat, M.; Sahibudin, S., *Combining ITIL, COBIT and ISO/IEC 27002 in Order to Make a Comprehensive IT Framework in Organizations.* Publisher: AMS, IEEE 2008.

[Simon 2009] Simon Adams. *ITIL V3 foundation handbook*. pp.7-11. TSO, 2009.

[Simos et al. 1996] Simos, M., et al. *Software Technology for Adaptable Reliable Systems (STARS) Organization Domain Modeling (ODM) Guidebook Version 2.0* (STARS-VC-A025/001/00). Manassas, VA: Lockheed Martin Tactical Defense Systems, 1996.

[Simos, M. 1995]. Simos, M., *Organization Domain Modeling (ODM): Formalizing the Core Domain Modeling Life Cycle.* SIGSOFT Software Engineering Notes, Special Issue on the 1995.

[Sommerville 2006] . Sommerville, I. *Software Engineering.* Addison Wesley. 8th edition, 2006

[Stackpole and Hanrion 2007] Bill Stackpole, Patrick Hanrion. *Software Deployment, Updating, and Patching. About Helpdesk and the CMDB.* Page 176. CRC Press, 2007.

[Steger et al. 2004] Steger, M.; Tischer, C.; Boss, B.; Muller, A.; Pertler, O.; Stolz, W.; Ferber, S. I*ntroducing PLA at Bosch Gasoline Systems.* Software Product Line Conferences 2004, Boston, MA, Aug/Sep 2004, Springer- Verlag LNCS 3154, pp 34-50. 2004.

[Steve 2009] Steve Hoberman. *Data Modeling Made Simple: A Practical Guide for Business and IT Professionals.* Page 87. Edition 2. Technics Publications, 2009.

[Steven and Bill 2005] Steven Feuerstein, Bill Pribyl. *Oracle PL/SQL programming.* O'Reilly Media, Inc., 2005

[Sun et al. 2008] Sun, J.; Zhang, H.; Li, Y.F.; Wang, H. *Formal semantics and verification for feature modeling.* ICECSS05, 2005.

[TADDM 2010] *Tivoli Application Dependency Discovery Manager and Tivoli Business Service Manager Integration Solution Guide*. IBM Corporation 2010 http://www.ibm.com/developerworks/wikis/display/tivoliaddm/Tivoli+Application+Dependency+Discovery+Manager+and+Tivoli+Business+Service+Manager+Integration+Solution+Guide

[Tai et al. 2009] Ling Tai, Ron Baker, Elizabeth Edmiston and Ben Jeffcoat. *IBM Tivoli Common Data Model: Guide to Best Practices.* IBM Corp. 2008.http://www.redbooks.ibm.com/redpapers/pdfs/redp4389.pdf

[Thomas 1992] Thomas Bruce. *Designing quality databases with IDEF1X information models.* Dorset House Pub., 1992.

[Thomas and Horst 2002] Thomas von der Maven, Horst Lichter *Modeling Variability by UML Use Case Diagrams.* IEEE Joint International Requirements Eng. Conference (RE02). ISBN 0-9724277-0-8. 2002 Avaya Inc.

[Thomas and Hunt 2001] Thomas, D.; Hunt, A. *Programming Ruby. The Pragmatic Programmers' Guide.* Addison Wesley, 2001.

[Thomas et al. 2004] Thomas, D.; Fowler C.; Hunt, A. *Programming Ruby: The Pragmatic Programmers' Guide.* Pragmatic Bookshelf; 2nd edition, October 2004.

[Thomas M. et al. 2004] Thomas M. Connolly, Carolyn E. Begg. *Database solutions: a step-by-step guide to building databases.* Page 274. Pearson Education, 2004

[Toby 1999] Toby J. Teorey. *Database modeling and design.* pp 24-26. Edition 3. Morgan Kaufmann, 1999.

[UML 2003] Object Management Group. *UML 2.0 Infrastructure Specification*, OMG document ptc/03-09-15, 2003.

[Variant 2010] *Pure Systems. pure::variants. Variant management Tool*. Link actualizado a Junio 2010. http://www.pure-systems.com/pure_variants.49.0.html

[Varmod-Prime 2010] Página web de la *herramienta VARMOD-PRIME* para la gestión de la variabilidad. Link actualizado a Junio 2010. http://www.sse.uni-due.de/WMS/EN/?go=139

[Wang et al. 2005] Wang, H.; Li, Y.; Sun, J.; Zhang, H.; Pan. J. *A semantic web approach to feature modeling and verification.* In Workshop on Semantic Web Enabled Software Engineering (SWESE05), Nov05.

[Weiss 1999] D.M. Weiss and C.T.R. Lai; SW *Product-Line Engineering – A Family-Based SW Development Process*, Addison-Wesley, Reading, Massachusetts, 1999.

[White et al. 2008, 2] White, J.; Doughtery, B.;Schmidt, D.C. Filtered Cartesian Flattening: *An Approximation Technique for Optimally Selecting Features while Adhering to Resource Constraints.* First Workshop on Analyses of Software Product Lines (ASPL 2008).

[White et al. 2008] White, J.; Benavides, D.; Schmidt, D. C.; Trinidad, P.; Ruiz-Corts, A. *Automated Diagnosis of Product- line Configuration Errors in Feature Models.* Software Product Line Conference 2008 (SPLC'08).

[William et al. 1989] L.E. William, K.A. Loparo, N.R. Nelson. *The Nature of Modeling.* Jeff Rothenberg in Artificial Intelligence, Simulation, and Modeling , John Wiley and Sons, Inc. pp. 75-92 . 1989

[Wood et al. 2001] Wood, M. F., & DeLoach, S. A. *An overview of the multiagent systems engineering methodology*. In Proceedings of first international workshop on agent-oriented software engineering. pp. 207–221. Springer-Verlag New York, Inc. 2001

[XFeature 2010] Página web de la *herramienta Xfeature* para la gestión de la variabilidad. Link actualizado a Junio 2010. http://www.pnp-software.com/XFeature/

[XML 2008]. *Extensible Markup Language (XML).* http://www.w3.org/TR/REC-xml

[XP 2003] Michele Marchesi, Giancarlo Succi. *Extreme programming and agile processes in software engineering*: 4th International Conference, XP 2003, Genova, Italy, May 25-29, 2003.

[Yu et al. 2008 2] Yu Y, do Prado Leite JCS, Lapouchnian A, Mylopoulos J *Configuring features with stakeholder goals.* In: Proceedings of SAC 08. ACM New York, N Y, USA, pp 645–649. 2007.

[Yu et al. 2008] Yu Y, Lapouchnian A, Liaskos S, Mylopoulos J, Leite. *JCSP From goals to high-variability software design.* In: Proceedings of ISMIS'08, vol 4994 of LNCS. Springer, Berlin, pp 1–16. 2008.

[Zave 2004] Pamela Zave, *FAQ Sheet on Feature Interactions*, www.research.att.com/~pamela/faq.html
Copyright AT&T, 1999, 2004.