Departamento de Sistemas Informáticos y Computación

Trabajo de Fin de Máster

MEMORIA TÉCNICA DEL PROYECTO

Knowledge Cube

Autor:
Verónica Pazos Corella

Directores:
Óscar Pastor López,
Ricardo Chalmeta Rosaleñ

9 de diciembre de 2009

Índice General

Lista de Figuras

Lista de Tablas

Resumen

El trabajo de fin de máster expuesto en esta memoria se ha realizado dentro del Programa de Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información impartido por la Universidad Politécnica de Valencia. El objetivo general de este trabajo es dar un paso adelante en la resolución de la problemática de la transformación de modelos empresariales en modelos informáticos, de forma que los empresarios puedan reutilizar el conocimiento que plasman en los mismos. Con este propósito, se proponen un conjunto de metamodelos para el modelado empresarial de las principales dimensiones de la empresa y un conjunto de reglas de transformación hacia un modelo informático para generar la mayor parte del mismo explotando el conocimiento que contienen los modelos empresariales. En concreto, en este trabajo se detalla:

1. El funcionamiento de un conjunto de metamodelos para el modelado empresarial incluyendo una propuesta para la discriminación de los elementos que forman parte del modelado empresarial pero no del modelo informático y una guía para facilitar el modelado con los metamodelos propuestos.

2. La definición de un conjunto de reglas de transformación entre los metamodelos y los elementos de un modelo informático.

3. El desarrollo de un caso de estudio para validar el trabajo realizado y detectar los límites del mismo.

Los metamodelos descritos en este documento permiten modelar las dimensiones principales de la empresa desde la perspectiva del empresario. Para facilitar el uso de los mismos, se propone una guía de modelado teniendo en cuenta los perfiles de las personas que participan en esta tarea. Además, están pensados para que tengan el poder expresivo suficiente para facilitar la transformación a un modelo informático.
En la transformación a un modelo informático, se debe tener en cuenta que cabe la posibilidad de que no todos los elementos del modelo empresarial formen parte del modelo informático. Para resolver esta problemática, el trabajo incluye una propuesta para representar el modelo empresarial con una estructura de datos manejable por un computador sobre la que aplicar algoritmos de fragmentación de modelos y extraer el subconjunto de elementos del modelo empresarial que forman parte del modelo informático. Así pues, es sobre este subconjunto de elementos sobre el que se aplican las reglas de transformación. En este trabajo, las reglas establecidas están enfocadas a generar un modelo informático para el nivel transaccional de la empresa.
En el caso de estudio se ponen en práctica las diferentes propuestas realizadas. En primer lugar, se modela el funcionamiento de una agencia de fotografía siguiendo la guía de modelado propuesta y utilizando los metamodelos propuestos. A continuación, se aplica la fragmentación de modelos, se ejecutan las reglas establecidas y se crea un modelo informático a partir de la ejecución de las mismas.
Finalmente, se revisan los resultados obtenidos, se detallan los problemas y puntos débiles detectados al realizar el caso de estudio y se proponen posibles extensiones y mejoras a realizar a partir de este trabajo.

Abstract

The master thesis presented in this document have been developed under the Software Engineering, Formal Methods and Information Systems Master Program of the Polithecnical University of Valencia. The main goal is to take a step forward in the transformation of enterprise models to computer models, in a way that could make possible that business people could be able to reuse the knowledge depicted in enterprise models. To make this possible, a set of metamodels for enterprise modeling are proposed as well as a set of mapping rules to a computer model to generate the major part of it exploiting knowledge depicted in enterprise models. Specifically, in this work are detailed:

- The operation of a set of metamodels for enterprise modelling, including a proposal for discriminate what elements, that are part of the enterprise models, are no part of the computer models and a guide for facilitate the modeling task.
- A set of mapping rules between the proposed metamodels and computer model elements.
- A case example to validate the work done and detect limitations.

Proposed metamodels make possible model the main enterprise dimensions. In order to facilitate its use, a guide for modeling is provided taking into account the profile of people that carry out this task. Moreover, the metamodels have been designed for having enough expressive power to facilitate computer model transformation task.
It must be taken into account the fact that not all enterprise model elements are depicted in computer models too. In order to solve this problem, this document includes a proposal for representing enterprise models as computer data structures where apply model slicing algorithms to extract the enterprise model element's subset that are going to be depicted in the computer model. Mapping rules will be applied over the extracted subset. In this work, mapping rules are focused to generate the transactional work level of an enterprise.
The case example is modelled with the proposed metamodels following the modeling guide, next model slicing is applied, mapping rules are executed and a computer model is created starting out from the rules' execution results.
Finally, a result review is done and detected problems are detailed as well as the work limits and possible extensions to do starting out from this work.

Capítulo 1

Descripción General y Objetivos del Proyecto

El punto de partida de este trabajo es la contribución realizada al proyecto *Red de Investigación Interoperabilidad para Aplicaciones y Software en Redes de Empresas de la Comunidad Valenciana (INTERVAL)* desarrollado desde Octubre a Diciembre del año 2008 en el que se participó en el diseño de un método para la aplicación del paradigma Model-Driven Interoperability [1]. En la realización de este proyecto se estudiaron las bases del desarrollo dirigido por modelos y diferentes técnicas de modelado empresarial.

En el ámbito empresarial, el paradigma Model-Driven Development (MDD) se centra en crear modelos para representar el funcionamiento de una empresa y transformar este modelo para generar un sistema software que le dé soporte [2] de forma lo más automática posible.

Actualmente, existen muchos lenguajes de modelado empresarial como IDEF (Integrated DEFinition Methods) [3] [4] [5], DFD (Data Flow Diagrams) [6], CIMOSA (Open System Architecture for CIM) [7], IEM (Integrated Enterprise Modelling) [8], METIS [9], GRAI [10] o GIM [11]. Sin embargo, no todos los lenguajes abarcan todas las dimensiones de las empresas. La mayoría están orientados a modelar aspectos relacionados con la información, los procesos y sus flujos [12] y no existen propuestas para transformar los modelos realizados con estos lenguajes y generar código a partir de los mismos.

No obstante, no se debe perder de vista que el objetivo del modelado empresarial no radica únicamente en que éste sea transformable a un sistema software que dé soporte a la empresa, sino que es un activo cuyas utilidades van más allá de reutilizar la información plasmada en los modelos para generar código automáticamente. Además, realizar un modelo empresarial completo implica utilizar el presupuesto de la empresa para obtener una representación gráfica de la misma. El coste en tiempo y dinero de realizar el modelo depende de la complejidad de la empresa y los empresarios pueden considerarlo elevado si no conocen los beneficios que puede aportarle contar un modelo de estas características. Por lo tanto, es importante que los empresarios conozcan las posibilidades de explotación de un modelo empresarial y que éste permita obtenerlas. En la figura 1.1 se esquematizan algunos de los usos que se le pueden dar a un modelo empresarial.

Un modelo empresarial permite tener una visión holística de la empresa de forma que es posible estudiar cómo se propagaría un cambio para evaluar el coste del mismo, así como detectar incongruencias en el funcionamiento global que pueden pasar desapercibidas con visiones aisladas, detectar puntos débiles, sobre los que se pueden definir estrategias para que no supongan amenazas para la empresa; o procesos cuya entrada o salida son out-of-core para definir puntos de interoperabilidad potencial con socios, proveedores o clientes.

Por otro lado, si en el modelo se integra la dimensión de recursos humanos y la dimensión operativa de la empresa (organigrama de la empresa y tareas a realizar por cada unidad organizativa), el modelo empresarial

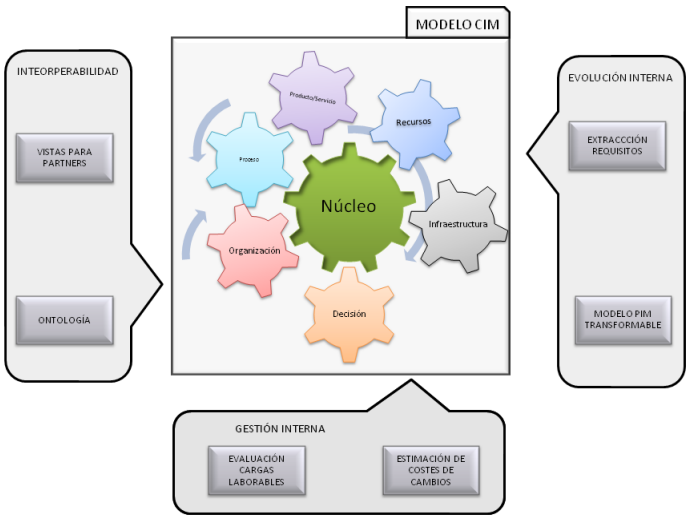

Figura 1.1: Posibles usos de los modelados empresariales

se puede utilizar desde la dirección de la empresa para evaluar la carga de trabajo que soporta cada cada perfil de trabajo de la empresa.

Además, el modelo empresarial refleja el vocabulario establecido en la empresa para designar los conceptos que intervienen en el funcionamiento de la misma así como las relaciones entre dichos conceptos. Por lo tanto, se puede utilizar como punto de partida para crear una ontología empresarial considerada un punto esencial en la tarea de hacer dos sistemas interoperables [13]. Asimismo, se podrían generar diferentes vistas del modelo para entregar a nuevos trabajadores incorporados a la empresa, socios, proveedores o clientes obteniendo un soporte de comunicación en forma de modelo.

Para poder utilizar el modelo empresarial en estos ámbitos y además generar software a partir del mismo lo más automáticamente posible, el modelo empresarial debe abstraer los detalles de un posible sistema informático, pero no por ello debe ser poco preciso en lo que a la representación de la empresa se refiere. Teniendo en cuenta los beneficios que se pueden obtener de un modelo empresarial, el propósito del trabajo expuesto en este documento está orientado principalmente a reutilizar el conocimiento que expresan los modelos empresariales para generar modelos informáticos sin perder de vista el hecho de que se puedan obtener el resto de beneficios a largo plazo. Por lo tanto, el objetivo principal del proyecto es: **Adaptar un conjunto de metamodelos de forma que se puedan establecer correspondencias hacia modelos conceptuales y reutilizar el conocimiento plasmado en los modelos empresariales en el proceso de desarrollo de software**.

Dentro del proyecto citado al comienzo de este capítulo, se empezó a realizar la primera parte del objetivo, adaptando los metamodelos propuestos en [14] y añadiendo los elementos característicos del modelado empresarial y la interoperabilidad. A partir de estos resultados, se comenzó el desarrollo del presente trabajo descartando los elementos relacionados con la interoperabilidad. La metodología seguida en el desarrollo de este trabajo ha sido la siguiente:

1. Revisión exhaustiva del estado del arte respecto al desarrollo de software dirigido por modelos y la transformación de modelos. Repaso de la adaptación de los metamodelos realizados en el proyecto INTERVAL y estudio del enfoque OO-Method [15].

2. Incorporación de nuevos elementos a los metamodelos para que ofrezcan la posibilidad de realizar el modelado en un nivel de detalle suficiente para poder establecer reglas de transformación hacia modelos de representación del sistema informático.
3. Creación de las relaciones inter-metamodelo.
4. Definición de la guía de modelado.
5. Definición del mecanismo de extracción de los elementos que forman el modelo informático.
6. Definición de las reglas de transformación.
7. Desarrollo del caso de estudio:
   a) Modelado a nivel CIM y extracción de los elementos que representan el modelo informático.
   b) Ejecución de las reglas de transformación definidas.
   c) Creación de un modelo con la herramienta OLIVANOVA [16] a partir del resultado del paso anterior.
   d) Revisión del modelo obtenido.

A lo largo de este documento, se detallan los resultados de cada uno de estos pasos. En primer lugar, en el capítulo 2 se realiza una revisión del estado del arte y los conceptos teóricos relacionados con el trabajo realizado. En el capítulo 3 se explica la adaptación de los metamodelos así como los elementos incorporados, las relaciones inter-metamodelo, la guía de modelado y el mecanismo de extracción de los elementos del sistema informático. En el capítulo 4 se definen las reglas de transformación. En el capítulo 5 se desarrolla el caso de estudio. Finalmente, en el capítulo 6 se detallan las conclusiones obtenidas.

Capítulo 2

Estado del Arte

En este capítulo se revisan los conceptos teóricos relacionados con las propuestas realizadas en los capítulos 3 y 4. En primer lugar, se exponen los conceptos básicos del paradigma MDD, centrándose en el estándar Model-Driven Architecture (MDA) que implementa este paradigma. A continuación, se explican los aspectos más destacados de las transformaciones de modelos. Finalmente, se detallan las características de la Propuesta MDK, origen de la propuesta de modelado realizada en este trabajo, y el enfoque OO-Method destino de la propuesta de reglas de transformación descrita en el capítulo 4.

2.1. El desarrollo de software dirigido por modelos

El paradigma MDD, enmarcada dentro del enfoque Model-Driven Engineering (MDE), centra el proceso de desarrollo de software en el uso de modelos. MDD se considera la evolución natural de la ingeniería del software. En este paradigma, se incorporan las herramientas necesarias para transformar los modelos, utilizados en el diseño de software, en el código que lo implementa, otorgando a los modelos un papel activo en el proceso de desarrollo de software. Así pues, los modelos están considerados como elementos de primer orden y las transformaciones de modelos el principal tipo de operación [17].

Echando la vista atrás, se puede concluir que en los inicios del desarrollo del software se utilizaban lenguajes de tipo ensamblador en el que se debía conocer el juego de instrucciones que manejaba el tipo de ordenador sobre el cual se iba a ejecutar el programa. Más tarde, aparecieron los lenguajes de alto nivel, que abstraían la complejidad de las características específicas del juego de instrucciones de la máquina destino. Con los lenguajes de alto nivel, aparecieron herramientas (intérpretes y compiladores) que transforman los lenguajes de alto nivel en código ejecutable. Con el tiempo, la historia se repite mediante el enfoque MDD. Con MDD se abstraen los detalles de una tecnología específica o lenguaje de alto nivel mediante el uso de modelos y aparecen diferentes herramientas para transformar directamente el modelo en el código ejecutable que lo implementa. Así pues, este paradigma de desarrollo se basa en separar la especificación de un problema de la plataforma sobre la que se implementa la solución al mismo y realizar transformaciones que permitan obtener automáticamente la implementación a partir de la especificación. Esta abstracción permite enfrentarse a la rapidez con la que evolucionan las tecnologías. Dado que el modelo abstrae estos aspectos los cambios tecnológicos no afectan al modelo. Por lo tanto, para transportar un sistema a nueva tecnología, bastaría generar de nuevo el modelo para la nueva tecnología.

En el contexto de MDD, aparecen diversos enfoques que lo implementan como son los Modelos Específicos del Dominio, (DSM), y MDA. Los DSM son modelos creados con el objetivo de representar un dominio en concreto y por lo tanto incluyen elementos para representar los conceptos típicos que aparecen en dicho dominio. Sin embargo, el desarrollo de este trabajo se centra en MDA, por lo que a continuación se explican de forma más detallada las características básicas de este enfoque.

2.1.1. Model-Driven Architecture

El Object Management Group, Inc. (OMG), fundado en 1989, es una organización internacional sin ánimo de lucro en forma de consorcio integrada por más de 800 miembros. Su propósito original era promover el uso de la teoría y práctica de la tecnología orientada a objetos en el desarrollo de sistemas software. Sin embargo, en los últimos años el OMG ha centrado sus esfuerzos en crear un estándar para el desarrollo de sistemas software dirigido por modelos. Como resultado de esos esfuerzos surgió el enfoque MDA cuyos objetivos son aumentar el grado de interoperabilidad, portabilidad y reutilización de los sistemas software separando la especificación del problema de su solución [18]. Este enfoque propone desarrollar los sistemas software utilizando modelos que se sitúan en distintos niveles de abstracción. Los niveles que proponen representan un sistema bajo diferentes puntos de vista siendo la transformación de un modelo situado en un determinado nivel en otro modelo la operación principal dentro de este enfoque. Por lo tanto, el propósito es generar automáticamente[1] los modelos que componen los distintos niveles y finalmente el código del software que representa el sistema. En concreto, proponen tres niveles de abstracción:

- **Computer Independent Model (CIM):** Representa los requisitos del sistema en el entorno en el que va a trabajar, teniendo en cuenta los modelos de negocio y un punto de vista empresarial.
- **Platform Independent Model (PIM):** En este nivel se modela la funcionalidad de un sistema sin definir ni cómo ni en qué plataforma se implementará. Los modelos se centran en la información desde un punto de vista computacional.
- **Platform Specific Model (PSM):** Este nivel lo conforman los modelos que representan un sistema incluyendo las características de la plataforma elegida para su implementación. Los modelos están centrados en un punto de vista tecnológico.

Dentro de MDA, la OMG propone utilizar UML (Unified Modelling Language [19]) como lenguaje de representación gráfica de un sistema software. A partir del metamodelo de UML se pueden realizar un conjunto de diagramas que permiten representar tanto la estructura como el comportamiento de un sistema informático. Para representar la estructura se pueden utilizar diagramas de clases, de estructuras compuestas o de componentes entre otros. Respecto al comportamiento, se pueden utilizar diagramas de secuencia, de comunicación, de interacción, etcétera.

2.2. Transformación de modelos

La clave del proceso dirigido por modelos es la transformación de un modelo en otro de forma lo más automática posible. Para ello se necesita relacionar los componentes de los metamodelos a través de los niveles de abstracción. La definición de reglas de transformación (*mapping* en inglés) es la operación que permite realizar la correspondencia de elementos entre modelos. Estos elementos pueden ser unitarios, formados por un solo elemento, o complejos, formados por más de un elemento. En función del tipo de elementos que se están relacionando, se pueden distinguir entre *mappings* elemento a elemento, si ambos son unitarios, elemento a subgrafo si un elemento es unitario y el otro complejo, o subgrafo a subgrafo si ambos son complejos. A continuación, se proporciona una definición general del concepto de *mapping*:

Dados dos modelos A y B (A, $B \in$ MOD) siendo MOD un conjunto de modelos. El *mapping* entre A y B es una relación m de $Sub(A)XSub(B)$ siendo $Sub(X)$ el conjunto de todos los subgrafos de X. Es decir m relaciona los subgrafos de A (que pueden estar formados por un solo elemento) con subgrafos de B.

[1] Cabe destacar que la diferencia entre utilizar MDA y otro paradigma que implemente MDE estriba en el hecho que en MDA únicamente se utilizan los estándares propuestos por la OMG respecto a los lenguajes de representación de sistemas y de especificación de transformaciones.

Esta definición deja un grado de libertad muy amplio, ya que habla de una relación m entre subgrafos pero no la define. Para poder aplicar está definición, es necesario concretar dicha relación teniendo en cuenta puntos de vista tanto sintácticos como semánticos y resolver los conflictos que aparecen en cada caso.

Desde el punto de vista sintáctico, los conflictos que pueden aparecer son de correspondencia de sintaxis sin que se dé la correspondencia semántica. Es decir, dados dos modelos A y B, un elemento $a \in A$ y un elemento $b \in B$ cuyo significado sea el mismo en ambos modelos, puede estar representado de forma sintácticamente diferente en cada uno de ellos. Desde el punto de vista semántico ocurre al contrario: dado un elemento $a \in A$ puede estar representado de la misma forma sintáctica que un elemento del $a \in B$ pese a que su significado sea diferente. Por lo tanto, se necesita definir de forma muy precisa como se deben realizar las transformaciones entre modelos a través de las relaciones establecidas con las reglas de transformación.

Por consiguiente, definir las reglas de transformación entre modelos es el hecho de establecer las relaciones entre los elementos de los modelos involucrados teniendo en cuenta los conflictos tanto sintácticos como semánticos que puedan aparecer. Esta relación se puede establecer desde dos perspectivas:

- A nivel horizontal: Se relacionan modelos que se encuentran en el mismo nivel del abstracción.
- A nivel vertical: Se relacionan modelos que se encuentran en diferentes niveles del abstracción.

En la figura 2.1 se conceptualiza el proceso de generación de software basado en transformaciones utilizando los niveles de abstracción del estándar MDA. En cada paso de transformación se reutiliza la información plasmada en un modelo de cierto nivel de abstracción para generar el modelo del siguiente nivel. En la mayoría de casos, el desarrollo de un sistema software toma como punto de partida el nivel PIM ya que las reglas de transformación para generar código a partir de este nivel están mucho más desarrolladas ([20], [21]). Además, ambos niveles son afines en el sentido de que los modelos que se representan en los mismos son sistemas informáticos, mientras que en el nivel CIM se representan modelos empresariales.

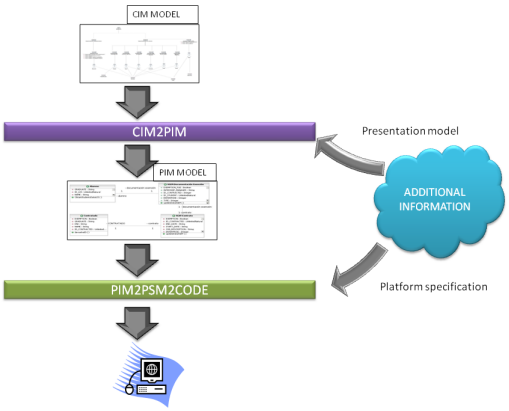

Figura 2.1: Esquema conceptual del proceso de transformación

Sin embargo, existen menos soluciones cuya raíz esté situada en un modelo empresarial de nivel CIM debido a que el hueco semántico entre el nivel CIM y PIM es mayor que entre modelos de nivel PIM y PSM. Para poder establecer las reglas de transformación entre metamodelos de un nivel CIM a un nivel PIM se necesita un lenguaje a nivel CIM que esté bien estructurado, ya que en caso contrario sería casi imposible llevar a cabo la transformación [22]. El trabajo expuesto en este documento se basa en la Propuesta MDK (Model Driven Knowledge [14]), que es una propuesta de modelado que permite representar un sistema de gestión del conocimiento y cuyos elementos están bien estructurados. A continuación, se describe la Propuesta MDK y el enfoque OO-Method, seleccionados respectivamente como origen y destino de las reglas de transformación.

2.3. Metodología KM-IRIS: La Propuesta MDK

La Propuesta MDK desarrolla un lenguaje de representación del conocimiento en el nivel CIM de abstracción de modelos creando una representación gráfica propia mediante la extensión del lenguaje UML a partir del uso de perfiles. Esta propuesta es parte del resultado del proyecto denominado 'Gestión del conocimiento en el ámbito de las empresas virtuales' financiado por la Comisión Interministerial de Ciencia y Tecnología (CICYT) que llevó a cabo el Grupo de Investigación en Integración y Re-Ingeniería de Sistemas (Grupo IRIS) de la Universitat Jaume I en el período 2003-2006. En este proyecto se desarrolló la metodología KM-IRIS cuyo propósito fue el desarrollo de una arquitectura que permita representar, gestionar y aplicar el conocimiento en cualquier tipo de organización. Los objetivos concretos de este proyecto fueron:

1. Metodología KM-IRIS: Metodología para guiar el proceso de desarrollo e implementación de un sistema de gestión del conocimiento en la empresa virtual [23].
2. Propuesta MDK: Un conjunto de modelos para permitir la identificación, representación y comunicación del conocimiento inherente a la empresa virtual [14].
3. El diseño de una infraestructura tecnológica que permita almacenar, procesar y distribuir el conocimiento dentro de la empresa virtual.

La metodología KM-IRIS es una metodología práctica que se puede utilizar para dirigir el proceso de desarrollar e implementar un sistema de manejo del conocimiento para procesar, almacenar, recolectar y distribuir el conocimiento generado dentro de una empresa y de las relaciones que mantiene con las diferentes organizaciones con las que coopera. Esta metodología define, entre otros, las diferentes fases del proyecto, las tareas a realizar en cada una de ellas, las técnicas de soporte, las plantillas y cuestionarios a realizar, los lenguajes de modelado y la infraestructura tecnológica. A continuación, se definen brevemente las fases de las que consta la metodología KM-IRIS.

Identificación de los bloques conceptuales de conocimiento, conocimiento objetivo (requisitos) y clasificación del mismo en categorías.

Extracción del conocimiento y definición de los procedimientos de extracción y cálculo.

Representación del conocimiento.

Procesamiento: desarrollo de una infraestructura tecnológica de soporte a la representación del conocimiento.

Explotación: establecer mecanismos de formación y mejora continua entre los integrantes de la organización, así como llevar a cabo el proceso de retroalimentación al sistema de gestión del conocimiento.

En el contexto de la fase de Representación del Conocimiento, se incluye la Propuesta MDK que se compone de un conjunto de metamodelos para representar el conocimiento empresarial en todas las dimensiones de la empresa. En la figura 2.2 se muestra la jerarquía de modelos planteada en esta propuesta, así como los diagramas de los que se compone cada modelo. Esta jerarquía consta de dos niveles: CIM-Knowledege y CIM-Business. A continuación, se describen las características principales de cada uno de ellos.

CIM-Knowledge: representa el nivel superior del CIM. En este nivel se determina cual es el objetivo de la empresa en relación con el conocimiento y su posterior gestión. La empresa decide los bloques sobre los cuales desea basar su sistema de gestión del conocimiento y cual es su conocimiento objetivo. El modelo propuesto se denomina 'Modelo de Conocimiento' y es el punto de partida del sistema de gestión

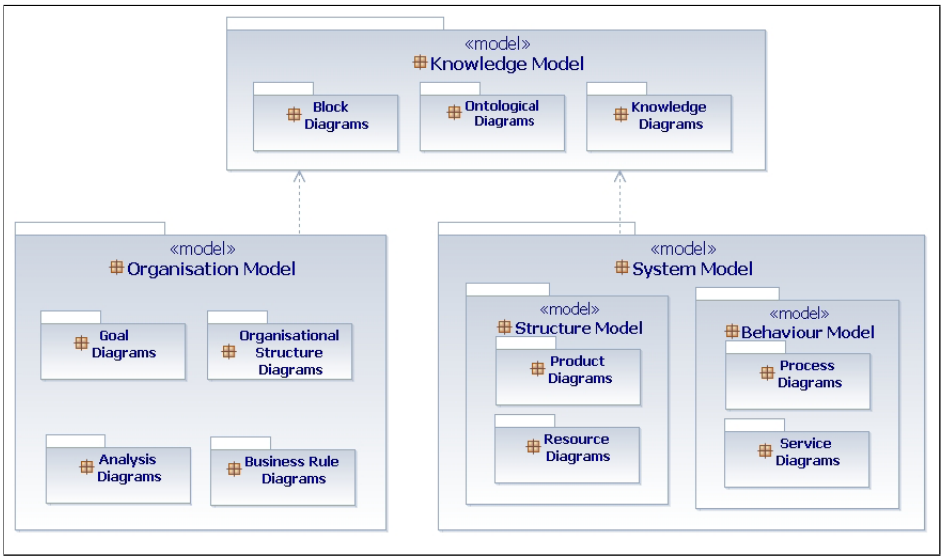

Figura 2.2: Bloques conceptuales de la Propuesta MDK

del conocimiento. En él, deben estar representados todos los elementos sobre los que la empresa desea gestionar su conocimiento, ofreciendo un modelo global del mismo. Siguiendo la Metodología KM-IRIS, se representan los bloques conceptuales definidos en la empresa y el conocimiento objetivo identificado a partir de ellos, las variables y fuentes de conocimiento utilizados para su extracción, y la conexión entre estos elementos y los del nivel inferior. Este modelo global es el más importante, puesto que en él se define cual es el objetivo de la empresa respecto al sistema de gestión del conocimiento y en qué se debe basar la representación del mismo en los siguientes niveles de abstracción.

CIM-Business: representa el nivel inferior del CIM, en el cual se ofrece una visión detallada del conocimiento objetivo de cada uno de los bloques conceptuales mostrados en el nivel superior. Para ello se proponen dos modelos: el 'Modelo de Organización' y el 'Modelo de Sistema'. El 'Modelo de Organización' representa la empresa desde el punto de vista organizativo. Esta visión debe estar interrelacionada con el nivel superior y los bloques conceptuales allí definidos, en especial con el de *Organización*. El 'Modelo del Sistema' permite representar de forma detallada los bloques conceptuales de *Producto*, *Recurso*, *Proceso* y *Servicio*. Este modelo recibe este nombre puesto que representa el sistema que forma la empresa desde un punto de vista operativo y recoge, además de una visión detallada del conocimiento de los bloques antes citados y los requisitos para la implementación del sistema informático transaccional de la empresa. Este modelo se ha divido en dos para ofrecer una visión separada de la estructura y del comportamiento de la empresa, proporcionadas respectivamente por el 'Modelo de Estructura' y el 'Modelo de Comportamiento'. En este caso, el objetivo es modelar en mayor nivel de detalle los procesos de negocio y como se utilizan los elementos involucrados en estos procesos.

2.3.1. CIM-Knowledge

Tal y como se ha comentado anteriormente, el 'Modelo de Conocimiento' tiene por objetivo presentar una visión holística de la empresa desde el punto de vista del conocimiento. En la figura 2.3 se representa el 'Metamodelo de Conocimiento' desarrollado para representar los conceptos relacionados con el conocimiento según se han definido en la Metodología KM-IRIS, es decir, bloques conceptuales de conocimiento, conocimiento objetivo, las categorías y subcategorías ontológicas en las cuales se agrupa y fuentes de conocimiento, entre otros. Permite obtener el 'Modelo de Conocimiento'.

La aportación principal de este trabajo parte de la adaptación del 'Metamodelo de Conocimiento'. Por lo tanto, a continuación se exponen las características de los elementos del mismo con mayor detalle que el resto de metamodelos en aras de facilitar la comprensión del trabajo realizado.

KnowledgeBlock: representan una agrupación a nivel conceptual del conocimiento del mismo tipo que existe en una determinada área de la empresa. En concreto, debe representar los bloques de conocimiento sobre los cuales la empresa desea fundamentar su mapa de conocimiento y posterior sistema de gestión del conocimiento. Los bloques de conocimiento están relacionados entre sí de forma que unos están basados en otros y por lo tanto se establecen entre ellos relaciones de dependencia. Para este elemento se definen los atributos *type* (enumeración 'KnowledgeBlockType') y *priority*.

OntologicalCategory: permite clasificar el conocimiento objetivo de un mismo bloque en diferentes categorías ontológicas, que se dividen en otras subcategorías y así sucesivamente. Para esta clase se define el atributo *isLeaf* que indica si una categoría es hoja del árbol ontológico.

TargetKnowledge: representa el conocimiento objetivo que se desea gestionar en la empresa.

InstanceKnowledge: representa las posibles instancias que se pueden definir para un determinado conocimiento objetivo, y sirve para ejemplificar el conocimiento objetivo que se está modelando.

Concept: se utiliza para establecer una definición del conocimiento objetivo, indicando cual es la descripción asociada a ese conocimiento desde el punto de vista cognitivo. La finalidad es mostrar la idea o noción básica que reside en el conocimiento.

Procedure: representa el conocimiento de tipo procedural.

Attitude: representa el conocimiento de tipo actitudinal.

InputVariable: representa las entradas necesarias a partir de las cuales se puede obtener el conocimiento objetivo con la finalidad de sistematizarlo. Estas variables de entrada pueden ser explícitas o tácitas dependiendo de su origen (determinado por el atributo *type*).

ExtractionProcedure: representa el primer paso para la obtención del conocimiento objetivo. Mediante este procedimiento se extraen las variables de entrada necesarias para la obtención del mismo. El tipo de procedimiento viene determinado por el atributo *type* ('ExtractionProcedureType').

CalculationProcedure: representa el segundo paso para la obtención del conocimiento objetivo en el que se obtiene conocimiento empresarial a partir de las variables de entrada obtenidas mediante los procedimientos de extracción aplicando técnicas de cálculo o procesamiento de la información. El tipo de procedimiento viene determinado por el atributo *type* ('CalculationProcedureType').

KnowledgeSource: representa el origen a partir del cual se pueden obtener las variables que servirán para la extracción y/o el cálculo del conocimiento objetivo. Es una clase abstracta, cuyas especializaciones indican si la fuente es tácita o explícita.

ExplicitSource: representa las fuentes de conocimiento explícitas a partir de las cuales es posible extraer conocimiento. Para esta clase se han definido los atributos *type* ('ExplicitSourceType') y *location'* que especifica la ubicación de la fuente explícita.

TacitSource: representa las fuentes de conocimiento tácitas que pueden proporcionar el conocimiento que poseen. Su tipo viene determinado por el atributo *type* ('TacitSourceType').

La Propuesta MDK, plantea el 'Modelo de Conocimiento' como el conjunto de tres diagramas que proporcionan diferentes perspectivas sobre el conocimiento empresarial: *Diagrama de Bloques* en el que se representan los bloques conceptuales de conocimiento así como las categorías en las que se puede dividir el conocimiento objetivo dentro de cada bloque; *Diagrama Ontológico* en el que se representan el conocimiento objetivo, sus variables de entrada y procedimientos de extracción y cálculo para la obtención del mismo;

y el *Diagrama de Conocimiento* que permite representar gráficamente algunos ejemplos de conocimiento objetivo para ayudar a comprender el Modelo de Conocimiento. En la tabla 2.1 se muestran algunas de las representaciones gráficas de los elementos del metamodelo de conocimiento.

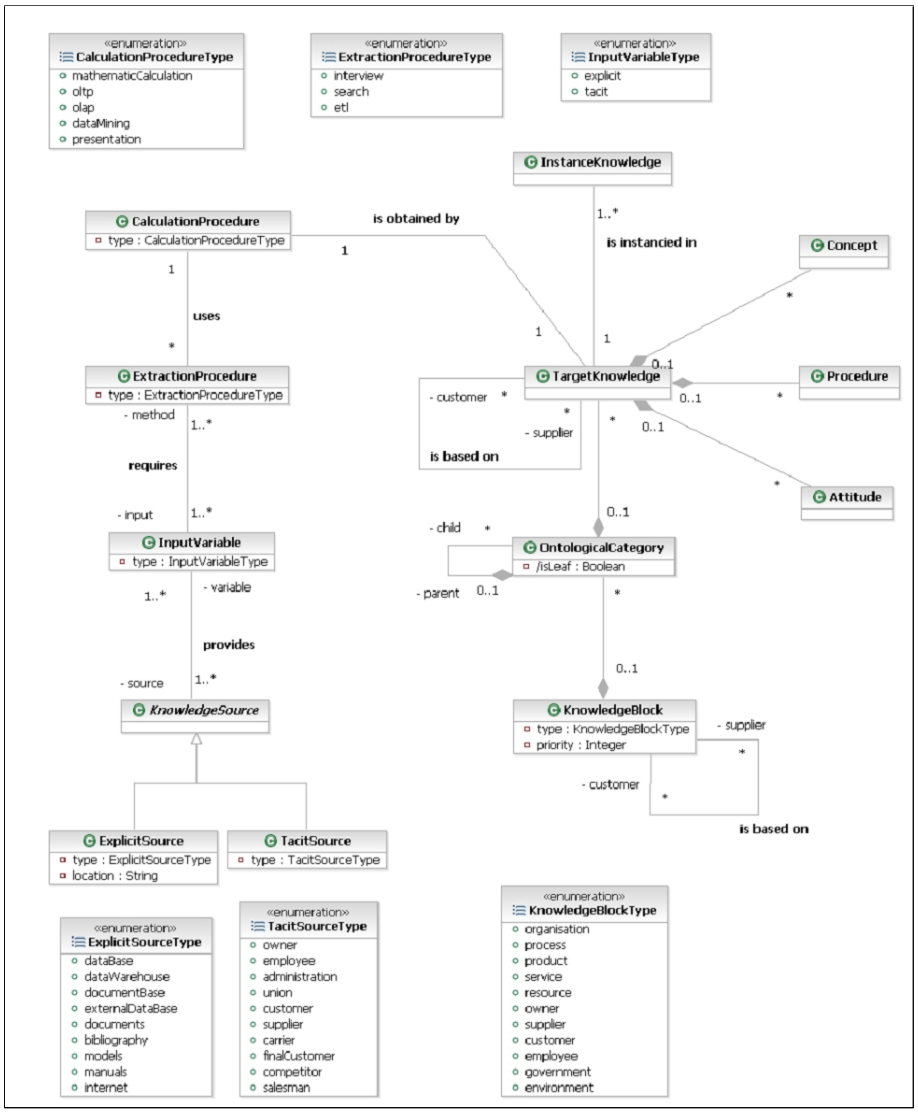

Figura 2.3: Metamodelo de Conocimiento

Tabla 2.1: Representación gráfica de los elementos del metamodelo de conocimiento

KnowledgeBlock	OntologicalCategory	TargetKnowledge	InstanceKnowledge
ExtractionProcedure	InputVariable	ExplicitSource	TacitSource

2.3.2. CIM-Business

Este nivel está compuesto por tres metamodelos: 'Metamodelo de Organización', 'Metamodelo de Estructura' y 'Metamodelo de Comportamiento'. A continuación, se describen las características elementales de cada uno de ellos.

Metamodelo de Organización

El 'Metamodelo de Organización' permite representar el 'Modelo de Organización', cuyo objetivo es representar la organización de la empresa desde cuatro puntos de vista diferentes: objetivos y estrategia, estructura organizativa, análisis y reglas de negocio. En la figura 2.4 se representa el 'Metamodelo de Organización'.

Los cuatro puntos de vista mencionados anteriormente dan lugar a cuatro diagramas que los representan: 'Diagrama de Objetivos', que permite representar los objetivos empresariales (estratégicos, tácticos u operativos) junto con las estrategias, planes y variables que permitirían llevarlo a cabo; 'Diagrama de Estructura Organizativa', que permite detallar el organigrama de la empresa; 'Diagrama de Análisis', que representa los sistemas (conjuntos de elementos interconectados que cooperan para lograr un objetivo común), centros de análisis y los parámetros, variables y elementos a analizar. Finalmente, incluye el 'Diagrama de Reglas de Negocio' para representar las directrices por las que se rige la empresa.

Metamodelo de Estructura y Comportamiento

Los metamodelos de Estructura y Comportamiento conforman el 'Modelo del Sistema' y completan el mapa de conocimiento haciendo posible la representación de los componentes del sistema informático transaccional de la empresa. Dichos metamodelos modelan el sistema informático desde el punto de vista de su estructura y comportamiento respectivamente. Además, ambos modelos permiten una visión detallada del conocimiento objetivo de los bloques *Producto* y *Recurso* en el primer caso, y del de los bloques *Proceso* y *Servicio* en el segundo. En la figura 2.5 y 2.6 se exponen los metamodelos de estructura y comportamiento respectivamente. A partir del 'Metamodelo de Estructura' se pueden realizar el 'Diagrama de Producto' y el 'Diagrama de Recursos'. El 'Diagrama de Producto' permite representar con mayor detalle el conocimiento relativo a los productos o servicios que realiza o presta la empresa, incluyendo las características de los procesos empresariales en los que están involucrados así como los elementos de mercadotecnia que lo promocionan. Por su parte, en el 'Diagrama de Recursos' se representan los recursos humanos y de infraestructura de la empresa.

El 'Metamodelo de Comportamiento' permite implementar el 'Modelo de Comportamiento' cuyo objetivo es representar las operaciones transaccionales que lleva a cabo la empresa mediante la representación de procesos, roles, flujos de información o actividades.

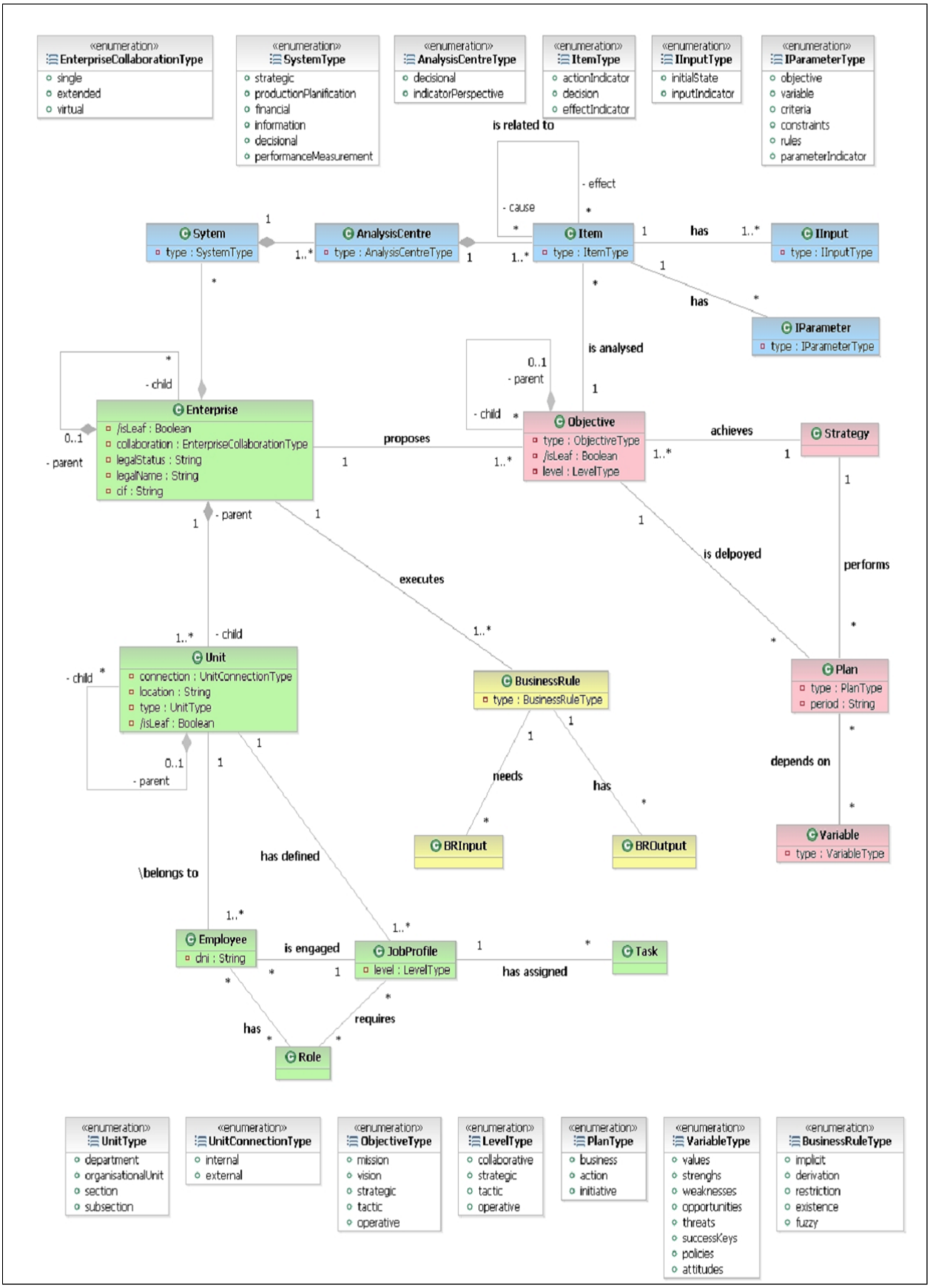

Figura 2.4: Metamodelo de Organización

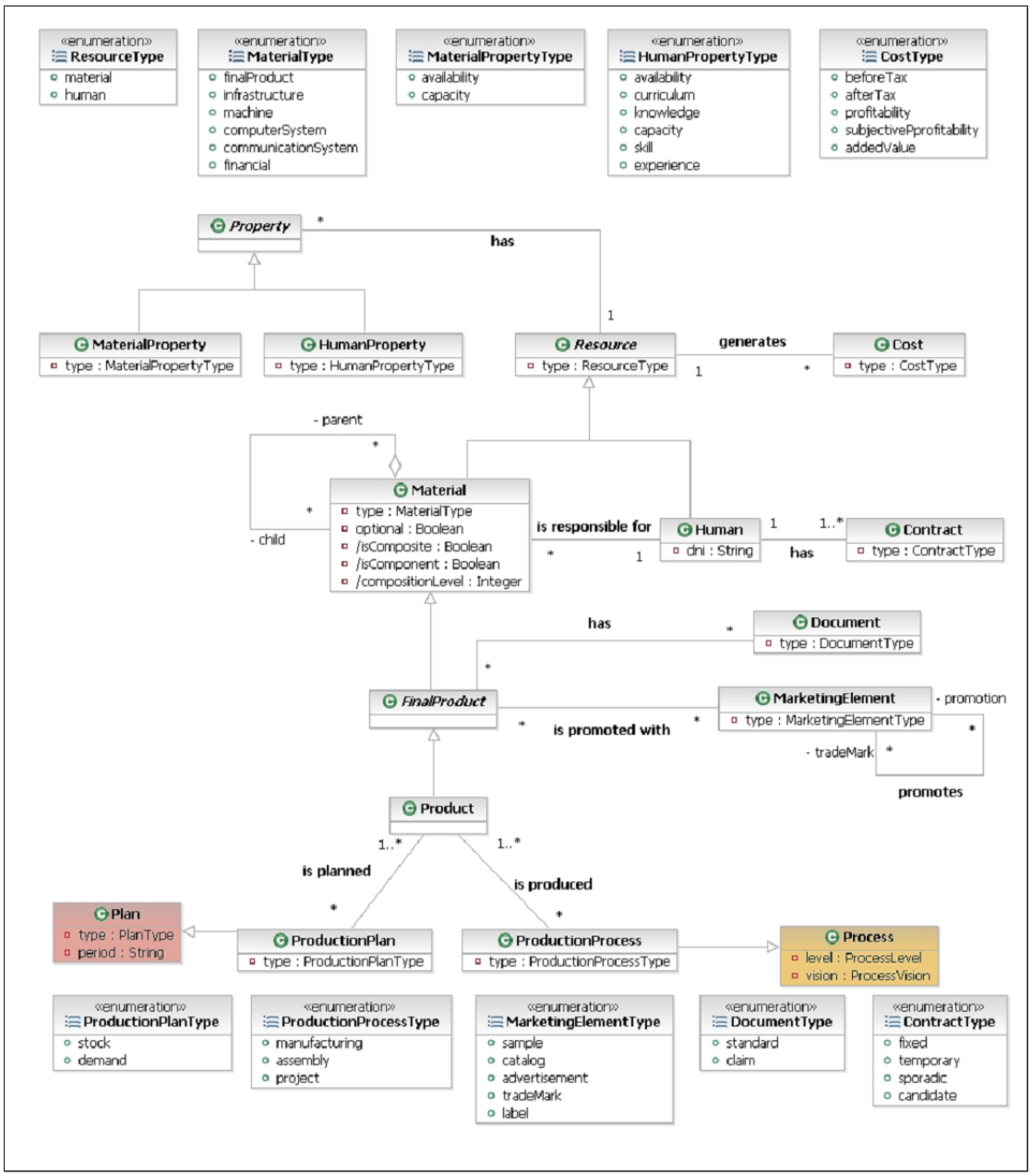

Figura 2.5: Metamodelo de Estructura

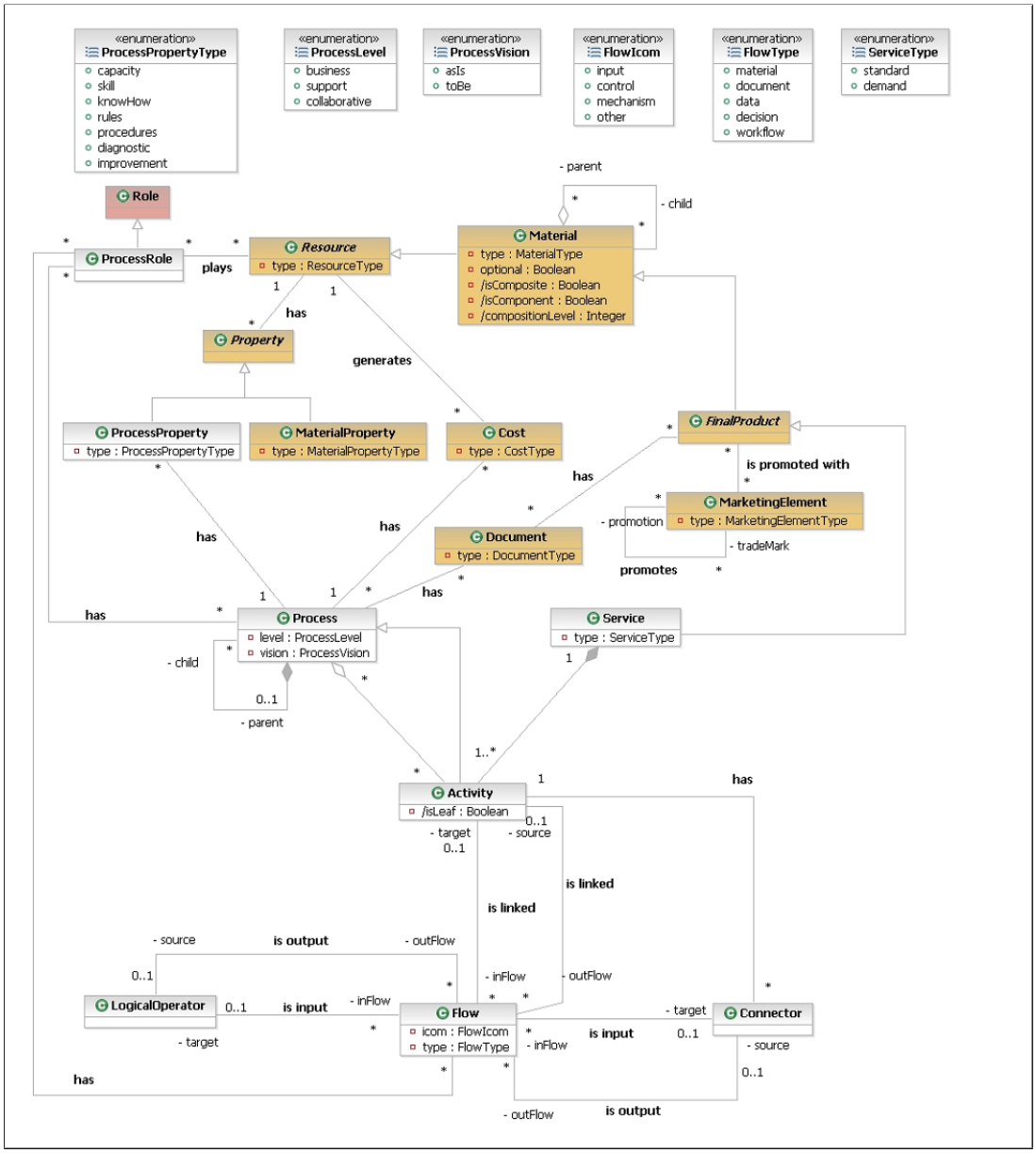

Figura 2.6: Metamodelo de Comportamiento

2.4. OO-Method

En esta sección se explica el enfoque OO-Method ([15], [24]), seleccionado como destino de las reglas de transformación entre los elementos de la adaptación de la Propuesta MDK realizada en este trabajo. En este enfoque se distinguen cuatro modelos: Modelo de Objetos, Dinámico, Funcional y de Presentación.

A continuación, se detallan los aspectos más destacados del Modelo de Objetos y el Modelo Dinámico con el fin de comprender el propósito del trabajo expuesto en este documento. Respecto al Modelo Funcional cabe destacar que su propósito es capturar la semántica operacional asociada a los cambios de estado que se producen en los objetos. Para ello, se definen los efectos que producen los eventos y se especifican las condiciones de evaluación, entre otros. Por su parte, el Modelo de Presentación representa la forma en la que se captura la información en un sistema mediante modelos de interacción con el usuario.

2.4.1. Modelo de Objetos

El Modelo de Objetos es un modelo gráfico en el que se definen las clases del sistema, sus atributos, servicios y relaciones entre clases, incluyendo la agregación y herencia, agentes y servicios compartidos. Seguidamente, se explican de forma genérica las características de algunos de los elementos que conforman el Modelo de Objetos.

- Clases: artefacto que permite abstraer la estructura y comportamiento que comparten un conjunto de objetos.
- Atributos: propiedades estructurales de una clase que la caracterizan. Para cada atributo se debe definir: un nombre, un tipo (Constante, Variable o Derivado), un tipo de dato (entero, real, cadena, etc...), longitud (para atributos de tipo de dato cadena), valor por defecto, un indicador para establecer si el atributo se requiere en la creación del objeto, otro indicador para apuntar si el atributo admite valores nulos y un último indicador para añadirlo al servicio de edición de la clase.
- Servicios: los servicios asociados a un evento modifican el estado de la instancia del objeto sobre el que se aplica y representan el comportamiento del mismo. Para cada servicio se puede indicar un nombre (único en el contexto de la clase en la que se aplica), un alias, un comentario, un mensaje de ayuda y precondiciones. Por otro lado, se pueden indicar los argumentos de entrada y salida del servicio indicando de cada uno de ellos las mismas características que en el caso de los atributos. Se pueden distinguir varios tipos de servicios:
  - Eventos: son servicios atómicos. En este tipo se incluyen los eventos para la creación y destrucción de una instancia de un objeto, eventos atómicos definidos por el usuario y que únicamente afectan al estado de un objeto y eventos compartidos, que afectan al estado de dos o más objetos.
  - Transacciones: son servicios cuya ejecución incluye dos o más eventos con la condición de que los diferentes estados por los que pasa el objeto en la ejecución de los eventos que forman la transacción no son visibles al usuario y la ejecución se debe llevar a cabo completa, es decir, siempre se ejecutarán todos los eventos que conforman la transacción.
  - Operaciones: son servicios cuya ejecución incluye dos o más eventos sin que se deban cumplir necesariamente las condiciones indicadas para las transacciones.
- Precondiciones: una precondición es una fórmula booleana bien formada que se puede definir utilizando constantes, funciones, atributos del servicio sobre el que se aplica la misma.
- Agentes: son clases cuya función es activar los servicios de otras clases. Así pues para cada servicio de una clase no agente se debe definir la clase agente responsable de su ejecución.

- Relaciones: la cardinalidad de las relaciones en OO-Method se define desde dos perspectivas. Dadas dos clases A y B, las cardinalidades mínima y máxima se definen de A respecto a B y de B respecto a A. Además, una relación se puede definir como estática o dinámica. Una relación establecida como estática implica que los objetos involucrados no se pueden modificar por un tercero. Esta restricción no se cumple en el caso de que sea dinámica.

2.4.2. Modelo Dinámico

En el Modelo Dinámico se especifican aspectos relacionados con el comportamiento dinámico del sistema informático. Con los servicios definidos en el Modelo de Objetos se especifican los diferentes cambios de estado que pueden ocurrir mientras la instancia de un objeto permanece en el sistema. Sin embargo, no define ninguna restricción respecto a la aplicación de los mismos. La secuencia de vida válida de un objeto se define como el hecho de que los cambios de estado hayan ocurrido en un orden lógico. Esto significa que una vez definidos los servicios disponibles para cada clase, en el Modelo Dinámico se especifica el orden en el éstos que deben ocurrir. El objetivo de este modelo es otorgar la capacidad a OO-Method de determinar este orden. Por ejemplo, si las reglas de negocio de una determinada empresa determinan que las facturas no se pueden modificar una vez archivadas, el Modelo Dinámico reflejará esta restricción. Las secuencias de vida válidas de un objeto se definen mediante un Diagrama de Transición de Estados. Este diagrama contiene dos tipos de elementos:

- Estado: representan las diferentes situaciones particulares en las que se puede encontrar un objeto. Se definen tres tipos de estados: De pre-creación o inicial, representa la situación inmediatamente anterior a la creación de un objeto; de destrucción o final, representa la situación inmediatamente posterior a que un objeto se destruya; y estados simples que representan cualquier otro estado del objeto.
- Transiciones: asocian dos estados del objeto e indican que ese cambio de estado está permitido. Para ello, cada transición está asociada a un servicio de forma que si sobre un objeto que se encuentra en un estado n se ejecuta el servicio x, el estado del objeto pasará a ser el estado asociado a n mediante x. El Modelo de Objetos permite asociar condiciones de control sobre las transiciones para que esas transiciones no estén disponibles a no ser que se cumpla esa condición.

Por otro lado, este modelo permite definir transacciones globales que incluyen servicios de clases diferentes pero se ejecutan como una sola unidad, además permite determinar los servicios que están disponibles para ser ejecutados si se cumple una determinada condición.

Capítulo 3

Propuesta de Modelado a Empresarial

En este capítulo se describe un conjunto de metamodelos que permiten modelar la empresa a nivel CIM. Además, se propone una guía para llevar a cabo el modelado centrada en facilitar esta tarea a las personas no familiarizadas con la misma. Finalmente, se describe un método para discriminar los elementos del modelo empresarial que no forman parte del modelo informático.

3.1. Adaptación de la Propuesta MDK

En esta sección se describen los cambios realizados respecto a la Propuesta MDK para el modelado empresarial. La Propuesta MDK está orientada a modelar el conocimiento que maneja la empresa, los cambios realizados en la misma están orientados a permitir el modelado del dominio empresarial. Para ello, se han modificado los metamodelos ya que son los que determinan el tipo de elementos y las relaciones que se pueden establecer en el modelo. Los principales cambios efectuados sobre los metamodelos son los siguientes:

- Incorporación de elementos: la propuesta original propone un conjunto de metamodelos que permiten modelar la empresa desde el punto de vista del conocimiento que maneja. El trabajo realizado incorpora nuevos elementos a los metamodelos con el objetivo de modelar la empresa desde la perspectiva de los objetos del mundo real que maneja y su funcionamiento, independientemente de un sistema informático.

- Incorporación de un conjunto de relaciones inter-metamodelo: estas relaciones permiten completar el modelo empresarial mediante la asociación de diferentes elementos a través de los metamodelos. El objetivo de realizar estas asociaciones es determinar la función de un mismo elemento en las diferentes dimensiones de la empresa. Por ejemplo, el *Diagrama de Reglas de Negocio* ya no estaría aislado, las nuevas relaciones permitirían determinar sobre qué procedimiento se aplican qué reglas de negocio y qué elementos participan, información que no se puede deducir de un modelo creado con la propuesta original. Por otro lado, las relaciones inter-metamodelo además de completar el modelo, también son de utilidad en el momento de establecer las reglas de transformación entre los elementos de los metamodelos de nivel CIM y los definidos en el enfoque OO-Method. Permiten asociar los elementos estructurales con su comportamiento y restricciones de integridad (formuladas en forma de reglas de negocio) que se corresponde con la estructura de una aplicación diseñada con un enfoque Orientado a Objetos.

Para la definición de las modificaciones a realizar de la Propuesta MDK se han tenido en cuenta una serie de premisas establecidas con el objetivo de mantener la abstracción respecto a un sistema informático. Estas premisas son las siguientes:

1. La denominación de los elementos que componen los metamodelos se ha realizado, dentro de lo posible, con un lenguaje alejado del lenguaje propio de los sistemas informáticos.

2. Los elementos que se modelan son respectivos a la empresa, es decir el hecho de no formen parte de un sistema informático no implica que no deban aparecer en el modelo empresarial.

3. La definición de elementos y procesos que conforman la empresa es la misma independientemente de que estos procesos se automaticen o no.

4. La adaptación realizada de la Propuesta MDK proporciona un lenguaje de modelado que permite la definición de los conceptos y procedimientos empresariales de forma que faciliten la definición de reglas de transformación a un lenguaje de nivel PIM para reutilizar la información plasmada en el modelo CIM y generar la mayor parte del modelo destino.

Las adaptaciones propuestas para la representación de la empresa se encapsulan dentro de una estructura de datos que se ha denominado **Knowledge Cube**. El Knowledge Cube es una estructura de datos diseñada con el objetivo de representar las inter-relaciones entre los modelos de una empresa en forma de una estructura de datos manejable por un computador. El Knowledge Cube se puede tratar como un grafo compuesto por una serie de niveles, siendo cada nivel un diagrama del modelo CIM. En este grafo se tendrían dos tipos de aristas: las que relacionan los elementos dentro de un mismo nivel o diagrama (intra-metamodelo) y la instanciación de las relaciones inter-metamodelo entre los diferentes niveles. Por lo tanto, un elemento del dominio empresarial queda conectado a las reglas de negocio que debe satisfacer, así como a los procesos en los que participa, tareas y responsables de llevarlas a cabo mediante las relaciones inter-metamodelo. A lo largo de este capítulo, se presentan las modificaciones realizadas de la Propuesta MDK así como el funcionamiento del Knowledge Cube.

3.1.1. Descripción de los elementos incorporados

El cambio principal realizado respecto a la Propuesta MDK original está centrado en el nivel CIM-Knowledge, en concreto en el 'Metamodelo de Conocimiento'. En este trabajo el 'Modelo de Conocimiento' pasa a ser el 'Modelo de Dominio' con su correspondiente 'Metamodelo de Dominio', ya que el objetivo es modelar el dominio empresarial. El resto de metamodelos se han mantenido como en la propuesta original dado que se adaptan al modelado de las diferentes dimensiones de la empresa y por lo tanto no ha sido necesaria su modificación. En la figura 3.1 se muestra el 'Metamodelo de Dominio' que permite modelar los elementos del dominio empresarial y los procesos en los que participan.

Los principales cambios realizados respecto al metamodelo original es que se han descartado los elementos relativos al modelado del conocimiento, en concreto los elementos: el elemento *KnowledgeBlock*, *OntologicalCategory*, *TargetKnowledge* y *InstanceKnowledge*.

Para poder realizar el modelado empresarial, se han incorporado los siguientes elementos (resaltados en azul en la figura 3.1): *DomainBlock* para representar los bloques del dominio empresarial, *Domain* para subdividir los bloques del dominio y poder modularizar el modelado empresarial, *DomainElement* cuya función es representar los conceptos que abstraen a los objetos reales que utiliza la empresa para su funcionamiento y *Property* para definir las características de dichos elementos. Los elementos de tipo *DomainBlock* y *Domain* tienen las mismas características que los *KnowledgeBlock* y *OntologicalCategory* de la propuesta original y se ha utilizado la misma representación gráfica para representarlos en el modelo. A continuación, se enumeran las características concretas de los artefactos *DomainElement* y *Property*.

Un *DomainElement* se representa gráficamente como un *targetKnowledge* (véase tabla 2.1) y está formado por: un nombre que debe ser único en el modelo y que describe el conjunto objetos que representa, un valor de persistencia que puede tomar los valores *True* o *False*, según si un elemento del dominio se puede descartar o no en un futuro dentro del sistema empresarial y un conjunto de propiedades asociadas. En el estado actual del trabajo, las propiedades se representan gráficamente como clases de UML. Las propiedades permiten

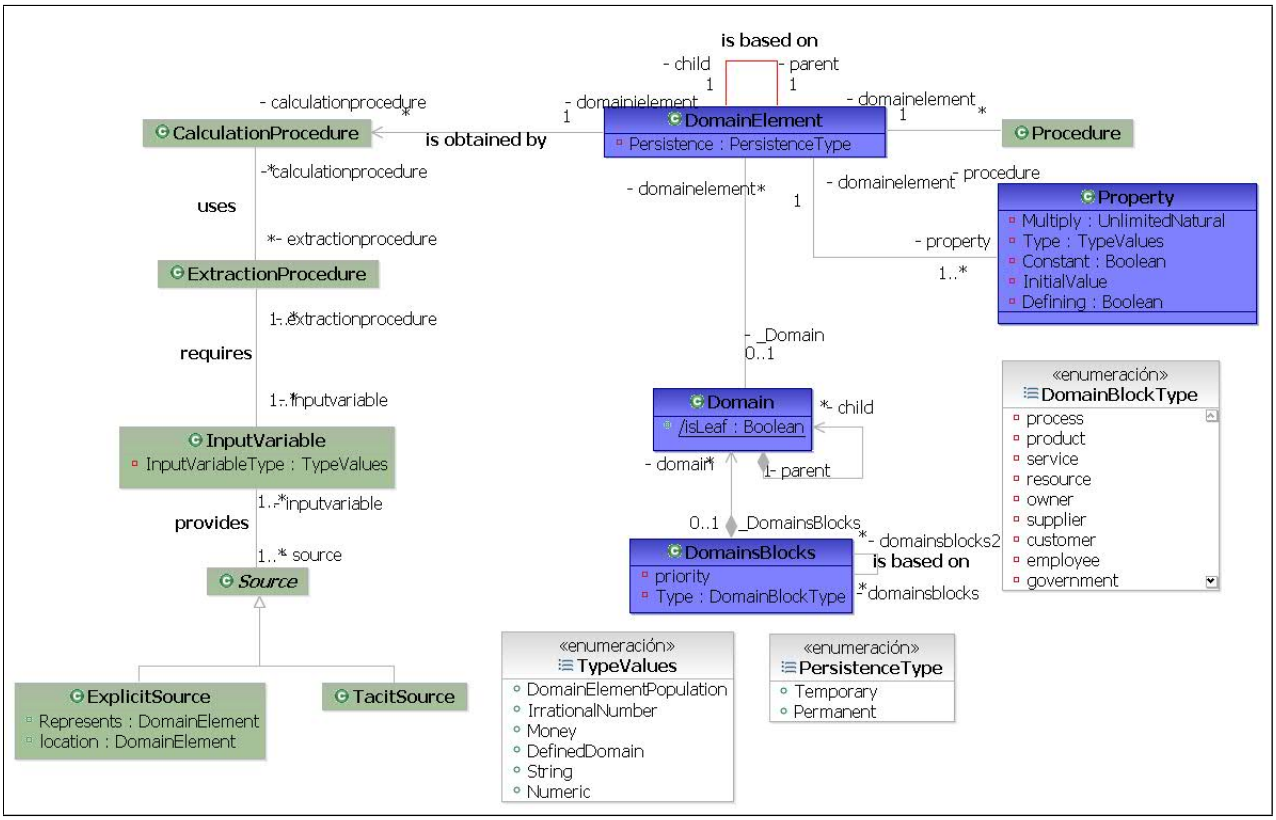

Figura 3.1: Metamodelo de Dominio

caracterizar los objetos y para cada *DomainElement* se debe indicar al menos una propiedad. Para cada una de las propiedades establecidas sobre un elemento del dominio, se deben indicar los siguientes aspectos:

- *Defining*: Para indicar si una propiedad es definitoria o no. El conjunto de propiedades definitorias es aquél que es necesario y suficiente para que una instancia del mundo real pertenezca a la categoría de un elemento del dominio.
- *InitialValue*: Indica si la propiedad tiene un valor inicial determinado, este atributo puede ser una fórmula matemática o un valor atómico.
- *Constant*: Indica que una vez establecido el valor de la propiedad se puede modificar o no.
- *Type*: Puede tomar los valores: numérico, texto, dominio definido, fecha, moneda, irracional o población de elemento. El tipo población de elemento indica que una propiedad toma valores de las instancias (la población) de otro DomainElement presente en el modelo.
- *Multiply*: Indica la multiplicidad de un atributo.

Además, se ha introducido la relación *isBasedOn* para facilitar la transformación de relaciones de agregación y composición tal y como se explica en el capítulo 4.

Finalmente, se ha añadido el atributo *type* para los flujos de tipo *input* del 'Metamodelo de Comportamiento'. Este elemento se ha incorporado con el fin de establecer los tipos de datos que participan en los procesos y facilitar la transformación a nivel PIM. Puede tomar los mismos valores que el atributo *Type* de las propiedades.

3.2. Knowledge Cube

En esta sección se detallan las relaciones inter-metamodelo que conforman el Knowledge Cube y cuya instanciación permite tratar el modelo como si fuera un grafo en el que cada nodo es un diagrama del modelo y del que entran y salen aristas que pueden proceder de otros modelos. En la figura 3.2 se representa geométricamente el Knowledge Cube cuyo nombre viene determinado por la forma cúbica que obtendríamos si se representara cada diagrama como un cuadrado y éstos se unieran mediante las relaciones inter-metamodelo.

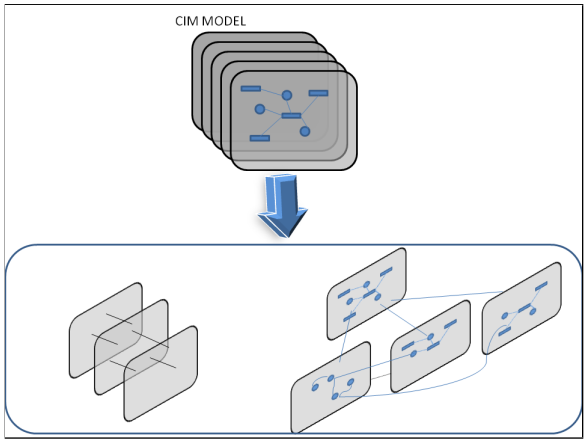

Figura 3.2: Representación geométrica del Knowledge Cube

Las relaciones inter-metamodelo se han definido de forma gráfica formando un metamodelo (ver figura 3.3) cuyos elementos están presentes en los metamodelos de la Propuesta MDK adaptada al modelado empresarial. La leyenda de la figura 3.3 es la siguiente: en azul oscuro se muestran los elementos que pertenecen al 'Metamodelo de Dominio', en verde los que pertenecen al 'Metamodelo de Comportamiento', en amarillo los que pertenecen al 'Metamodelo de Organización', en morado los que pertenece al 'Metamodelo de Estructura', en azul claro se han representado los elementos que no existen realmente en los metamodelos y representan agrupaciones de elementos con el fin de simplificar la representación gráfica y en rojo las relaciones inter-metamodelo establecidas.

Así pues, las relaciones inter-metamodelo permiten establecer vínculos entre los elementos de los diferentes metamodelos. Además, estos vínculos tienen un significado determinado que ayudan a determinar el papel de un elemento de un metamodelo para una determinada dimensión de la empresa en otros metamodelos y por lo tanto en otras dimensiones. A continuación, se explica el significado de cada una de estas relaciones.

isResponsabileOf

La relación *isResponsabileOf* relaciona elementos de tipo *Profile*, *Rol* o *Employee* del 'Metamodelo de Organización' con elementos de tipo *TacitSource* del 'Metamodelo de Dominio'. Esta relación permite definir que personas son responsables de ser la fuente tácita para la obtención de la información asociada a un elemento del dominio. La cardinalidad de esta relación se ha determinado como $N : N$, para no obligar a definir ningún responsable de una fuente tácita y permitir la definición de cuantos sea necesario.

instancedBy

La relación *instancedBy* relaciona elementos de tipo *Human* del 'Metamodelo de Estructura' con elementos de tipo *Profile*, *Rol* o *Employee* del 'Metamodelo de Organización'. Esta relación permite definir que personas

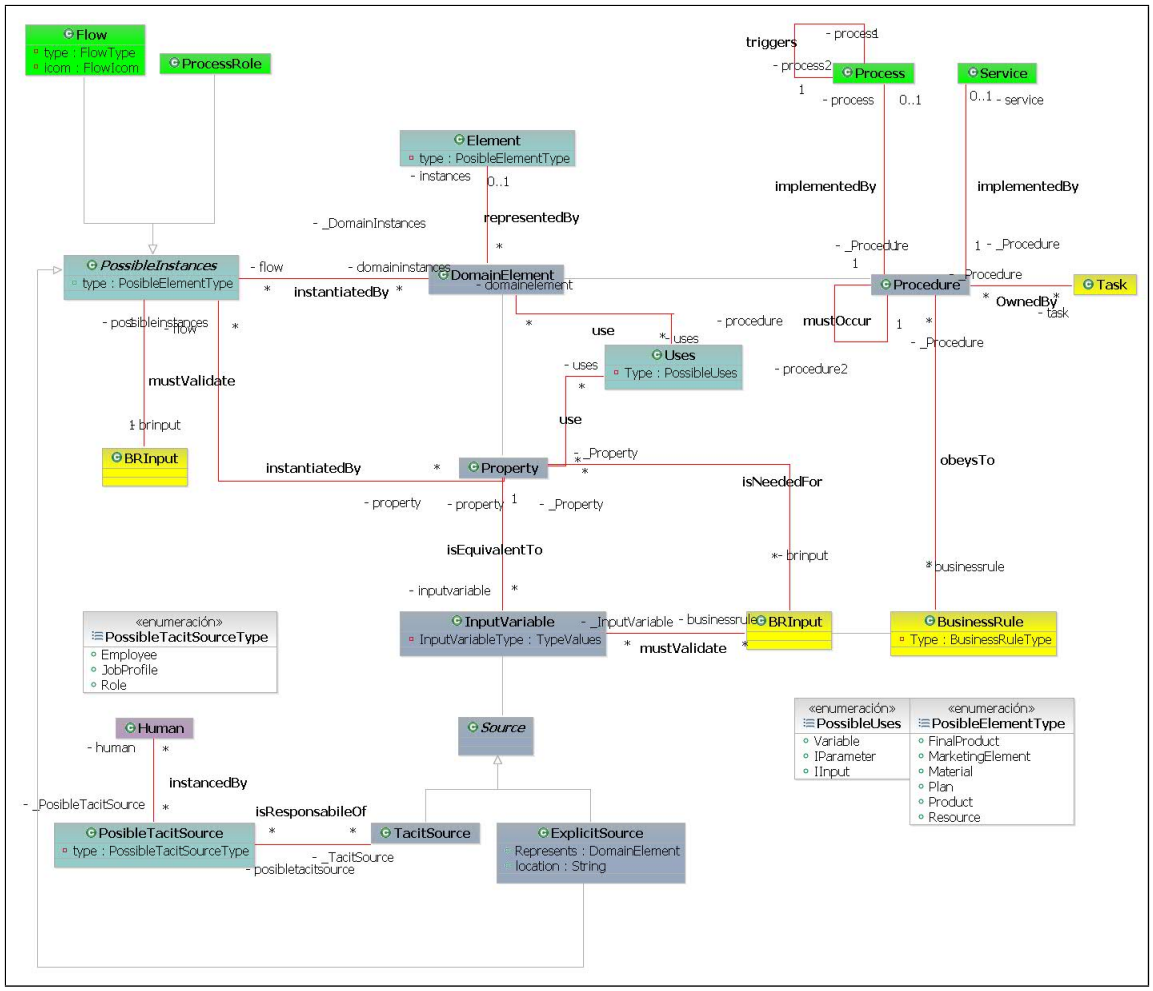

Figura 3.3: Relaciones inter-metamodelo

pertenecen a un perfil determinado o juegan un papel determinado en la empresa. La cardinalidad de esta relación no impone ninguna restricción ($N : N$) ya que una persona puede tener varios perfiles o roles y viceversa. El caso del tipo empleado no es diferente, ya que denota un empleado cualquiera independientemente de su perfil o rol.

use

La relación *use* relaciona elementos de tipo *Property* o *DomainElement* del 'Metamodelo de Dominio' con elementos de tipo *IInput*, *IParameter* o *Variable* del 'Metamodelo de Organización'. Esta relación permite definir que propiedades se utilizan como entradas, parámetros o variables de los planes estratégicos empresariales y los centros de análisis. La cardinalidad de esta relación no impone ninguna restricción ($N : N$) ya que una propiedad se puede utilizar en cualquiera de estos ámbitos. Para mejorar la legibilidad del modelo se ha creado la clase *uses* que no existe realmente y cuyo atributo *type* toma los valores de los tipos de elementos con los que se puede relacionar.

mustValidate

La relación *mustValidate* relaciona elementos de tipo *inputVariable* del 'Metamodelo de Dominio' con elementos de tipo *BRInput* del 'Metamodelo de Organización'. Esta relación permite indicar que una variable de entrada de un procedimiento de extracción se utilice como entrada de una regla de negocio para verificar si es válida en el contexto empresarial. En este caso tampoco se imponen restricciones respecto a la cardinalidad.

isNeededFor

La relación *isNeededFor* relaciona elementos de tipo *property* del 'Metamodelo de Dominio' con elementos de tipo *BRInput* del 'Metamodelo de Organización'. Esta relación es similar a la anterior (por lo que tampoco se imponen restricciones respecto a la cardinalidad), pero en este caso las propiedades no tienen por qué ser el elemento a validar, sino que simplemente pueden ser parte del funcionamiento de la regla de negocio.

implementedBy

La relación *implementedBy* relaciona elementos de tipo *Procedure* del 'Metamodelo de Dominio' con elementos de tipo *Process* o *Service* de 'Metamodelo de Comportamiento'. Tal y como su nombre indica, la relación *implementedBy* es para indicar qué procedimiento está implementado mediante qué proceso o servicio. La cardinalidad mínima de esta relación se ha establecido en 0 para no imponer ninguna restricción respecto al modelado y así no obligar al usuario final a modelar todos los procedimientos empresariales como procesos o servicios. La cardinalidad máxima es 1 ya que un procedimiento no puede estar implementado por más de un proceso o servicio al mismo tiempo. Esto supone que tampoco se debe permitir que un procedimiento se relacione al mismo tiempo con un proceso y un servicio, sin embargo la representación gráfica del 'Metamodelo de Relaciones' no permite expresar este hecho gráficamente, para hacerlo se debería especificar mediante una restricción expresada en un lenguaje como OCL (Object Constraint Language [25]).

ownedBy

La relación *ownedBy* relaciona elementos de tipo *Procedure* del 'Metamodelo de Dominio' con elementos de tipo *Task* del 'Metamodelo de Organización'. El objetivo es ofrecer la posibilidad de relacionar las tareas con los procedimientos que lo componen y crear un nexo de unión con los perfiles de trabajo que deben llevar a cabo dicha tarea (relacionados de forma directa en el 'Metamodelo de Organización'). Dado que una tarea puede estar formada por cero o más procedimientos y un mismo procedimiento puede formar parte de más de una tarea no se han impuesto restricciones de cardinalidad para la relación.

obeysTo

La relación *obeysTo* relaciona elementos de tipo *Procedure* del 'Metamodelo de Dominio' con elementos de tipo *BusinessRule* del 'Metamodelo de Organización'. El objetivo es asociar a nivel de procedimiento las restricciones que éste debe cumplir. Se debe tener en cuenta que si un mismo procedimiento está relacionado mediante *ownedBy* a varias tareas, las reglas de negocio asociadas a dicho procedimiento se aplicarán a todas las tareas. La decisión de establecer esta relación a nivel de procedimiento viene determinada por el hecho de que se han considerado las tareas como una agrupación conceptual de procedimientos, siendo los procedimientos los que realmente se ejecutan en la empresa. Por lo tanto, es dicha ejecución la que debe cumplir una cierta regla de negocio si así corresponde.

representedBy

representedBy es una relación de identidad entre un elemento de tipo *DomainElement* del 'Metamodelo de Dominio' y cualquier elemento de cualquier otro metamodelo que lo represente. Para representar gráficamente esta relación se ha creado un nuevo tipo *Element* que no existe realmente y únicamente se ha añadido con el fin de simplificar la representación. El atributo *type* y sus posibles valores, definidos en la enumeración *PossibleInstanceType*, indican los elementos con los que se puede establecer esta relación. La cardinalidad se ha definido como 0 : 1 debido a que para cada tipo de elemento se puede establecer esta relación como máximo con 1 elemento que lo instancie. En el sentido contrario la cardinalidad sería 0:N ya que un elemento del dominio puede estar representado mediante diferentes roles con distintos elementos.

isEquivalentTo

La relación *isEquivalentTo* relaciona elementos de tipo *Property* con elementos de tipo *inputVariable* ambos pertenecientes al 'Metamodelo de Dominio'. El objetivo es vincular las variables de entrada con las propiedades que caracterizan los elementos del dominio empresarial. En este caso, la cardinalidad máxima se ha restringido a 1 en el sentido de que una propiedad únicamente puede estar asociada a una variable de entrada, sin embargo en el sentido contrario la cardinalidad no presenta restricciones ya que una variable de entrada podría estar asociada a más de una propiedad.

instantiatedBy

La relación *instantiatedBy* relaciona elementos de tipo *Property* o *DomainElement* del 'Metamodelo de Dominio' con elementos de tipo *Flow::Other* o *ProcessRole* del 'Metamodelo de Comportamiento'. El objetivo es asociar los elementos estructurales expresados en el 'Modelo de Dominio' con la función que realiza en la dinámica o el comportamiento de la empresa. En esta relación no se han impuesto restricciones respecto a la cardinalidad.

mustOccur

La relación *mustOccur* relaciona entre sí elementos de tipo *Procedure* del 'Metamodelo de Dominio'. El objetivo es tener la capacidad de modelar una relación de orden de ejecución de los procedimientos empresariales. La cardinalidad máxima se ha restringido a 1 con el fin de evitar el indeterminismo en la la ejecución de dichos procedimientos.

triggers

La relación *triggers* relaciona entre sí elementos de tipo *Process* del 'Metamodelo de Comportamiento'. El objetivo es tener la capacidad de modelar una relación de orden de ejecución de procesos que pertenecen a diferentes procedimientos y que siempre deben ejecutarse secuencialmente y como si fueran un única unidad de ejecución. Como en el caso de la relación *mustOccur*, la cardinalidad máxima se ha restringido a 1 con el fin de evitar el indeterminismo en la la ejecución de dichos procesos.

3.2.1. Consideraciones sobre las relaciones inter-metamodelo

El conjunto de relaciones inter-metamodelo se ha creado con el objetivo de relacionar los metamodelos para que los modelos creados a partir de los mismos no sean visiones aisladas ya que, por ejemplo, un

modelo de reglas de negocio que no incluye sobre que procedimientos se aplican o un conjunto de procesos que no se sabe que procedimientos empresariales implementan no permite tener una percepción completa del sistema empresarial. Sin el conjunto de relaciones inter-metamodelo, este conocimiento se considera tácito ya que los empresarios sí tienen en mente estas asociaciones. Por lo tanto, uno de los objetivos de crear el Knowledge Cube mediante este tipo de relaciones es hacer explícito este conocimiento de forma que se pueda reutilizar en otros ámbitos y no se pierda si las personas que lo poseen abandonan la empresa. Por lo tanto, el Knowledge Cube hace que el conjunto de modelos que representan la empresa que hasta ahora estaba formado por elementos independientes (independientes en el sentido de que las relaciones no estaban explícitamente en el modelo), pase a ser un conjunto de elementos relacionados entre sí, cuyas relaciones tienen una semántica precisa que permite completar la visión de la empresa que ofrece el modelo. Además, estas relaciones se utilizan en el proceso de transformación a un modelo informático tal y como se explica en el capítulo 4. Así pues, el Knowledge Cube tiene un papel crucial en el desarrollo de este trabajo ya que es el nexo de unión entre la propuesta de modelado y la transformación del modelo.

3.3. Proceso de Desarrollo del Modelado Empresarial con la Propuesta MDK

Una empresa es un sistema complejo y por lo tanto, antes de comenzar a realizar un modelo completo de este sistema se debe reflexionar sobre el perfil de las personas que van a participar en el modelado con el fin de crear un proceso de desarrollo que facilite al máximo esta tarea. En general, se tendrían dos tipos de perfiles: los modeladores y los empresarios.

Los modeladores son las personas especialistas en el modelado empresarial que conocen perfectamente los metamodelos del lenguaje de modelado, la semántica de cada uno de los elementos y que, además, conocen el funcionamiento general de las empresas.

Los empresarios denotan al conjunto de personas que trabajan en la empresa. Estas personas no tienen por qué conocer en absoluto el lenguaje de modelado ni el concepto de modelado en sí mismo. Además, se debe considerar el hecho de que crear un modelo significa abstraer la realidad para representarla en un determinado lenguaje de modelado y esta tarea no siempre resulta fácil para personas no familiarizadas con este tipo de actividad.

Así pues, se propone una guía constituida por una serie de pasos a seguir con el objetivo de facilitar el modelado empresarial y la tarea de abstracción. Los pasos propuestos están centrados en el modelado de un sistema transaccional. A continuación, se enumeran y describen los pasos a seguir para desarrollar el modelo empresarial.

Paso 1 Crear el Diagrama de Estructura Organizativa. Empezar creando el organigrama de la empresa a modelar permite a los modeladores hacerse una idea bastante clara de los departamentos que componen la empresa, los perfiles y roles de trabajo y además obtener un listado completo de las tareas que lleva a cabo cada departamento. Por otro lado, permite a los empresarios comenzar en un entorno cómodo y que dominan para así empezar la tarea de modelado desde algo más concreto y no creando el Modelo de Dominio que es más abstracto y por lo tanto, puede resultar más difícil para los empresarios.

Paso 2 Descomponer las tareas en los procedimientos que la conforman. Esto supone comenzar a crear el Diagrama de Dominio desde el punto de vista del funcionamiento transaccional de la empresa que es el más tangible para los empresarios. A medida que se van incorporando los procedimientos se debe ir pensando en las reglas de negocio que deben cumplir los procedimientos y así, al mismo tiempo ir creando el Diagrama de Reglas de Negocio. Además se debe establecer la secuencia de ejecución válida de dichos procedimientos si así procede (relación *mustOccur*). Se debe tener en cuenta, que los procedimientos que indican la entrada en la empresa de una nueva instancia de un elemento del dominio no se representan como procedimientos, ya que quedan definidos mediante los procesos de extracción

y cálculo. En este caso, son los modeladores los que deben tener en cuenta este aspecto al realizar el modelo.

Paso 3 Incorporar los elementos del dominio sobre los que se ejecutan los procedimientos y completar el Diagrama de Reglas de Negocio con las características que participan en la validación de las reglas de negocio. Este paso debe estar guiado por los modeladores que son los que conocen las relaciones inter-metamodelo.

Paso 4 Implementar los procedimientos con los Diagramas de Procesos y Servicios, instanciar los flujos y roles de proceso con los elementos de dominio y propiedades que participan y establecer las relaciones *implementedBy* entre procedimientos y procesos y la relación *triggers* si es necesaria. En este punto deben intervenir los empresarios, que son los que conocen bien sus procesos de negocio.

Paso 5 Crear el Diagrama de Productos y/o Servicios. Una vez definidos los procedimientos, el siguiente paso es plasmar a nivel CIM los productos y servicios resultado de los procesos de negocio. Este paso va a permitir completar el Diagrama de Dominio con los nuevos elementos que pueden aparecer en este paso.

Paso 6 Completar, si es necesario, el Diagrama de Dominio con los roles de procesos, productos y servicios y establecer las relaciones inter-metamodelo de *instantiatedBy* y *representedBy*.

Paso 7 Agrupar los elementos del dominio en categorías y crear los bloques empresariales: Este es el nivel más abstracto y son los modeladores los que deben guiar a los empresarios para crear este tipo de diagramas. El valor de realizar este diagrama estriba en el hecho de que permite organizar los elementos del dominio y clasificarlos en función de los bloques que se definan proporcionando una visión conceptualizada de la empresa.

Paso 8 Finalmente, se representaría el plan estratégico creando el Diagrama de Objetivos y de Análisis. Esta tarea se ha dejado como la última para realizarla una vez se tiene una visión holística del modelo ya que permitirá crearlo conociendo todos los detalles de la empresa.

Uno de los objetivos de seguir esta guía es ir subiendo el nivel de abstracción poco a poco para facilitar la participación de los empresarios y al mismo tiempo que los modeladores vayan conociendo la empresa sobre la que están trabajando. Por otro lado, se debe tener en cuenta que el modelado se puede realizar desde dos perspectivas: modelar la empresa tal y como es (modelo *AS IS*) o modelarla tal y como los empresarios desean que funcione (modelo *TO BE*). Sin embargo, realizar el modelado desde ambas perspectivas permitiría conocer los cambios a realizar sobre el sistema actual para llegar a la situación deseada y reflexionar sobre si es realmente plausible llegar a ese estado desde el existente. De todas formas, la estrategia de modelado sería la misma independientemente del tipo de modelo que se vaya a realizar. La diferencia sería que al crear un modelo *AS IS* se modelan los procedimientos y procesos tal y como se ejecutan y al crear el modelo *TO BE* se modelan los procedimientos tal y como los empresarios desean que se realicen. Además, se podrían incorporar nuevos productos y planes de producción que los empresarios tengan en mente incluir en la empresa.

3.4. Extracción del modelo CIM a informatizar

Desde la perspectiva del modelado empresarial, el objetivo de las relaciones entre elementos de diferentes metamodelos es completar la visión holística que proporciona el Metamodelo de Dominio con la función que realizan en el resto de dimensiones de la empresa. Así pues, quedan relacionados los elementos del diagrama ontológico con cualquier elemento que lo represente en otro diagrama; las propiedades, entradas al sistema empresarial y procedimientos con las reglas de negocio que deben cumplir y los procedimientos con sus implementaciones como procesos o servicios.

Dentro del enfoque de MDA, y con el fin de reutilizar la información plasmada a nivel CIM para generar una parte del modelo PIM, estas relaciones se pueden considerar como asociaciones entre los elementos que

componen la estructura de la empresa y su comportamiento, de forma que encaja con las partes de un sistema informático: persistencia y lógica de funcionamiento. Sin embargo, no todos los elementos que se modelan a nivel CIM tienen por qué aparecer a nivel PIM. Así pues, es necesario disponer de un mecanismo que permita discriminar entre los elementos del nivel CIM que también forman parte del nivel PIM de aquéllos que no lo hacen.

Para realizar esta discriminación se debe tener en cuenta que los elementos que componen el modelo PIM representan un sistema informático que debe cumplir los requisitos impuestos por los usuarios y por lo tanto son subjetivos respecto a la visión del usuario final.

Tal y como se ha comentado anteriormente, el Knowledge Cube se puede representar como un grafo, que es una estructura de datos manejable por un computador. Así pues, la idea es representar el modelo como un grafo en la que cada nodo representa un elemento del modelo y cada arista las relaciones (incluyendo las relaciones inter-metamodelo) entre estos elementos, representando de esta forma el Knowledge Cube. A continuación, los usuarios, deben seleccionar los elementos del dominio, tareas y procedimientos que se desea que formen parte del sistema informático. Una vez seleccionados, automáticamente se propagan las selecciones por el grafo con aquellos elementos que son necesarios para crear el sistema informático. La propagación se realiza de la siguiente forma:

```
Para cada elemento del dominio:
  Seleccionar las propiedades definitorias.
  Seleccionar las variables de entrada equivalentes a las propiedades definitorias.
  Seleccionar los procedimientos de extracción y cálculo y los agentes.
  Para cada propiedad definitoria de tipo 'población de elemento de dominio' verificar
    que el elemento del dominio correspondiente está seleccionado.

Para cada tarea:
  Seleccionar todos los procedimientos asociados.
  Seleccionar cada proceso que implementa a los procedimientos.
  Seleccionar las propiedades que aparecen como flujo de entrada \\
    o con el estereotipo other que no sean definitorias \\
    y que pertenezcan a un elemento de dominio seleccionado.
  Seleccionar el perfil o rol responsable de realizar esa tarea.

Para cada procedimiento:
  Seleccionar la tarea a la que pertenece.
  Seleccionar cada proceso que lo implementa.
  Seleccionar las propiedades que aparecen como flujo de entrada \\
    o con el estereotipo other que no sean definitorias \\
    y que pertenezcan a un elemento de dominio seleccionado.
  Seleccionar el perfil o rol responsable de realizar esa tarea.
```

Así pues, bastaría con seleccionar los elementos del dominio que se desea que estén representados en el sistema informático y las tareas a informatizar, en el caso de que se informaticen todos los procedimientos asociados o los bien los procedimientos individuales. El resto de elementos, se seleccionan automáticamente ya que para cada elemento del dominio al menos deben representarse sus propiedades definitorias y cada procedimiento debe tener disponibles los elementos sobre los que actúa.

Para realizar esta tarea se necesita incorporar un nuevo atributo de tipo booleano a cada elemento de cada metamodelo de forma que si ese atributo es verdadero es porque debe formar parte del sistema informático y por lo tanto del modelo PIM. Una vez el usuario final ha decidido qué requisitos debe cumplir el sistema informático y ha seleccionado los elementos del modelo CIM que lo representan a este nivel, se pueden extraer los elementos seleccionados mediante algoritmos de model slicing ([26]). Dado que el Knowledge Cube se trataría como un grafo, el algoritmo de model slicing debería recorrer el grafo partiendo de los elementos

que ha seleccionado el usuario manualmente, y si un elemento se ha seleccionado se añade a un modelo intermedio, trasladando a este modelo intermedio las relaciones entre otros elementos si éstos elementos también se han seleccionado y forman parte de dicho modelo intermedio.

El modelo intermedio creado es que el que se debería transformar y generar el nivel PIM. Sin embargo, partiendo de una especificación de una empresa realizada con la Propuesta MDK no se puede generar un modelo PIM completo, ya que no cuenta con un lenguaje de especificación formal de los procesos, pero sí se puede reutilizar la información plasmada a nivel CIM para generar la mayor parte del nivel PIM. En el capítulo 4 se detallan los las reglas de transformación que se han establecido para generar parte del modelo PIM siguiendo el enfoque OO-Method.

Capítulo 4

Mapping a OO-Method

El trabajo realizado se completa con la definición de un conjunto de reglas de transformación de los elementos de la adaptación de la Propuesta MDK a los elementos del lenguaje de modelado definidos en el enfoque OO-Method. Las reglas se han definido con el objetivo de reutilizar la mayor cantidad posible de información representada a nivel CIM y generar parte del modelo PIM sobre el que añadir la información necesaria que permita generar código a partir del mismo.

Para poder establecer las reglas de transformación desde metamodelos de un nivel CIM a un nivel PIM se necesita un lenguaje a nivel CIM que esté bien estructurado, ya que en caso contrario sería casi imposible llevar a cabo la transformación [22]. En este aspecto, el lenguaje de modelado presentado en la Propuesta MDK simplifica el proceso de transformación al nivel PIM, debido a que los elementos que denotan tanto estructura y comportamiento están bien definidos. Los elementos que denotan comportamiento son aquéllos que aparecen en el 'Diagrama de Comportamiento' y los elementos de tipo *Procedure* del 'Diagrama de Dominio', el resto de elementos denotarían estructura. Además, la relación *instantiatedBy* permite saber que elementos participan como entrada o salida de los procesos de negocio.

Si se tiene en cuenta la representación del proceso de transformación de un modelo (véase figura 2.1), para llevarla a cabo se debe especificar información adicional que permita completar el modelo PIM de forma automática para completarlo y generar código a partir del nivel CIM. En este caso, la información adicional se añade al modelo PIM una vez éste se ha generado. En concreto, esta información es la siguiente:

- Especificación de Interfaces: el modelo CIM propuesto permite representar las principales dimensiones de la empresa. Sin embargo, la especificación de la interacción entre el usuario y una aplicación informática depende de las características que el usuario desee informatizar y debe tener en cuenta aspectos como la usabilidad o accesibilidad, por lo que se ha considerado que especificarlas a nivel CIM haría que el modelo generado perdiera parte de la abstracción respecto a la computación que caracteriza este nivel. En OO-Method esto se realiza creando el Modelo de Presentación.
- Especificación algorítmica de los procedimientos y procesos: el 'Modelo de Comportamiento' permite representar las entradas, salidas y mecanismos que utilizan los procesos y servicios empresariales, pero no indica el procesamiento específico que se realiza de los datos. El hecho de que especificar el procesamiento concreto de cada dato de cada proceso empresarial sea una tarea tediosa y larga medida respecto a la riqueza que aporta al modelo CIM, interpretado como modelo empresarial, hace que su especificación haya quedado descartada. Para ello, también se ha considerado el hecho de que los modelos empresariales son para los empresarios y la especificación algorítmica de un proceso puede resultar confusa si no se está familiarizado con este tipo de representaciones. Por otro lado, para llevar a cabo la especificación, se debería crear un álgebra cuyos tipos deberían ser capaces de representar los elementos que maneja la empresa y cuyas operaciones tengan la expresividad suficiente para crear algoritmos que indiquen cómo procesar esos elementos de una forma independiente a la computación

y permitiera representar qué debe hacer cada proceso. Esto conllevaría crear un álgebra adaptable a cada empresa en función de los elementos que maneja y los procesos que realiza y por lo tanto aumentaría mucho el coste temporal y la complejidad de realizar el modelo. Por otro lado, se debe añadir la complejidad de crear las reglas de transformación hacia el álgebra del modelo PIM al que se desee transformar desde el modelo empresarial, no del metamodelo ya que si se adapta a cada empresa se deben relacionar también estas adaptaciones del álgebra. Todas estas razones justifican el hecho de que se haya tomado la determinación de dejar el modelado de la lógica de procesos a nivel PIM, ya que los compiladores de modelos de este nivel ya tienen definidas álgebras computables y transformables a código.

En el resto de este capítulo se describen las reglas de transformación que permite generar los diferentes modelos que conforman el enfoque OO-Method. En primer lugar, se explican las relaciones establecidas entre los metamodelos de la Propuesta MDK y el Modelo de Objetos y a continuación se detalla la generación del Modelo Dinámico.

4.1. Generación del Modelo de Objetos

En esta sección se explican las asociaciones entre los elementos de la Propuesta MDK y los elementos que conforman el Modelo de Objetos de OO-Method. En primer lugar se definen las reglas que permiten generar la estructura de clases y relaciones, a continuación se indica como se genera la signatura de los servicios asociados a las clases y finalmente se detalla cómo se crean las clases estereotipadas como agentes.

4.1.1. Estructura de clases y relaciones

El elemento principal del 'Modelo de Objetos' son las clases y sus relaciones, ya que forman la estructura de datos que soporta al sistema informático de la empresa. En la Propuesta MDK los elementos que forman la estructura empresarial se definen mediante el 'Metamodelo de Dominio'. Por lo tanto, se deberán establecer las reglas de transformación a partir del 'Metamodelo de Dominio' para generar el 'Modelo de Objetos'. En concreto, las reglas serían como sigue:

- **Regla 1: DomainElement → Class**: Esta relación permite crear el conjunto de clases que darán lugar a la capa de persistencia de la futura aplicación informática. Al realizar la transformación, el nombre con el que se haya denominado el DomainElement pasará a ser el nombre de la nueva clase.
- **Regla 2: Property → Attribute**: Esta relación permite crear el conjunto de atributos de una clase para caracterizar al objeto del mundo real que representa. Al realizar la transformación, el nombre con el que se haya denominado la propiedad pasará a ser el nombre del nuevo atributo. Además, las características de la propiedad se asocian a las características de los atributos. En la tabla 4.1 se detallan los valores que tomaría cada característica de un atributo en función de los valores definidos a nivel CIM en las propiedades, en la primera columna se indica el número de regla asignado.

La regla 2.1 indica que si una propiedad está relacionada con una variable de entrada, entonces debe formar parte del servicio de creación de objeto y por lo tanto la característica 'request upon creation' (indicador que establece que la propiedad debe coincidir con un argumento de entrada del servicio de creación) debe ser verdadera. La regla 2.2 indica que si una propiedad se considera constante a nivel CIM también lo será a nivel PIM. La regla 2.3 indica que si una propiedad es definitoria, no permite nulos a nivel PIM. En caso de que no sea definitoria, implica que su valor se puede establecer mediante un cambio de estado y por lo tanto debe permitir nulos. Finalmente, la regla 2.4 establece las asociaciones entre los tipos de datos que se pueden utilizar en la propuesta de modelado a nivel CIM y los tipos de datos de OO-Method.

Tabla 4.1: Reglas de transformación para la caracterización de un atributo de OO-Method a partir de las propiedades de la Propuesta MDK

	Propuesta MDK		OO-Method	
R	Property	Valor	Attribute	Valor
2.1	relación isEquivalentTo	True	request upon creation	True
	no relación isEquivalentTo	False	request upon creation	False
2.2	Constant	True	Type	Constant
	Constant	False	Type	Variable
			añadir al servicio de edición	True
2.3	Defining	True	nulls allowed	False
	Defining	False	nulls allowed	True
2.4	Type	numérico	Data Type	Int
	Type	Texto	Data Type	String
	Type	moneda	Data Type	Real
	Type	irracional	Data Type	Real
	Type	fecha	Data Type	Date
	Type	dominio definido	No soportado directamente	

En la tabla 4.1 se han omitido dos características de las propiedades: la Multiplicidad y el tipo Población de Elemento. A continuación, se explican las asociaciones y tratamiento de los valores de cada una de estas características.

Regla 3: Tratamiento de la Multiplicidad

```
procedure tratarMultiplicidad (Propiedad p)
if p.Multiplicidad = 1 then
  do nothing
else
  crear nueva Clase c
  c.nombre = p.nombre
  // p.padre es el DomainElement al que pertenece
  // el atributo
  crearRelacion(nombreRelación, c, p.padre)
  if p.Defining then
    nombreRelación.cardMin = 1
  else if
    nombreRelación.cardMin = 0
  fi
  nombreRelación.cardMax = p.Multiplicidad
fi
fin tratarMultiplicidad
```

Es decir, si la multiplicidad de una propiedad es mayor que 1 se crea una nueva clase para representar el conjunto de valores que puede tomar la propiedad. Esta nueva clase que se ha creado constaría únicamente de un atributo para almacenar el valor, ya que al crear una nueva clase en OO-Method se crea automáticamente un identificador para la misma. Las características del atributo que representa el valor serían las mismas que la de la propiedad a partir de la cual se ha creado excepto en lo referente a la multiplicidad, que en este caso pasaría a ser 1.

Regla 4: Tratamiento Población de Elemento

```
procedure tratarPoblacionElemento(Property p)
  if p.tipo.nombre = "poblacionElemento" then
    if (crearRelacion(nombreRelacion, p.tipo.elementoDominio, p.padre)) then
      if p.defining then
        p.padre.nombreRelacion.cardMin = 1
      else
        p.nombre.nombreRelacion.cardMin = 0
      fi
      if p.Multiplicidad = 1 then
        p.nombre.nombreRelacion.cardMax = 1
      else
        p.nombre.nombreRelacion.cardMax = p.Multiplicidad
      fi
      // No se imponen restricciones de cardinalidad en el sentido contrario
      // además la relación se define como dinámica.
      p.tipo.elementoDominio.nombreRelacion.cardMin = 0
      p.tipo.elementoDominio.nombreRelacion.cardMin = N
      if not p.Constant then
        nombreRelacion.dinamica = True
      fi
    else if
  else if
fin tratarPoblacionElemento
```

A partir de las propiedades de tipo población de elemento se deducen las relaciones entre las clases del diagrama de clases del modelo de objetos. Este procedimiento crea una relación entre la clase que tiene esta propiedad como atributo y la clase generada a partir del elemento del dominio de la cual toma los valores el atributo. En OO-Method la cardinalidad de la relación se puede concretar del elemento A respecto al B y viceversa. En este caso, sólo se imponen las restricciones del elemento del dominio que se está tratando respecto a la relación, en sentido contrario se establece por defecto como $0 : N$. Se debe tener en cuenta que si no se ha seleccionado el elemento del dominio destino de la relación creada, es porque se ha decidido no almacenar datos sobre el mismo y por lo tanto no se crearía esta relación. Sin embargo, si la propiedad que se está tratando es definitoria entonces se debería avisar al usuario de que se es posible que se haya olvidado de seleccionar el elemento del dominio destino de la relación.

Agregación y composición La agregación y composición de objetos se calcula a partir de las instancias de la relación *isBasedOn* entre dos elementos de tipo *DomainElement* (véase 'Metamodelo de Dominio'). Según si la participación es obligatoria o no en la relación instanciada en el modelo se tratará de una agregación o una composición. La regla quedaría como sigue:

- **Regla 5: relación isBasedOn → Si relación.cardMin = 0 entonces tipoRelación = composición, sino tipoRelación = agregación**: La relación entre dos elementos del dominio de tipo isBasedOn se convierte en una relación de composición o agregación en función de la cardinalidad mínima establecida en el modelo.

4.1.2. Servicios

Los servicios en OO-Method modifican el estado de los objetos. De entre los posibles servicios relacionados con un objeto caben destacar tres tipos cuya semántica es muy precisa: servicio de creación, de edición y de destrucción. De estos tres tipos se dice que tienen una semántica precisa porque su semántica operacional es invariable independientemente del objeto sobre el que se aplique.

La finalidad del servicio de creación es que un objeto del mundo real quede representado en el sistema informático mediante el almacenamiento de las características del objeto en los atributos de la clase que lo representa en el sistema informático. Así pues, este servicio siempre formará parte del conjunto de servicios asociados a una clase. Tendrá como parámetros de entrada todos aquellos atributos marcados con 'request upon creation'.

El servicio de destrucción elimina la representación de un objeto en el sistema. Este servicio únicamente formará parte del conjunto de servicios si la característica de persistencia del *DomainElement* es `False`, ya que indica que en la empresa, o bien el objeto en sí o bien los datos del mismo, no tienen representación a lo largo del tiempo y por lo tanto tampoco es necesario mantener esta representación en el sistema informático. En este caso, se debe tener en cuenta que las instancias de las clases auxiliares creadas a partir de las propiedades con multiplicidad mayor que 1 también deberán desaparecer del sistema si lo hace el objeto principal a partir del cual se han creado.

La semántica operacional del servicio de edición de una clase es permitir modificar los valores de los atributos que caracterizan el objeto. Así pues, la creación de este servicio se puede automatizar para que se puedan modificar aquellos atributos cuyo tipo sea variable. Se debe tener en cuenta que si un objeto no tiene propiedades variables este servicio no formará parte del conjunto de servicios asociados a un objeto.

El resto de los procedimientos definidos en el modelo CIM, y seleccionados para formar parte del futuro sistema de información y por lo tanto del modelo PIM, también deberán formar parte del conjunto de servicios de un objeto. Así pues, por cada procedimiento seleccionado se creará un nuevo servicio siendo el nombre del procedimiento de nivel CIM el del servicio creado a nivel PIM.

En OO-Method se definen diferentes tipos de servicios: propios y compartidos. Los servicios propios afectan al estado de un sólo objeto, mientras que los compartidos afectan al estado de varios objetos. Así pues, si todos los flujos de tipo *other* van únicamente al elemento del dominio propietario de ese proceso, el servicio creado será de tipo propio, mientras que si tiene estereotipos *other* con destino a propiedades de más de un elemento del dominio, el servicio creado será de tipo compartido.

Los parámetros de entrada y los cambios de estado de cada servicio se obtienen a partir del 'Modelo de Comportamiento' de nivel CIM. De forma que si un procedimiento está relacionado con un proceso o servicio mediante la relación *implementedBy* los flujos marcados como entradas al proceso (estereotipo *input*) serán parámetros de entrada y, análogamente, los flujos marcados mediante el estereotipo *other* serán los cambios de estado.

Por otro lado, se debe tener en cuenta que los servicios no siempre son atómicos sino que pueden dar lugar a transacciones. Esta característica se deduce a partir de dos premisas. Por un lado, el hecho de que un proceso actúe como disparador de otro (relación *triggers*) implica crear una transacción que incluya todos los procesos relacionados. Por otro lado, se da un caso especial cuando el estereotipo *other* tiene como destino un elemento del dominio en lugar de una propiedad. Esto implica el registro de un nuevo elemento del dominio en la empresa forma parte del proceso, lo que se traduce como el hecho de crear una nueva instancia en un sistema informático. Por lo tanto, se crea una transacción que puede incluir un servicio de tipo interno creado automáticamente y cuya función es cambiar el estado de las clases que corresponda (estereotipo *other* a propiedades de elementos del dominio) e invocar al servicio de creación de elemento del dominio destino del flujo de tipo *other*.

Finalmente, las precondiciones que deben cumplirse para poder ejecutar un procedimiento así como las restricciones de integridad, se calcularían a partir de las reglas de negocio, si el procedimiento a partir del que se crean está relacionado con una regla de negocio mediante *obeysTo*, siendo los parámetros de entrada a la precondición las propiedades relacionadas mediante *isNeededFor* y las variables de entrada asociadas a la entrada de la regla de negocio mediante *mustValidate*.

A continuación, se detallan las reglas definidas para el tratamiento de los servicios, parámetros de entrada y restricciones. Se debe tener en cuenta que estas reglas se aplican a cada elemento del dominio seleccionado y los servicios se incorporan al conjunto de servicios de las clases creadas a partir de los mismos.

- **Regla 6: Añadir un servicio de creación**: Los parámetros de entrada coincidirán con las variables de entrada definidas como equivalencia de las propiedades y marcadas a nivel PIM con 'request Upon Creation'.
- **Regla 7: Añadir servicio de edición**: Si el elemento del dominio tiene propiedades no constantes, se añade el servicio de edición. En este caso los parámetros de entrada serán todas aquellas propiedades que tengan esta característica.
- **Regla 8: Añadir servicio de borrado**: Si la persistencia se establece como falsa, se añade el servicio de borrado.
- **Regla 9: Proceso → Servicio**. Casuística:
  - 9.1: Si todos los flujos *other* tienen como destino propiedades del elemento del dominio propietario del procedimiento, se añade un servicio de tipo propio.
  - 9.2: Si existe algún flujo *other* que tiene como destino propiedades de otros elementos del dominio, se añade un servicio compartido entre las clases que se corresponden con los elementos del dominio destino de este tipo de flujos.
  - 9.3 Si el proceso incluye un flujo estereotipado como *other* cuyo destino sea un elemento del dominio se crea una transacción que incluya un servicio propio que realice los cambios de estado sobre las propiedades pertinentes y la invocación del servicio de creación del elemento del dominio destino del flujo *other*. El caso en el que se registren varios elementos del dominio no supone ningún problema, ya que se invocaría a cada uno de los servicios de creación correspondientes. El orden de invocación de estos servicios será el mismo que el establecido mediante las actividades de los procesos en el nivel CIM. El nombre de la transacción sería: 'transN' siendo N un número entero, de forma que la primera transacción sería 'trans1', la segunda 'trans2', etcétera. El servicio creado para resolver el cambio de estado se marcaría como interno y no sería visible al usuario.
  - 9.4: Si el proceso es origen de una relación *triggers*: Crear una nueva transacción que incluya al proceso que se está tratando así como el proceso apuntado por *triggers* y al resto de procesos que pertenezcan a una secuencia de relación *triggers*. Nótese que los argumentos de entrada de la transacción coinciden con el conjunto de argumentos de entrada de todos los servicios involucrados. El nombre de la transacción sería el del servicio que es origen de la relación *triggers*. Los servicios que incluya la transacción se marcan como internos para que no sean visibles al usuario.
- **Regla 10: Flujo estereotipado como *input* → parámetro de entrada del servicio creado a partir del proceso al que pertenece**. Se debe tener en cuenta que este tipo de flujo será un parámetro de entrada siempre y cuando el elemento origen del flujo de entrada no sea destino de un flujo de cambio de estado (*other*).
- **Regla 11: Flujo estereotipado como *other* → cambios de estado realizados por el servicio creado a partir del proceso al que pertenece**. Si el flujo tiene como destino una propiedad de un elemento de dominio, entonces se trata de una valuación de dicha propiedad. Si el destino es un elemento de dominio en sí, entonces se invoca el proceso de registro de ese elemento.
- **Regla 12: Precondiciones**: Se añade una precondición a un servicio si existe una relación *obeysTo* entre un procedimiento y una regla de negocio. De la precondición añadida se crea la signatura y se indican los parámetros de entrada a partir de la relación *isNeededFor* y *mustValidate* de las propiedades y variables de entrada respectivamente.

4.1.3. Agentes

OO-Method define los agentes cómo clases que tienen permisos para ejecutar los servicios asociados a una determinada clase. En la Propuesta MDK se tienen dos clases de agentes. Por un lado, los representados como elementos del tipo *Source* con dos subtipos: *Tacit Source* y *Explicit Source*. Por otro lado, podemos

definir un agente a partir de la interrelación *ownedBy* de una tarea y un procedimiento y la relación entre la tarea y un perfil, rol o empleado de trabajo.

Si un elemento de tipo *DomainElement* tiene como origen una fuente tácita, esta relación se convertiría en una clase agente que represente la fuente tácita con permisos para ejecutar los servicios básicos de creación, edición y borrado sobre la clase.

Si lo que ocurre es que un procedimiento está relacionado mediante *OwnedBy* con una tarea y esta tarea está relacionada con un jobProfile o Rol se debería crear (si no existe ya en el Modelo de Objetos) un agente que representa a este rol, tenga permisos para ejecutar el servicio creado a partir del procedimiento y visibilidad sobre los parámetros de entrada y salida del mismo así como sobre las relaciones del objeto que estén involucradas. Así pues, estas relaciones permiten generar el sistema de permisos para el sistema informático. Las reglas establecidas para la generación de los agentes de OO-Method son las siguientes:

- **Regla 13: A partir de la relación *isResponsabileOf* entre un JobProfile/Role/Employee y una fuente tácita: se crea un agente con visibilidad al servicio de creación, edición y borrado del elemento del dominio relacionado con esta fuente tácita**. Además, este agente tendrá visibilidad sobre los atributos de la clase y las relaciones obtenidas a partir de las reglas 3 y 4 (tratamiento de la multiplicidad y del tipo población de elemento).
- **Regla 14: A partir de la relación *ownedBy* entre un procedimiento y una tarea se crea un agente que represente al perfil de trabajo responsable de realizar esta tarea**. El agente tendrá permisos para ejecutar los procedimientos asociados a la tarea y visibilidad sobre los parámetros de entrada y salida del mismo.
- **Regla 15: Incorporar un agente global del sistema.**

4.2. Generación del Modelo Dinámico

El Modelo Dinámico de OO-Method se compone de un conjunto de estados y transiciones que determinan las secuencias de acciones válidas para un objeto. OO-Method propone un un diagrama de estados por defecto para cada clase en el que no hay restricciones sobre la secuencia de ejecución de servicios.

Este diagrama de estados por defecto es válido siempre y cuando no existan relaciones de tipo *mustOccur* entre los procedimientos a partir de los cuales se han creado los servicios. En este caso, el orden de ejecución de los servicios debe respetar el orden impuesto a nivel CIM. Además, se debe tener en cuenta que cabe la posibilidad de que no todos los procedimientos estén marcados para formar parte del nivel PIM y como consecuencia no estarán representados en el diagrama de estados. Sin embargo, esto no supone ningún problema ya que la relación *mustOccur* es una relación de orden y por lo tanto es transitiva. Así pues, si un procedimiento no está representado a nivel PIM y está emplazado entre dos relaciones de tipo *mustOccur* (una de entrada y otra de salida) la relación de entrada pasaría al procedimiento apuntado por la relación de salida *mustOccur* y ésta se eliminaría. A continuación, se indica el algoritmo que permite generar el Modelo Dinámico.

- **Regla 16: Para cada clase del Modelo de Objetos cuyos procedimientos de nivel CIM no tengan inter-relaciones *mustOccur*, crear el Modelo Dinámico por defecto. En caso contrario, crear el Modelo Dinámico mediante el siguiente algoritmo:**

```
PASO 1: Eliminar procedimientos no seleccionados para formar
parte del nivel PIM.
PASO 2: Agrupar en un solo nodo los procedimientos que forman
parte de una misma transacción.
```

```
PASO 3: Crear estado inicial y un estado intermedio y unirlos
mediante una arista que indique la transición de creación
del objeto.
PASO 4: Crear estado final
PASO 5: Para cada procedimiento:
PASO 5.1: Crear un nuevo nodo en el MODELO DINÁMICO
PASO 5.2: Para cada nodo creado:
PASO 5.2.1: Añadir un arista con el nodo anterior. La etiqueta
de la transición coincide con el nombre del servicio
o transacción.
PASO 6: Si existe el servicio de destrucción:
PASO 6.1: Añadir una transición entre el nodo final de la secuencia
  de procedimientos y el nodo final.
PASO 7: Si existe el servicio de edición:
PASO 7.1: Crear una transición cíclica en cada nodo para el servicio de
  edición, excepto para el nodo inicial y final.
```

4.3. Conclusión

A lo largo de este capítulo se ha indicado cómo se reutiliza y establece la relación entre la información plasmada en un modelo CIM creado mediante la Propuesta MDK y el modelo PIM que se puede realizar siguiendo el enfoque OO-Method. Se han definido un total de 16 reglas de transformación para realizar la transformación de CIM a PIM.

Tal y como se ha comentado anteriormente el modelado de interfaces de usuario quedan fuera del alcance de este trabajo. Sin embargo, del modelo CIM se pueden deducir cuántas interfaces de usuario se van a tener cómo mínimo, ya que habrá una interfaz por cada servicio cuyo responsable de ejecución sea una fuente tácita. En dicha interfaz se deberán poder introducir los parámetros del servicio al que están asociado.

Así pues, a partir del modelo a nivel CIM se obtiene el Modelo de Objetos completo: clases, atributos y relaciones; la signatura de los servicios incluyendo los parámetros de entrada y los cambios de estado que se deben realizar; la signatura de las precondiciones a cumplir por cada servicio (relación *ObeysTo*); y el Modelo Dinámico que establece el orden de ejecución de los servicios en caso de que ésta sea relevante (relación *mustOccur*). Por lo tanto, para finalizar el modelo PIM bastará con definir el Modelo Funcional, indicando los cambios de estado que realizan los servicios, las restricciones de integridad, precondiciones y el Modelo de Presentación.

Capítulo 5

Ejemplo práctico

Para validar el trabajo expuesto en este documento se ha modelado parte de la especificación de requisitos del funcionamiento de una agencia de fotografía. Se ha utilizado la adaptación de la Propuesta MDK para modelar a nivel CIM y se ha creado un modelo PIM con el enfoque OO-Method siguiendo las reglas establecidas en el capítulo 4.

En este capítulo se explican los detalles de la realización del caso de estudio. Para realizar el modelado, se ha puesto en práctica la guía de modelado descrita anteriormente. Seguidamente, se ha extraído la parte del modelo CIM sobre la que crear el sistema informático y finalmente se han ejecutado las reglas de transformación para crear el modelo PIM correspondiente.

5.1. Descripción del Caso de Estudio

El caso de estudio desarrollado en este capítulo está enfocado en el funcionamiento actual de la empresa (modelo *AS IS*), ya que se parte del supuesto de que la empresa no desea modificar dicho funcionamiento, sino incorporar un sistema informático que permita agilizar los procesos que de momento se llevan a cabo de forma manual. Con el propósito de simplificar el ejemplo e ilustrar mejor el funcionamiento del trabajo expuesto en este documento, el ejemplo práctico se centra únicamente en un proceso de negocio de la empresa, en concreto en la gestión de reportajes de tipo exclusiva de una agencia fotográfica. En la figura 5.1 se muestra un ejemplo de la ficha de una exclusiva que utilizan en la agencia. Los clientes, editoriales en este caso, solicitan reportajes en exclusiva sobre temas que no están cubiertos por la agencia en los reportajes generales. Las editoriales estipulan el precio que están dispuestas a pagar por la exclusiva. Cada solicitud es atendida por el departamento técnico que abre una ficha de exclusiva asignándoles un código y anotando los datos del reportaje. Una copia de la ficha se deja en el tablón de anuncios.

Por otro lado, los fotógrafos están en contacto permanente con la agencia, para ver las exclusivas que se han solicitado por las editoriales. Si algún fotógrafo está interesado en una exclusiva, coge la ficha del tablón y va al departamento técnico para que se la asignen. Cada exclusiva se asigna a un solo fotógrafo, con la condición de que no tenga ninguna exclusiva previa pendiente de entregar. Cuando se asigna la exclusiva, se entrega al fotógrafo una copia de la ficha de exclusiva para que tenga todos los datos.

Cuando un fotógrafo finaliza una exclusiva la entrega en el Departamento de Producción junto con la copia de la ficha de la exclusiva. Allí se registra la fecha de entrega y emiten un cheque nominal por el 40 % del precio de la exclusiva (que es la parte que le corresponde al fotógrafo). La agencia envía la exclusiva a la editorial con un albarán a través de un mensajero. Una vez entregada la exclusiva, el mensajero devuelve el albarán firmado al Departamento Comercial de la agencia.

El proceso acaba con la facturación del importe de la exclusiva realizada por el departamento Comercial. A principios de cada mes, se revisan todos los albaranes para enviar una factura a cada editorial con el importe de los reportajes y exclusivas solicitadas.


FICHA DE EXCLUSIVA

Código Reportaje: X345
Título: La flora y fauna de Benicadell

Código Editorial: E444
EL ESPEJO
Reus, 36
46005 Valencia
Tel. 3333333

Importe: 1050€.
Observaciones: 12 placas de 6x6 a seleccionar 4.

Fotógrafo asignado:
23445468 Pascual Rovira Assensi

Fecha Petición: 23-02-94
Fecha Asignación: 12-12 94
Fecha Entrega: 14-1-95
conforme

Figura 5.1: Ficha de Exclusiva

5.2. Modelado

En esta sección se detalla el modelado de la Agencia Fotográfica, siguiendo los pasos establecidos en la sección 3.3 y representando en cada paso las relaciones inter-metamodelo que intervienen.

5.2.1. PASO 1: Diagrama de Estructura Organizativa

El primer paso para realizar el modelado empresarial es completar el organigrama de la empresa a modelar. En la figura 5.2 se expone el organigrama de la Agencia Fotográfica incluyendo las tareas que llevan a cabo. En este caso, no sólo se exponen las tareas relacionadas con las exclusivas, sino que se han incluido todas las tareas para dar una visión general del funcionamiento de la agencia.[1]

5.2.2. PASO 2: Descomposición de tareas en procedimientos

El segundo paso definido para el modelado, es la descomposición de las tareas en los procedimientos que las conforman. Las tareas relacionadas con las exclusivas son: Gestionar Exclusivas (Departamento Técnico) y Recoger y Enviar Exclusivas (Departamento de Producción). En este caso se ha omitido la facturación de exclusivas, ya que este proceso no incluye únicamente la facturación de exclusivas sino que también la de otro tipo de productos que oferta la agencia. En la figura 5.5 se representan gráficamente los procedimientos, resaltados en verde, obtenidos en este paso. Nótese, que el procedimiento 1.1 no aparece reflejado como tal. Esto se debe a que se caracteriza mediante los procedimientos de extracción y cálculo siendo los modeladores quienes deben saber que este proceso no se debe representar como un procedimiento. En este punto se incluye el elemento de dominio Exclusiva debido a que sobre el mismo se ejecutan los procedimientos. Así pues, cada tarea quedaría descompuesta en los siguientes procedimientos (relación *OwnedBy*):

[1] La información relativa al organigrama se ha extraído del texto completo que explica el caso de estudio y del que únicamente se ha incluido un resumen en este documento.

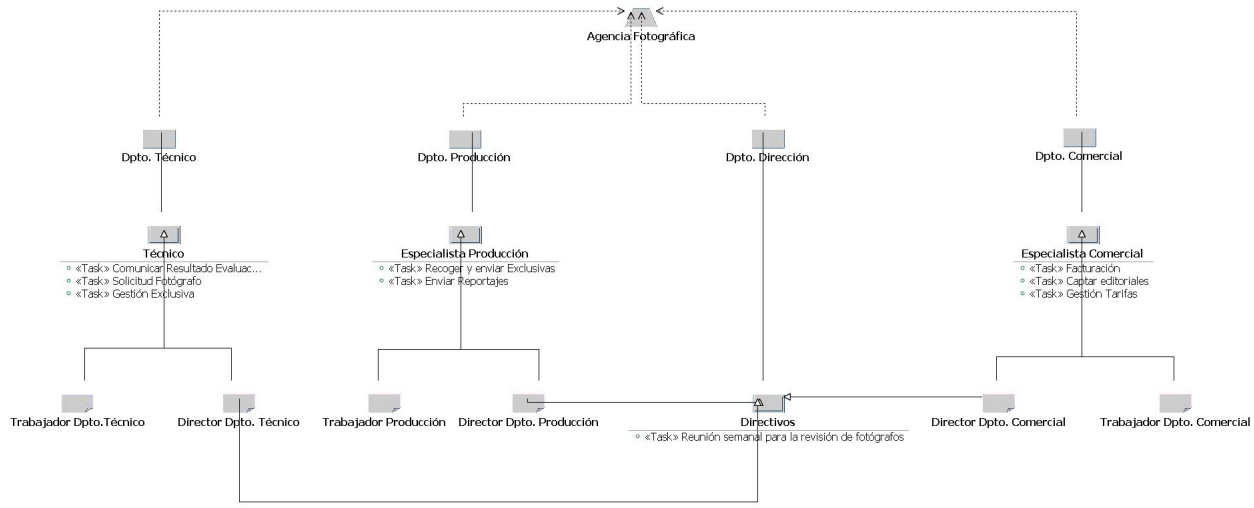

Figura 5.2: Diagrama de Estructura Organizativa

- TAREA 1: Gestión de exclusivas
  - PROCEDIMIENTO 1.1: Registrar los datos de la exclusiva.
  - PROCEDIMIENTO 1.2: Colgar anuncio en el tablón.
  - PROCEDIMIENTO 1.3: Asignar fotógrafo.
- TAREA 2: Recoger y Enviar exclusiva
  - PROCEDIMIENTO 2.1: Registrar entrega de la exclusiva.
  - PROCEDIMIENTO 2.2: Enviar la exclusiva

Este paso incluye establecer la relación *mustOccur* entre los procedimientos detectados si es necesario. En este caso, sí que se establece esta relación entre los procedimientos, ya que no se puede enviar una exclusiva si no se ha registrado su entrega y a su vez, no se puede registrar la entrega si no se ha asignado el fotógrafo. En la figura 5.3 se muestran la secuencia válida de ejecución de estos procedimientos.

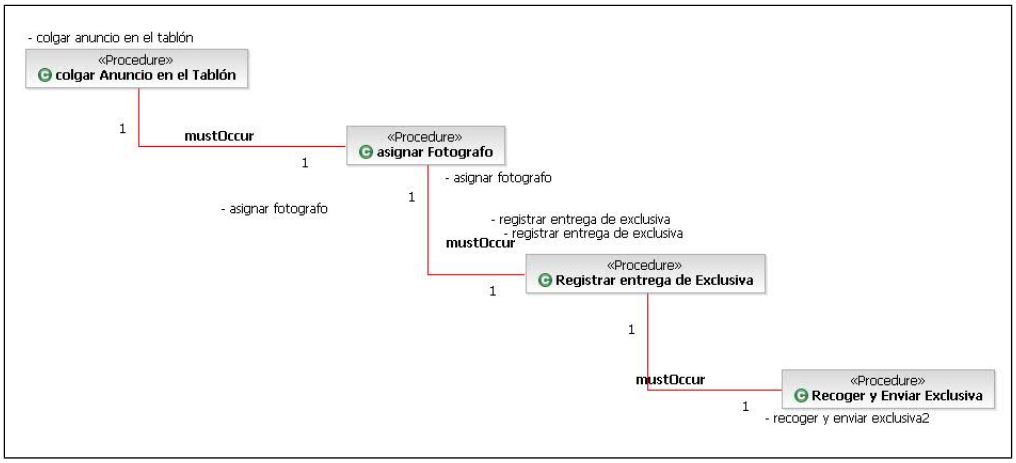

Figura 5.3: Relación *mustOccur* entre los procedimientos asociados a la Exclusiva

Por otro lado, se debe comenzar a esbozar el Diagrama de Reglas de Negocio, definiendo las reglas de negocio que deben cumplir los procedimientos en los que se descomponen las tareas (relación *obeysTo*). En este caso, la regla de negocio (identificada como REGLA 3.1) a cumplir por el procedimiento 1.3 es el hecho de que un fotógrafo no puede tener ninguna exclusiva pendiente de entrega para poder solicitar la siguiente. En la figura 5.4 se muestra la regla de negocio definida en este paso.

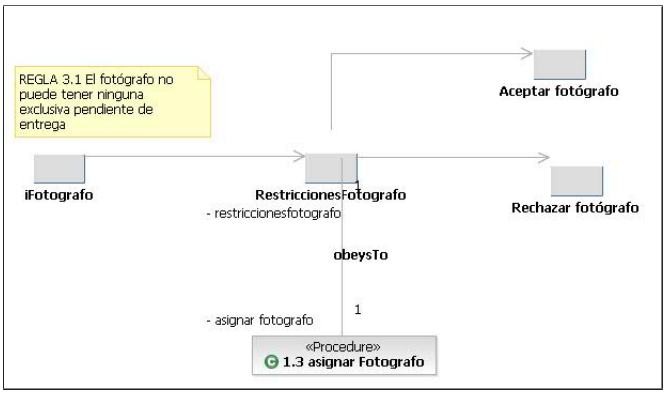

Figura 5.4: Diagrama de reglas de negocio y relación *obeysTo*

5.2.3. PASO 3: Elementos del Dominio

A continuación, se debe completar el Diagrama de Dominio con los elementos del dominio asociados a los procedimientos. Para detectar los elementos del dominio hay que reflexionar sobre los objetos y personas del mundo real que intervienen en los procedimientos. En este caso los elementos que intervienen son: la exclusiva en sí misma, la editorial que pide la exclusiva, el fotógrafo que la realiza y la empresa de mensajería que la entrega a la editorial. En la figura 5.5 se resaltan en azul los elementos incorporados al Diagrama de Dominio en esta fase, que son: Editorial, Fotógrafo y Empresa de Mensajería. Los valores de permanencia en este caso serían *True* para la Editorial y el Fotógrafo ya que los datos relativos a los mismos no se descartan y falso para el caso de la empresa de mensajería. Al incorporar los elementos del dominio se deben incluir también las propiedades de los mismos así como sus características. Dado que el modelo gráfico no permite visualizar las características de las propiedades, éstas se detallan en la tabla 5.1.

Por otro lado, al introducir las propiedades se incorporan también los procesos de extracción y cálculo de las mismas así como las variables de entrada. El caso de las variables de entrada se gestiona de forma muy simple, ya que se tiene una variable de entrada por cada propiedad definitoria (relación *isEquivalentTo*), además se pueden definir otras variables de entrada que aunque no sean definitorias pueden formar parte del registro de un objeto en el sistema empresarial. Finalmente, se deben detallar las fuentes de conocimiento que proporcionan los valores de las variables y asociarlas a los elementos del Diagrama de Estructura Organizativa en caso de que sea necesario (relación *isResponsabileOf*). En la parte izquierda de la figura 5.6 se muestran las relaciones de tipo *isEquivalentTo* establecidas entre las propiedades de la Exclusiva y las variables de entrada definidas. En la parte derecha de la misma, se muestran las relaciones de responsabilidad entre los perfiles de trabajo y el registro de datos de los elementos del dominio.

Los procedimientos de extracción y cálculo, las variables de entrada y fuentes de conocimiento únicamente se muestran para el elementos del dominio *Exclusiva* con el objetivo de que el modelo sea más legible. Sin embargo, no se debe perder de vista el hecho de que cualquier elemento del dominio contaría con estos procesos para permitir el registro de la entrada en la empresa del objeto al que abstrae el elemento del dominio correspondiente, independientemente de que este registro se lleve a cabo mediante la incorporación de datos en un sistema informático o completando una ficha en papel.

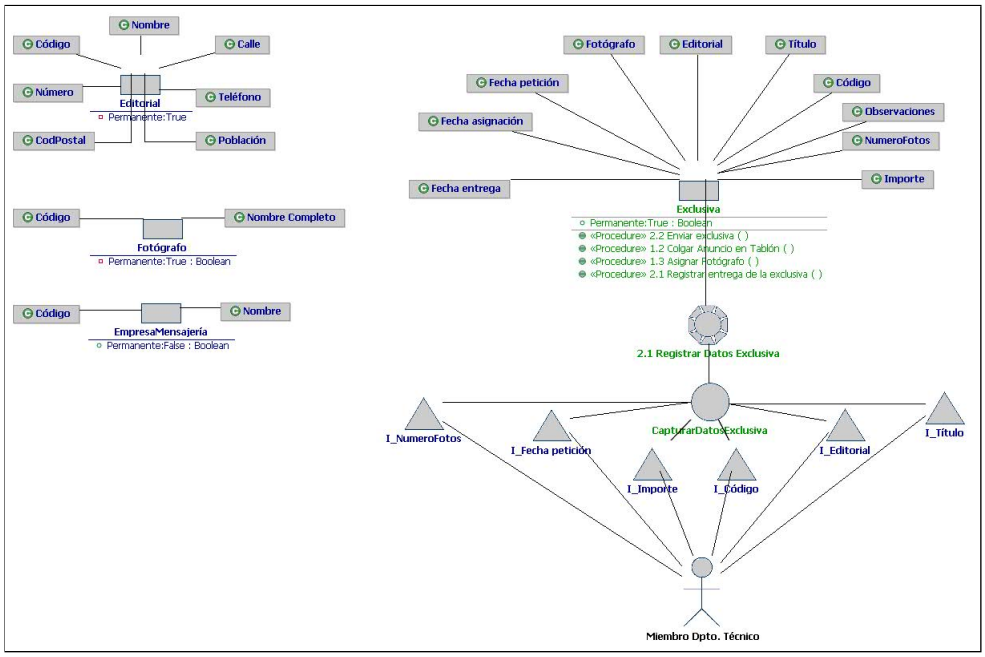

Figura 5.5: Diagrama de Dominio

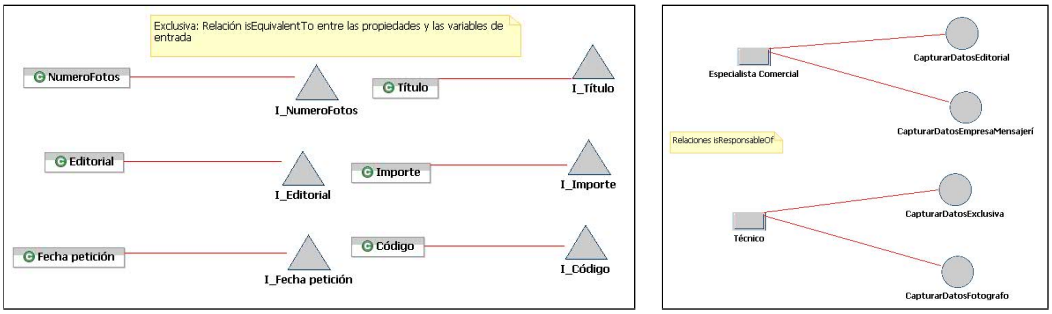

Figura 5.6: Representación de las relaciones *isEquivalentTo* y *isResponsabileOf*

5.2.4. PASO 4: Diagramas de Procesos

El siguiente paso es implementar los procedimientos empresariales mediante los diagramas de procesos y servicios y establecer las relaciones *implementedBy*. En este caso, únicamente se han utilizado los diagramas de procesos. Las formas ovaladas representan lo que en la Propuesta MDK se denomina roles de procesos y las rectangulares las actividades. A continuación, se explica el funcionamiento de cada uno de los procesos.

Procedimiento 1.2: Colgar Anuncio en el Tablón

En la figura 5.7 se muestra el proceso de colgar el anuncio de que hay una exclusiva disponible y en la figura 5.8 las relaciones *instantiatedBy* del proceso. Este proceso está formado por dos actividades: Realizar una copia de la exclusiva y colgarla en el tablón, teniendo como entrada una flujo de tipo población de elemento Exclusiva, como mecanismo a un miembro del departamento técnico y el estereotipo *Other* que indica el cambio de estado del tablón de anuncios e instancia a un elemento de tipo Exclusiva.

Tabla 5.1: Relación de propiedades y características de los elemento del dominio

Exclusiva					
Propiedad	Defining	Ini. Val.	Const.	Mult.	Tipo
Fotógrafo	NO	None	YES	1	Población Elemento Fotógrafo
Editorial	YES	None	YES	1	Población Elemento Editorial
Código	YES	None	YES	1	Texto
Título	YES	None	YES	1	Texto
Importe	YES	None	NO	1	Moneda
Observaciones	NO	None	NO	1	Texto
Fecha petición	YES	fecha actual	YES	1	Fecha
Fecha asignación	NO	None	YES	1	Fecha
Fecha entrega	NO	None	YES	1	Fecha
Número fotos	YES	None	NO	1	Número
Editorial					
Propiedad	Defining	Ini. Val.	Const.	Mult.	Tipo
Código	YES	None	YES	1	Texto
Nombre	YES	None	YES	1	Texto
Calle	YES	None	NO	1	Texto
Número	YES	None	NO	1	Número
Cod. Postal	YES	None	NO	1	Texto
Población	YES	None	NO	1	Texto
Teléfono	YES	None	NO	1	Número
Fotógrafo					
Propiedad	Defining	Ini. Val.	Const.	Mult.	Tipo
Código	YES	None	YES	1	Texto
Nombre Completo	YES	None	YES	1	Texto
Empresa de Mensajería					
Propiedad	Defining	Ini. Val.	Const.	Mult.	Tipo
Código	YES	None	YES	1	Cadena
Nombre	YES	None	YES	1	Texto

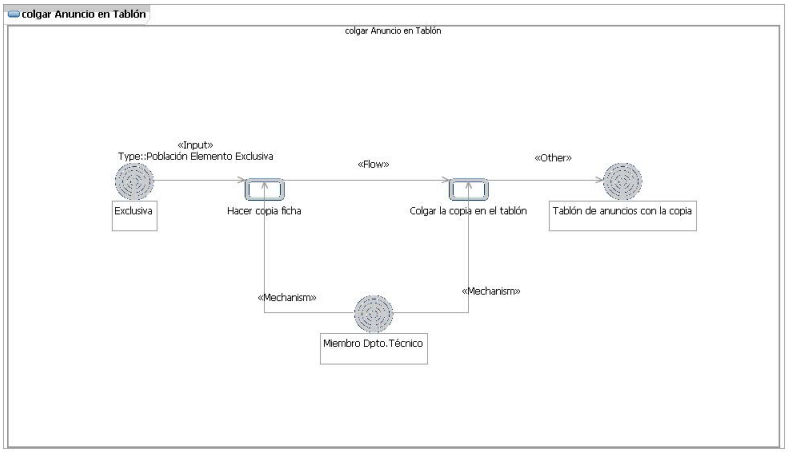

Figura 5.7: Proceso Colgar Anuncio en el Tablón

Procedimiento 1.3: Asignar fotógrafo

En la figura 5.9 se muestra el proceso de asignar un fotógrafo como responsable de realizar una exclusiva determinada y en la figura 5.10 las relaciones *instantiatedBy* del mismo. Este proceso está formado

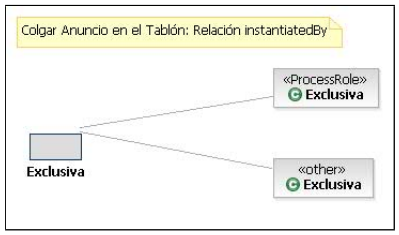

Figura 5.8: Instanciación de elementos del proceso 'Colgar Anuncio en el Tablón'

únicamente por la actividad de registrar el fotógrafo responsable y la fecha de petición de una exclusiva. Por lo tanto, las entradas del proceso serían un dato de tipo fecha y un dato de tipo población de elemento Fotógrafo. Los cambios de estado, estereotipados con *other*, se instancian con las propiedades Fecha de Asignación y Fotógrafo del elemento Exclusiva. Cabe destacar que los flujos *other* tienen como destino las propiedades del elemento Exclusiva, pero se ha decidido indicarlo gráficamente de forma más simple, indicando únicamente el elemento del dominio para facilitar la legibilidad del modelo. En la figura 5.10 se indican las relaciones de tipo *instantiatedBy* para este caso.

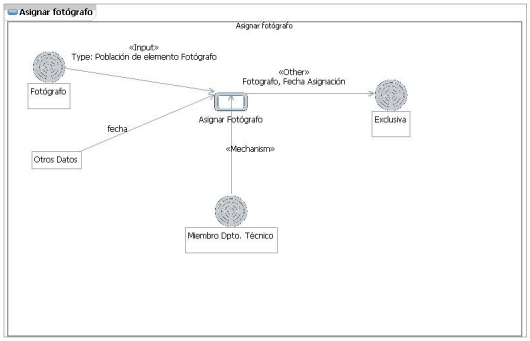

Figura 5.9: Proceso Asignar Fotógrafo

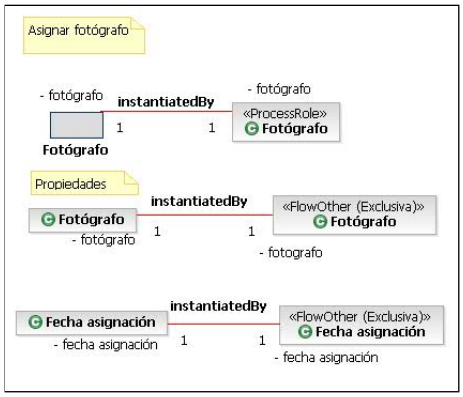

Figura 5.10: Instanciación de elementos del proceso 'Asignar Fotógrafo'

Procedimiento 2.1: Registrar entrega de exclusiva

En la figura 5.11 se muestra el proceso de registrar la entrega de una exclusiva. Este proceso se compone de dos actividades: completar la ficha de exclusiva con la fecha en la que se ha entregado y generar el cheque

para pagar al fotógrafo por el 40 % del total del importe de la exclusiva. En la figura 5.12 se muestran las relaciones de tipo *instantiatedBy* que intervienen en este proceso.

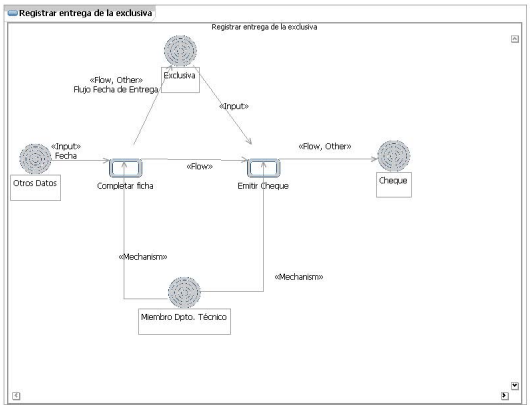

Figura 5.11: Proceso Registrar Entrega de Exclusiva

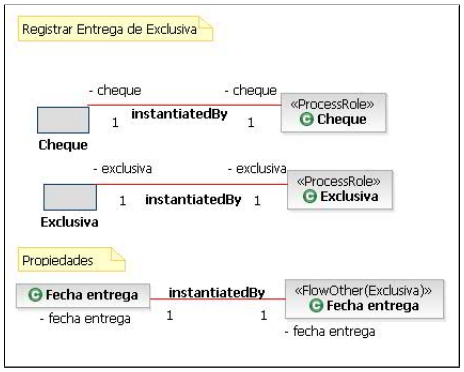

Figura 5.12: Instanciación de elementos del proceso 'Registrar Entrega de Exclusiva'

Procedimiento 2.2: Enviar exclusiva

En la figura 5.13 se muestra el proceso de enviar la exclusiva a la editorial que consta de las actividades: generar el albarán de envío y avisar a la empresa de mensajería para que se realice el envío. En la figura 5.14 se muestran las relaciones de tipo *instantiatedBy* que intervienen en este proceso.

Relación implementedBy

En la figura 5.15 se muestran las relaciones entre las tareas, procedimiento y procesos. En concreto se representan gráficamente las relaciones *ownedBy* entre los procedimientos y las tareas a las que pertenecen así como la relación *implementedBy* entre los procedimientos y los procesos. Además, indica que el procedimiento 'asignar Fotógrafo' tiene una regla de negocio asociada: 'RestriccionesFotógrafo'. Esta regla de negocio debe validar el fotógrafo que tiene como entrada el proceso correspondiente.

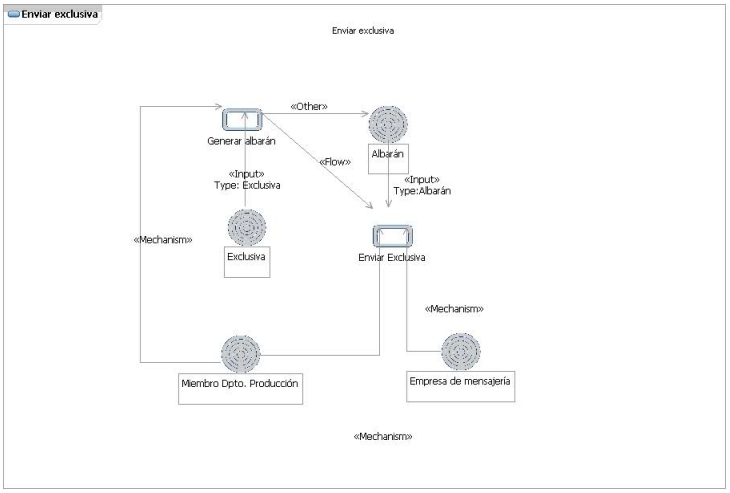

Figura 5.13: Recoger y Enviar Exclusiva

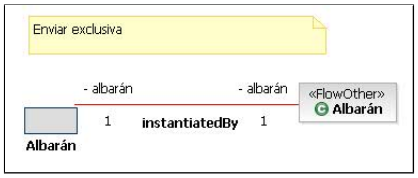

Figura 5.14: Instanciación de elementos del proceso 'Recoger y Enviar Exclusiva'

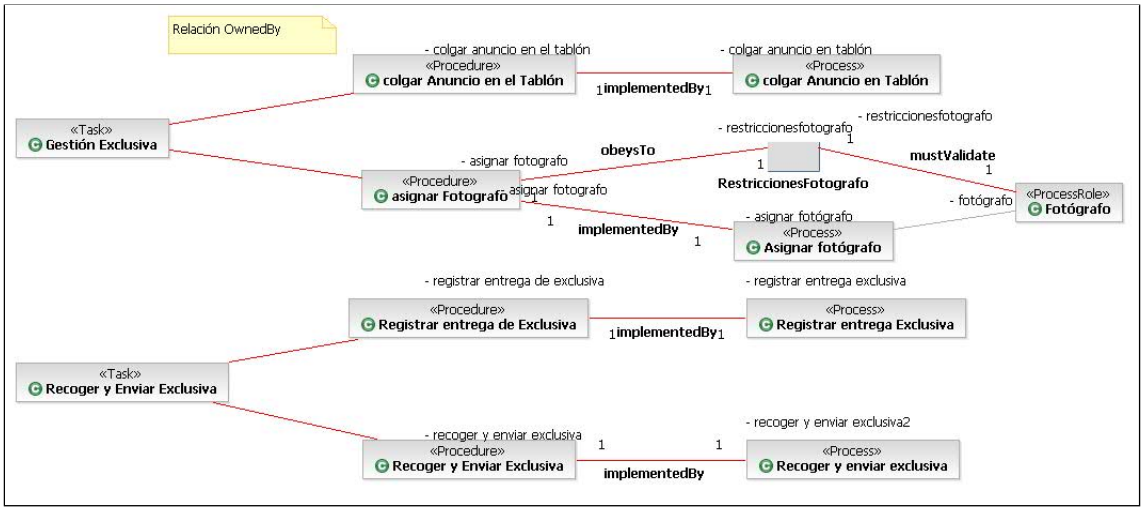

Figura 5.15: Relaciones *ownedBy* y *implementedBy*

5.2.5. PASO 5: Diagrama de Productos

En la figura 5.16 se muestra el diagrama de productos y las relaciones *representedBy* con los elementos del dominio. En este caso, el diagrama de productos únicamente incluye a la exclusiva. En cuadros amarillos se indica qué tipo de elementos del Metamodelo de Estructura se están utilizando. Así pues, se tiene el producto exclusiva cuyos materiales son las fotografías tomadas por el fotógrafo, que es el responsable de ese material y el proceso de producción es la toma de fotografías. Nótese que dado que en el caso de que el proceso

de producción lo llevara a cabo la empresa, el procedimiento y el correspondiente proceso implementado deberían formar parte del diagrama de dominio y de procesos respectivamente.

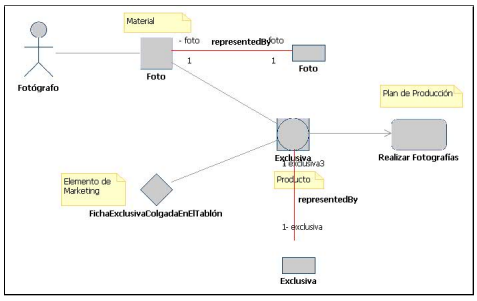

Figura 5.16: Diagrama de Productos

5.2.6. PASO 6: Completar el diagrama de dominio

Llegados a este punto, se debe revisar el Diagrama de Dominio para incorporar aquellos elementos que hayan aparecido al realizar los diagramas de procesos y producto. En este caso, en el proceso de registrar la entrega de una exclusiva nos aparece un nuevo elemento: el cheque que se le entrega al fotógrafo como pago por la realización de la exclusiva. Por otro lado, en el proceso de enviar la exclusiva aparece el elemento albarán y el diagrama de productos las fotografías individuales que componen la exclusiva. En la figura 5.17 se resaltan en naranja los elementos incorporados en esta fase. Como en el caso anterior, se han omitido los procedimientos de extracción y cálculo. En la tabla 5.2 se listan las características de las propiedades de los nuevos elementos incorporados en esta fase. Del elemento del dominio *Foto* no se incluyen propiedades dado que no se ha proporcionado información al respecto.

Tabla 5.2: FASE 6: Relación de propiedades y características de los elementos del dominio

Exclusiva					
Propiedad	Defining	Ini. Val.	Const.	Mult.	Tipo
Fotos	NO	None	NO	N	Población Elemento Foto
Cheque					
Propiedad	Defining	Ini. Val.	Const.	Mult.	Tipo
Fotógrafo	YES	None	YES	1	Población Elemento Fotógrafo
Exclusiva	YES	None	YES	1	Población Elemento Exclusiva
Albarán					
Propiedad	Defining	Ini. Val.	Const.	Mult.	Tipo
Editorial	YES	None	YES	1	Población Elemento Editorial
Exclusiva	YES	None	YES	1	Población Elemento Exclusiva

5.2.7. PASO 7: Abstracción en categorías ontológicas y bloques conceptuales

En la figura 5.18 se muestran las categorías ontológicas y los bloques conceptuales en los que se han agrupado los elementos del dominio detectados. La agrupación de los elementos en bloques se ha realizado en base al hecho de que pertenezcan a una misma dimensión de la empresa. Las categorías ontológicas permiten dividir los bloques conceptuales en categorías que identifican un grupo de elementos del dominio con características comunes.

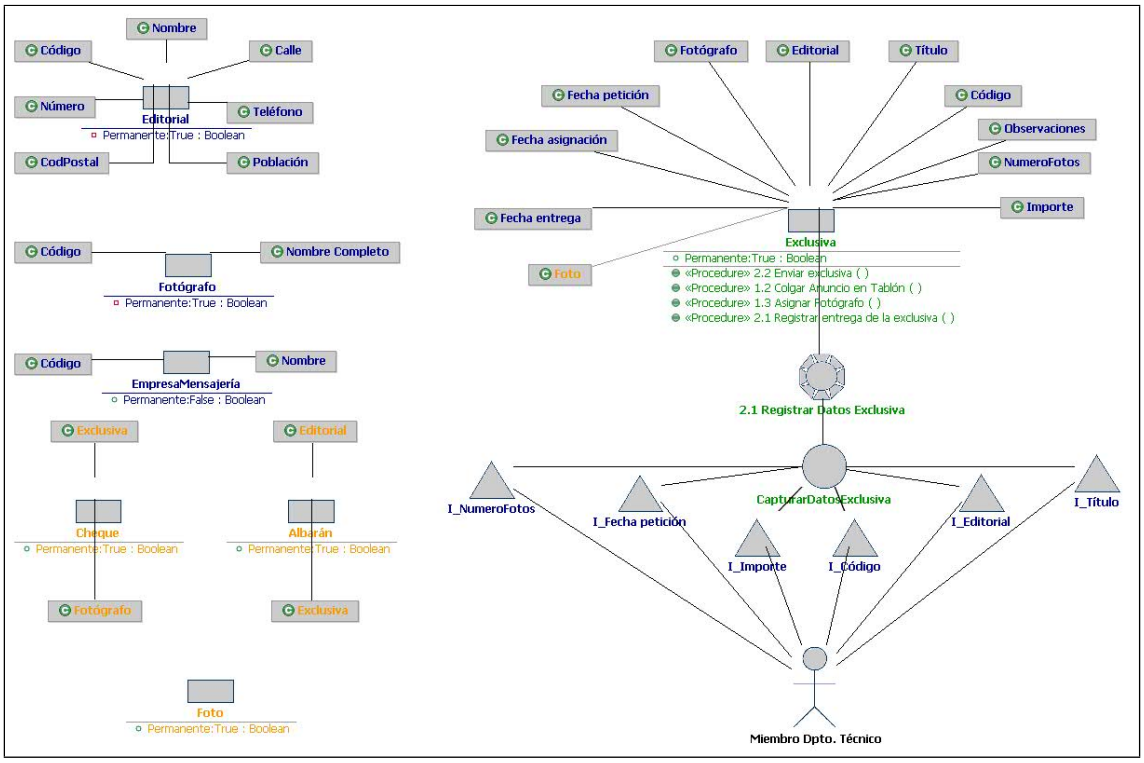

Figura 5.17: Diagrama de Dominio, FASE 6

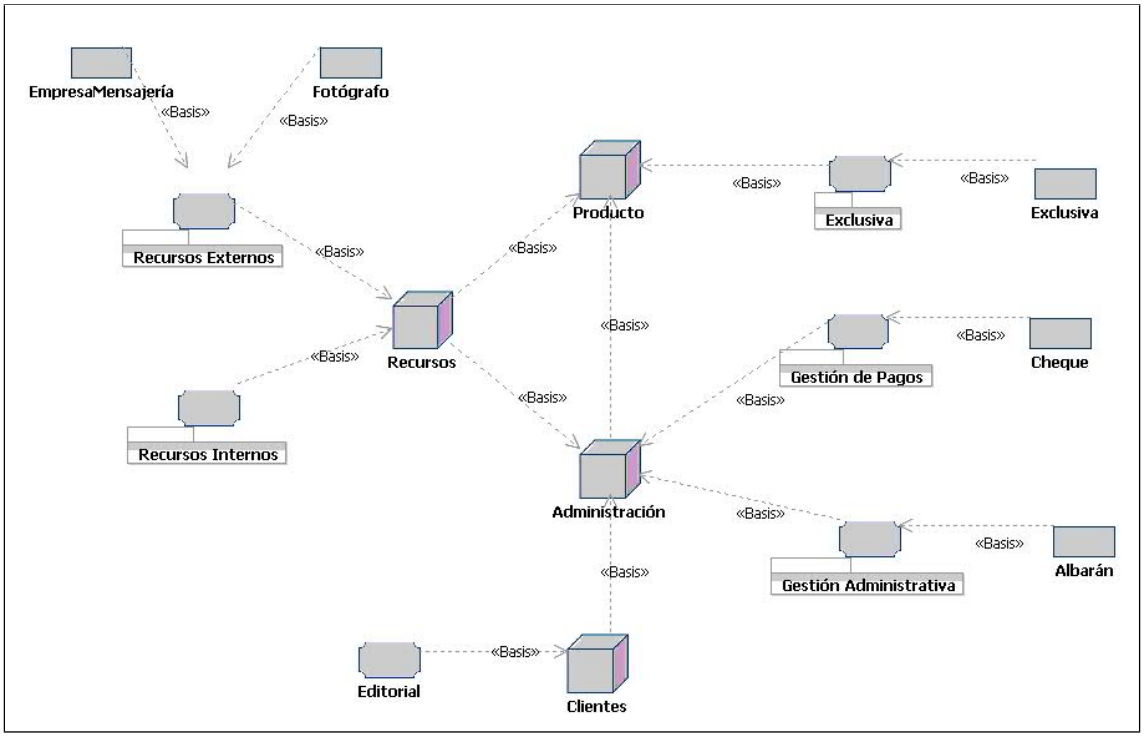

Figura 5.18: Diagrama de Bloques

5.2.8. PASO 8: Creación de diagramas de Análisis y Objetivos

En el desarrollo de este ejemplo no se han realizado estos diagramas ya que la especificación de requisitos no incluye información al respecto.

5.3. Extracción del modelo CIM a transformar

El modelo de la Agencia Fotográfica presentado a lo largo de este capítulo describe un modelo completo de todos los elementos del dominio empresarial que intervienen en la gestión de exclusivas. Sin embargo, al crear un sistema informático no siempre se representan en el mismo todos los objetos del mundo real que sí se representan en el modelo CIM. Por lo tanto, se necesita un mecanismo para discriminar los elementos del nivel CIM que forman el modelo PIM de aquellos que no.

Tal y como se ha explicado en la sección 3.1, para realizar la diferenciación de elementos se seleccionan los elementos del dominio, tareas y procedimientos que conformaran el modelo PIM y que además reflejan los requisitos del sistema informático en términos de sus requisitos de información y sus requisitos funcionales. En la figura 5.19 se muestra el *Knowledge Cube* en forma de grafo obtenido para el caso de estudio. Con el objetivo de simplificar la representación y que ésta sea legible, las propiedades se han agrupado como un solo nodo y se han omitido la mayoría de elementos de estructura organizativa así como los del diagrama de productos. En el *Knowledge Cube* se han resaltado en verde oscuro los elementos que se seleccionan manualmente. En verde claro y azul, se destacan los elementos seleccionados automáticamente a partir de la selección inicial. Finalmente, se han resaltado en naranja los elementos del diagrama de estructura organizativa que más adelante se asociarán a los agentes del sistema. Estos elementos se seleccionan automáticamente al ser responsables de los servicios básicos de los elementos del dominio seleccionados manualmente. Nótese que el especialista en producción, se selecciona por ser responsable de la tarea 'recoger y enviar exclusiva'. Los elementos seleccionados tanto automática como manualmente son aquellos que formarán parte del modelo PIM y por lo tanto los únicos sobre los que se aplicarán las reglas de transformación.

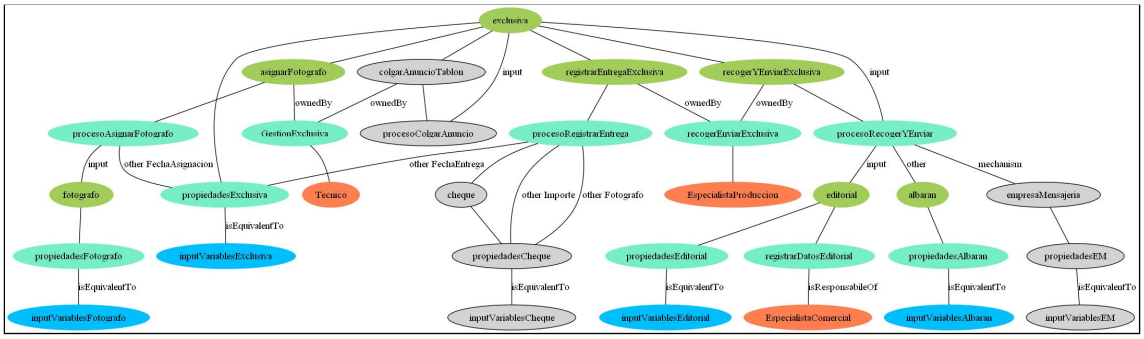

Figura 5.19: Representación en forma de grafo del *Knowledge Cube*

5.4. Mapping a OO-Method

En esta sección se explica el resultado de aplicar las reglas de transformación establecidas en el capítulo 4 sobre el modelo de la Agencia Fotográfica. Para cada una de las reglas establecidas se indica cuál sería el resultado obtenido al aplicarla sobre el caso de estudio. Además, se incluyen capturas de pantalla del modelo PIM resultante creado con la herramienta OLIVANOVA [16].

Regla 1: DomainElement → Class: Al aplicar la regla 1 sobre los elementos del dominio del fragmento del modelo CIM a transformar obtendríamos las clases: Exclusiva, Editorial, Fotógrafo y Albarán. En la figura 5.20 se muestra el Modelo de Objetos generado a partir de esta regla.

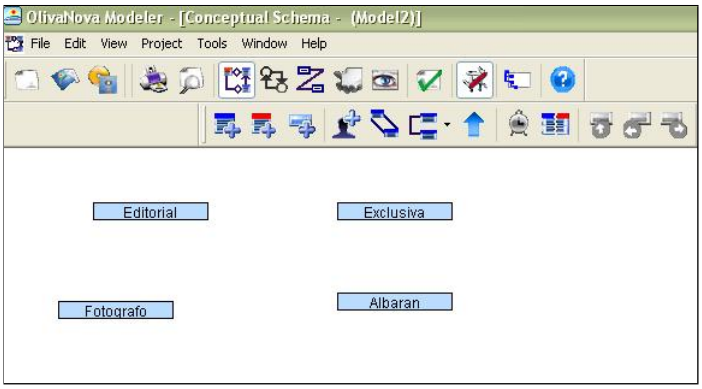

Figura 5.20: Regla 1: Estructura de clases

Regla 2: Property → Attribute, Regla 3 y Regla 4: Al aplicar la regla 2 sobre las propiedades de los elementos de dominio se obtiene el conjunto de atributos de las clases generados a partir de éstos. Para caracterizar los atributos se aplican las reglas desde la 2.1 a la 2.5 (tabla 4.1), la regla 3 y la regla 4. A modo de ejemplo, se muestra cómo se aplicarían las reglas para las propiedades Fotógrafo, Editorial y Título del elemento de dominio Exclusiva, análogamente se caracterizan el resto de atributos.

```
Propiedad Fotógrafo:
  Regla 2.1: Tiene relación de equivalencia con una variable de entrada: NO
    'request Upon Creation' = False
  Regla 2.2: MDK::Constant = YES --> OOM::Type = Constant
  Regla 2.3: MDK::Defining = False --> OOM::Nulls allowed = True
    // Se modifica el atributo OOM::type
             OOM::'request Upon Creation' = False
               --> OOM::Type = Variable
  Regla 4: MDK::Type = Población Elemento Fotógrafo -->
      OOM::Relación entre Fotógrafo y Exclusiva
      Cardinalidad Mínima = 0 (defining = NO)
      Cardinalidad Máxima = 1 (multiplicidad = 1)

Propiedad Editorial:
  Regla 2.1: Tiene relación de equivalencia con una variable de entrada: SÍ
    'request Upon Creation' = True
  Regla 2.2: MDK::Constant = YES --> OOM::Type = Constant
  Regla 2.3: MDK::Defining = YES --> OOM::Nulls allowed = False
  Regla 4: MDK::Type = Población Elemento Editorial -->
      OOM::Relación entre Editorial y Exclusiva
      Cardinalidad Mínima = 1 (defining = YES)
      Cardinalidad Máxima = 1 (multiplicidad = 1)

Propiedad Título:
  Regla 2.1: Tiene relación de equivalencia con una variable de entrada: SÍ
    'request Upon Creation' = True
  Regla 2.2: MDK::Constant = YES --> OOM::Type = Constant
  Regla 2.3: MDK::Defining = YES --> OOM::Nulls allowed = False
```

```
Regla 2.4: MDK::Type = Texto --> OOM::Type = String
```

El caso del tratamiento del valor inicial de la propiedad 'fecha petición', que indica la fecha en la que la editorial pidió la exclusiva a la agencia y por lo tanto coincide con la fecha en la que se registra la petición en la empresa, no se trata en este trabajo ya que esta característica se incluye con el fin de que el modelo CIM contenga la mayor información posible y es tarea del que completa el modelo PIM trasladarlo al lenguaje destino.

En la parte izquierda de la figura 5.21 se muestra la evolución del Modelo de Objetos, al transformar las propiedades se crean las relaciones entre las clases y los atributos de las mismas y en la parte derecha el resultado de aplicar la Regla 2 a las propiedades de la Exclusiva. Nótese que únicamente las propiedades con equivalencia a variables de entrada forman parte de la creación del objeto (Regla 2.1) y además, no se permiten valores para los atributos correspondientes a las propiedades definitorias (Regla 2.3).

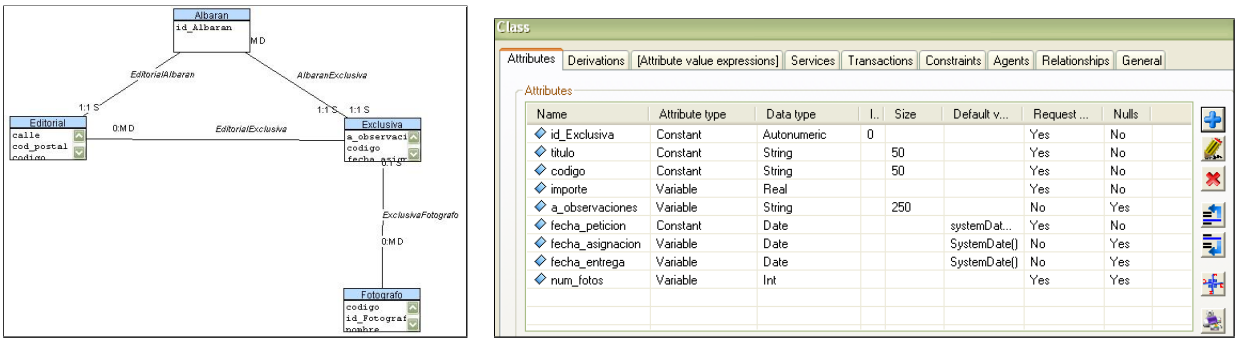

Figura 5.21: Regla 2, 3, 4: Estructura de atributos

Regla 5: Dado que el diagrama de dominio no incluye ninguna relación *isBasedOn*, esta regla no se ha aplicado.

Reglas 6, 7 y 8 : Estas reglas hacen referencia a la incorporación de los servicios de creación, edición y borrado respectivamente. Para todas las clases creadas se añade el servicio de creación. El servicio de edición existe si la clase tiene atributos de tipo variable. Finalmente el servicio de borrado existirá si el valor de permanencia del elemento del dominio correspondiente es falso. En el caso de estudio, ninguna clase tendría servicio de borrado ya que se guardan de forma permanente los datos relativos a las exclusivas, editoriales, fotógrafos y albaranes.

Respecto al servicio de edición, dado que la exclusiva y la editorial son los únicos elementos del dominio que tienen propiedades no constantes, las clases obtenidas a partir de éstos incluirán también servicio de edición.

En la figura 5.22 se muestran los servicios de creación y edición incorporados a la clase Exclusiva. Los argumentos del servicio de creación y edición se añaden automáticamente a partir de las características de los atributos 'Request Upon Creation' y 'add to Edit event'.

Regla 9: Creación de servicios: En este caso únicamente tenemos un servicio simple, el correspondiente al de asignar fotógrafo, ya que el resto de servicios forman parte de una transacción. Por lo tanto, las reglas aplicadas son las siguientes:

- Regla 9.1: Proceso Asignar Fotógrafo → Servicio simple asignarFotógrafo.

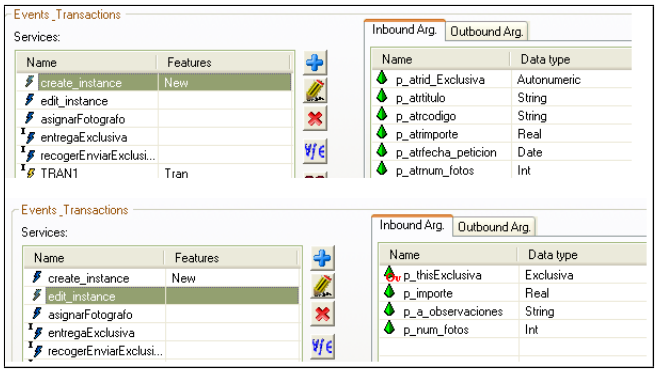

Figura 5.22: Regla 6 y 7: Servicios de creación y edición

- Regla 9.3: Del proceso Enviar Exclusiva: se crea una transacción para invocar el servicio de creación del albarán. El nombre de esta transacción sería 'trans1'. En este caso como no hay cambio de estado para la exclusiva, no se crea ningún servicio de cambio de estado.
- Regla 9.4: De la relación *triggers* entre Registrar Entrega Exclusiva y Enviar Exclusiva: Transacción que incluya al servicio creado para 'Registrar Entrega Exclusiva' y a 'trans1'. El nombre de la transacción sería: 'registrarEntregaExclusiva'. Además se marcan como internos el servicio de 'Registrar Entrega Exclusiva' y 'trans1'.

Reglas 10 y 11: Argumentos de entrada y cambios de estado Los argumentos de entrada de los servicios y los cambios de estado que realizan se obtienen a partir de los diagramas de procesos, los flujos de tipo *input* y *other* y las relaciones *instantiatedBy*. A continuación, se detallan los argumentos de entrada y los cambios de estado que realizan los servicios servicios.

- Servicio asignarFotógrafo:
  - Argumentos de Entrada: Elementos del Dominio Fotógrafo y una fecha
  - Cambio de estado: Exclusiva.fotógrafo y Exclusiva.fechaAsignacion
- transacción registrarEntregaExclusiva:
  - servicio registrarEntregaExclusiva:
    - Argumentos de Entrada: Fecha
    - Cambio de estado: Exclusiva.fechaEntrega
  - Invocación de trans1.
- transacción trans1:
  - Argumentos de Entrada: Exclusiva
  - Invocación del servicio de creación del Albarán.

Regla 12: Precondiciones Las precondicones que debe cumplir el sistema informático se deducen de los diagramas de reglas de negocio y se asocian a los servicios si a nivel CIM el procedimiento origen del servicio y la regla de negocio están relacionados mediante *ObeysTo*. En este caso existe esta relación entre el procedimiento asignar fotógrafo y la regla 3.1, por lo tanto se añade una precondición al servicio asignarFotógrafo. Dado que no se ha establecido un asociaciones entre las reglas de negocio y las fórmulas de las restricciones en OO-Method, esta regla no modifica el modelo PIM. Sin embargo, es útil para que,

en el caso de que se automatizara el proceso, obtener un listado completo de las restricciones a asociar a cada servicio, así como los elementos que utilizan (relación *mustValidate* y *isNeededFor*) para completar el modelo PIM.

Regla 13: Agentes: Las relaciones inter-metamodelo permiten asociar los agentes con los procedimientos de los que son responsables. En la figura 5.19 se han resaltado en naranja aquellos elementos cuya función es que sean agentes del sistema. En OO-Method un agente se representa como una clase estereotipada. Así pues de la fuente tácita mostrada en la figura 5.5 se crea un agente en el sistema cuyo nombre viene determinado por el nombre del perfil de trabajo, rol o empleado con el que está relacionado mediante *isResponsabileOf*. Así pues se crea el agente Técnico con visibilidad sobre los procesos de creación y edición de las exclusivas y fotógrafos, el agente Comercial con visibilidad sobre los procesos de creación y edición de editoriales.

Regla 14: Agentes a partir de la inter-relación ownedBy: Los agentes del resto de servicios se deducen a partir de la relación *ownedBy* entre las tareas y procedimientos que dan origen a los mismos.

Así pues, para el servicio asignarFotógrafo, cuyo origen es el procedimiento 1.3 Asignar Fotógrafo que pertenece a la tarea Gestión de Exclusivas realizada por el departamento Técnico, será el agente Técnico, creado anteriormente quién tenga visibilidad sobre este procedimiento.

Además, se crea un nuevo agente Producción con visibilidad para realizar la transacción entregaExclusiva, cuyos servicios derivan de la tarea 'Recoger y enviar Exclusiva' responsabilidad del departamento de producción.

Regla 15: Agente global del sistema: Una vez definidos todos los elementos del modelo de objetos se crea un agente global que tendrá visibilidad y permisos de ejecución sobre todos ellos.

Regla 16: Modelo Dinámico: El Modelo Dinámico se crea a partir de las relaciones *mustOccur* entre procedimientos siguiendo los pasos indicados en el capítulo 4.

En la figura 5.23 se muestra el diagrama de estados que se genera para la exclusiva. En el diagrama de estados participan los servicios básicos, el servicio propio asignar fotógrafo y la transacción EntregaExclusiva.

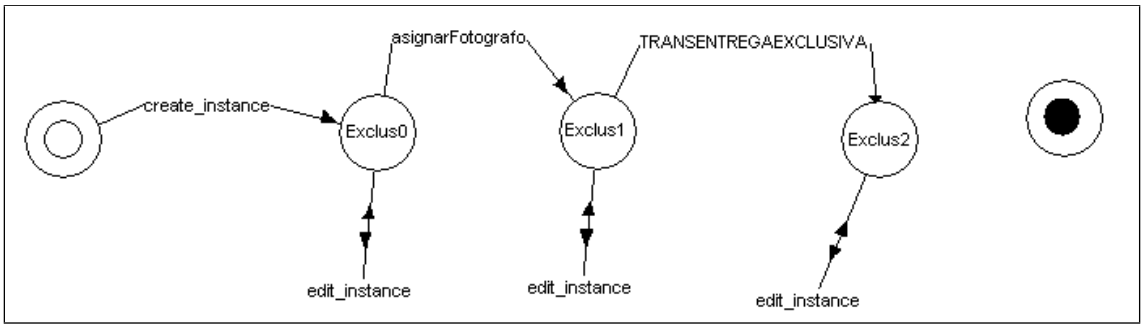

Figura 5.23: Modelo Dinámico

5.5. Conclusión

A lo largo de este capítulo se ha presentado un caso de estudio que pone en práctica tanto la propuesta de modelado como la ejecución de las reglas de transformación a OO-Method establecidas. El siguiente paso ha consistido en crear el modelo PIM resultado de la ejecución de estas reglas con la herramienta OLIVANOVA.

Las reglas establecidas permiten crear un modelo PIM con el enfoque OO-Method que incluye la estructura de clases y relaciones, la definición de atributos, de servicios, de agentes y la creación del Modelo Dinámico. Así pues, para completar el modelo PIM faltaría únicamente definir los alias de los nombres para que sean más intuitivos y la definición del Modelo Dinámico para los servicios. Por lo tanto, el tiempo de creación del modelo se reduciría considerablemente. Sin embargo, se debe tener en cuenta que existen algunos problemas que no resuelven las reglas establecidas.

El primer problema detectado, es que la regla de *mapping* 2.4 para transformar las propiedades de tipo Texto en un atributo de tipo *string* no se indica la longitud concreta de las cadenas y por lo tanto, se debería indicar a posteriori.

Otro problema detectado son los conflictos con las palabras reservadas de OO-Method. En concreto, al incluir el atributo 'observaciones' para la clase 'Exclusiva' la herramienta muestra un aviso de que no se puede utilizar esa palabra como nombre de atributo ya que coincide con una palabra reservada. Por lo tanto, si este proceso se automatizara se deberían tener en cuenta que nombres no se pueden utilizar para denominar a las clases, atributos, servicios, etcétera. Sin embargo, este hecho debería ser transparente al modelador de forma que el proceso de transformación se encargaría de renombrar de forma sistemática aquellas palabras que puedan crear conflicto.

Capítulo 6

Conclusión y Trabajo Futuro

A lo largo de este documento se ha explicado el funcionamiento de un lenguaje de modelado empresarial de nivel CIM, resultado de adaptar el conjunto de metamodelos de la Propuesta MDK, y se han definido un conjunto de reglas de para transformar el modelo de nivel CIM en un modelo PIM basado en el enfoque OO-Method. En este capítulo se detallan las conclusiones extraídas de la realización de este trabajo, los límites del mismo y sus posibles extensiones.

6.1. Resultados del trabajo

Las reglas de transformación definidas están orientadas a reutilizar al máximo el conocimiento contenido en un modelo empresarial para definir un modelo conceptual que le dé soporte. Así pues, del modelo CIM se extrae el siguiente conocimiento a nivel PIM:

- Estructuras de datos del modelo informático: en OO-Method esto son las clases del sistema, sus atributos y relaciones.

- Servicios para incorporar datos en el sistema, modificarlos y eliminarlos de las estructuras de datos: en OO-Method, se corresponde con los servicios de básicos de creación, edición y destrucción de una instancia de una clase.

- Servicios para modificar el estado del sistema: esta operación a nivel CIM se realiza mediante la implementación de los procedimientos como procesos y servicios que se aplican sobre elementos del dominio (a nivel CIM, estos servicios hacen referencia al conjunto de actividades diseñados para satisfacer las necesidades de un conjunto de clientes, no implican cambios de estado como a nivel PIM), a nivel PIM los cambios de estado son operaciones que se aplican sobre las instancias de las estructuras de datos. En concreto, en OO-Method se implementan como servicios asociados a una determinada clase o servicios globales. En este caso, el conocimiento que se reutiliza del nivel CIM es la signatura de los procesos y servicios. Por lo tanto, a cada clase se le incorporan los servicios correspondientes incluyendo los parámetros de entrada y los cambios de estado a realizar.

- Restricciones: Las reglas de negocio de una empresa determinan las condiciones que se deben cumplir para un correcto funcionamiento de la misma. Si una restricción afecta a un procedimiento que se ejecuta en el sistema informático, debe formar parte del nivel PIM. En este caso, la definición de reglas de negocio definidas con la Propuesta MDK es una definición poco precisa y por lo tanto el conocimiento que se puede reutilizar es el nombre de la restricción y sus parámetros de entrada.

Así pues se ha creado una especificación para el modelado empresarial que da soporte a OO-Method, y completa así el proceso de modelado de tres niveles definidos en el estándar MDA. Sin embargo, el modelo PIM generado no contendría la implementación funcional de los servicios y transacciones. Pero, al reutilizar la información del nivel CIM, el punto de partida del modelo PIM contiene toda la información necesaria para generar un primer prototipo no funcional que enseñar a la persona que encarga una solución informática. En el caso de que el cliente piense que faltan funcionalidades, estas se seleccionan del modelo CIM y se volvería a crear el prototipo. Una vez el cliente piense que el prototipo es completo, se incorporaría la funcionalidad a cada servicio y se finalizaría el proceso de desarrollo de software. Así pues los resultados de la puesta en práctica de las propuestas realizadas en este trabajo son las siguientes:

- Un modelo empresarial que puede abarcar todas las dimensiones de la empresa, realizado a partir de una guía (véase 3.3) que facilita la participación de los empresarios en la tarea de modelado.
- Un mecanismo para seleccionar las partes del modelo CIM que se desean informatizar, reflejando así los requisitos funcionales que el usuario final desea que contenga su sistema informático.
- Generación automática de la mayor parte del modelo PIM.

Sin embargo, el modelo PIM resultante está limitado al sistema transaccional de la empresa ya que no se han incluido reglas que permitan trasladar los diagramas de Análisis y Objetivos a nivel PIM. Las causas que han llevado a excluir estos elementos del conjunto de reglas de transformación es el hecho de que los sistemas de soporte a la toma de decisiones y de análisis de conjuntos de datos tienen características diferentes a los sistemas transaccionales.

Los sistemas transaccionales están orientados a dar soporte a los procesos de negocio que se llevan a cabo con una periodicidad temporal baja (todos los días, todas las semanas, etcétera), se ejecutan sobre los datos del sistema y el resultado de la ejecución de los mismos es un cambio de estado del sistema. Sin embargo, los procedimientos de análisis se suelen ejecutar en espacios de tiempo más separados y no cambian el estado del mismo sino que lo reflejan mediante informes realizados con técnicas de procesamiento de datos para obtener información útil a los directivos de la empresa para tomar decisiones. Por lo tanto, antes de definir este tipo de reglas se deben estudiar el funcionamiento de los sistemas de ayuda a toma de decisiones, tarea que forma parte de una posible extensión de este trabajo.

6.2. Aplicación de los conocimientos adquiridos en el Máster ISMFSI

El trabajo realizado está muy vinculado a la temática del Programa de Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información. De la asignatura Técnicas Avanzadas de Depuración se han aplicado las técnicas de fragmentación de programas al ámbito de los modelos para obtener el método de discriminación de elementos del modelo empresarial. Se han aplicado los conocimientos sobre transformación de modelos adquiridos en la asignatura Gestión de Modelos, las bases de la elicitación de requisitos aprendidas en la asignatura Ingeniería de Requisitos, el proceso de desarrollo de software basado en modelos estudiado en Introducción a MDA y los fundamentos de modelado a nivel PIM, el manejo de la herramienta OLIVANOVA y el desarrollo del caso de estudio aprendidos en la asignatura Tecnología Software de Ambientes Web.

6.3. Trabajo Futuro

Utilizar los modelos empresariales únicamente para generar un modelo conceptual implica perder todos los beneficios que supone contar con un modelo de este tipo, además si ese es el único objetivo será mejor realizar directamente el modelo PIM. A continuación, se describen posibles extensiones del trabajo realizado para aprovechar el conocimiento contenido en los modelos empresariales.

6.3.1. Extracción automática de requisitos del sistema informático

Definir los requisitos y el alcance de una nueva aplicación informática es una tarea fundamental, ya que a partir de los mismos se comienza a crear la aplicación, sea mediante un modelo de nivel PIM o escribiendo el código directamente. El trabajo realizado permite marcar los requisitos de la aplicación sobre el modelo CIM, seleccionando los elementos del dominio y los procedimientos a implementar. Respecto a esto, se podría implementar un sistema de representación automática de los requisitos de información y funcionales con el formato y plantillas propuestas en [27]. El objetivo es proporcionar una descripción escrita que contenga la información precisa de los elementos del sistema. La forma de realizar esta tarea sería crear un nuevo conjunto de reglas de *mapping* hacia los elementos que se describen en [27]. Estas reglas asociarían los elementos del dominio y sus propiedades a los requisitos de información y los procedimientos a los requisitos funcionales.

6.3.2. Generación de una ontología

Las ontologías de interoperabilidad son un recurso clave para hacer que las empresas interoperen ya que permiten establecer el vocabulario mediante el cual se van a comunicar las empresas. Como extensión de este trabajo se propone la creación de una ontología empresarial a partir de los elementos del dominio y las relaciones entre ellos.

De los diagramas de bloques y dominio se obtendrían los conceptos que forman la empresa. Estos elementos estarían relacionados como sigue: *DomainBlock isComposedBy Domain has DomainElement*. Por otro lado, las relaciones inter-metamodelo establecerían los axiomas que relacionan los elementos.

6.3.3. El modelo como mecanismo de comunicación

Otro posible trabajo futuro sobre las propuestas realizadas es incorporar un lenguaje de consulta al modelo de forma que se pueda extraer un subconjunto del mismo que entregar a los clientes, proveedores y nuevos trabajadores con el objetivo de que entiendan el funcionamiento de la empresa y que la colaboración con la empresa se base en la forma en la que ésta funciona.

6.3.4. Análisis de cargas de trabajo

Otra posible extensión podría ser incorporar a las tareas y procedimientos elementos que describan la periodicidad con la que se ejecutan así como el tiempo estimado que conlleva cada ejecución. Dado que la Propuesta MDK incluye la representación del organigrama de la empresa cuyos roles, perfiles de trabajo y empleados están relacionados con los procedimientos que ejecutan permitiría analizar cuánto tiempo necesita un empleado o un perfil para realizar todas las tareas que tiene asignadas y así distribuir mejor las cargas de trabajo.

6.3.5. Incorporación de la propuesta de modelado y las reglas de transformación a OLIVANOVA

Finalmente, se podrían incorporar las propuestas realizadas a la herramienta OLIVANOVA para completar el ciclo de generación de software dirigida por modelos utilizando todos los niveles de MDA. Antes de realizar está incorporación se deberían crear nuevos casos prácticos a partir de casos reales en lo que trabaje la empresa para verificar y completar tanto la estrategia de modelado como las reglas de transformación.

Bibliografía

[1] V. Pazos and R. Grangel. Entregable 2.2: Marco para la interoperabilidad empresarial a través de interval. Technical report, Interval, 2008.

[2] Huy N. Pham, Qusay H. Mahmoud, Alexander Ferworn, and Alireza Sadeghian. Applying model-driven development to pervasive system engineering. In *SEPCASE '07: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*, page 7, Washington, DC, USA, 2007. IEEE Computer Society.

[3] IDEF. Integrated DEFinition methods. http://www.idef.com/, 2007.

[4] J. C. Hernandez-Matias, A. Vizan, A. Hidalgo, and J. Rios. Evaluation of techniques for manufacturing process analysis. *Journal of Intelligent Manufacturing*, 17(5):571–583, OCT 2006.

[5] KS Chin, X Zu, CK Mok, and HY Tam. Integrated Integration Definition Language 0 (IDEF) and coloured Petri nets (CPN) modelling and simulation tool: a study on mould-making processes. *International Journal Of Production Research*, 44(16):3179–3205, AUG 15 2006.

[6] JM Cai, TM Chen, and LY Zhu. A structure modeling method for multi-task embedded software design. In Wu, ZH and Chen, C and Guo, MY and Bu, JJ, editor, *Embedded Software and Systems*, volume 3605 of *Lecture Notes in Computer Science*, pages 576–581, 2005. 1st International Conference on Embedded Software and Systems, Hangzhou, PEOPLES R CHINA, DEC 09-10, 2004.

[7] ESPIRIT Consortium AMICE. *CIMOSA: Open System Architecture for CIM*. Springer-Verlag, 1993.

[8] Markus Strohmaier. *Business process oriented knowledge management*. Springer-Verlag, 2003.

[9] METIS, 2007. http://www.computas.com.

[10] G Doumeingts and Y Ducq. Enterprise modelling techniques to improve efficiency of enterprises. *Production Planning & Control*, 12(2):146–163, MAR 2001.

[11] Michel Roboam, Marc Zanettin, and Lucas Pun. GRAI-IDEF0-Merise (GIM): Integrated methodology to analyse and design manufacturing systems. *Computer Integrated Manufacturing Systems*, 2(2):82 – 98, 1989.

[12] G Berio and FB Vernadet. New developments in enterprise modelling using CIMOSA. *Computers in Industry*, 40(2-3):99–114, NOV 1999.

[13] Jason J. Jung. Semantic business process integration based on ontology alignment. *Expert Systems with Applications*, 36(8):11013–11020, OCT 2009.

[14] Reyes Grangel. *Propuesta para el Modelado del Conocimiento Empresarial*. PhD thesis, Universitat Jaume I, 2007. PhD supervisor: Ricardo Chalmeta.

[15] Óscar Pastor and Juan Carlos Molina. *Model Driven Architecture in Practice*. Springer, 2007.

[16] Care Technologies. Olivanova. http://www.care-t.com/index.asp.

[17] A. Boronat, JA. Carsi, and I. Ramos. Automatic support for traceability in a generic model management framework. In Hartman, A and Kreische, D, editor, *Model Driven Architecture Foundations and Applications, Proceedings*, volume 3748 of *Lecture Notes in Computer Science*, pages 316–330, 2005. 1st European Conference on Model Driven Architecture - Foundations and Applications, Nuremberg, GERMANY, NOV 07-10, 2005.

[18] Object Management Group. MDA guide version 1.0.1, 2003.

[19] UML (Object Management Group), 2000. http://www.uml.org/.

[20] T Weis, A Ulbrich, and K Geihs. Model metamorphosis. *IEEE Software*, 20(5):46+, SEP-OCT 2003.

[21] G Graw and P Herrmann. Generation and enactment of controllers for business architectures using MDA. In Oquendo, F and Warboys, B and Morrison, R, editor, *Software Architecture*, volume 3047 of *Lecture Notes in Computer Science*, pages 148–166, 2004. 1st European Workshop on Software Architecture, St Andrews, England, MAY 21-22, 2004.

[22] W Zhang, H Mei, HY Zhao, and J Yang. Transformation from CIM to PIM: A feature-oriented component-based approach. In Briand, L and Williams, C, editor, *Model Driven Engineering Languages and Systems, Proceedings*, volume 3713 of *Lecture Notes in Computer Science*, pages 248–263, 2005. 8th International Conference on Model Driven Engineering Languages and Systems, Montego Bay, JAMAICA, OCT 02-07, 2005.

[23] Ricardo Chalmeta and Reyes Grangel. Methodology for the implementation of knowledge management systems. *Journal of the American Society for Information Science and Technology*, 59(5):742–755, MAR 2008.

[24] O. Pastor, J. Gomez, E. Insfran, and V. Pelechano. The OO-Method approach for information systems modeling: from object-oriented conceptual modeling to automated programming. *Information Systems*, 26(7):507–534, NOV 2001.

[25] OCL (Object Management Group), 2000. http://www.omg.org/spec/OCL/2.0/.

[26] John Hatcliff, Matthew B. Dwyer, and Hongjun Zheng. Slicing software for model construction. *Higher Order Symbol. Comput.*, 13(4), 2000.

[27] Amador Durán Toro and Beatriz Bernárdez Jiménez. *Metodología para el Análisis de Requisitos de Sistemas Software*. Universidad de Sevilla, 2001.