MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE Y SISTEMAS INFORMÁTICOS

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ITINERARIO EN INGENIERÍA DE SOFTWARE – 31105128

EVOLUCIÓN DEL DESARROLLO DE PRODUCTOS SOFTWARE BASADOS EN PROCESOS ANALÍTICOS AVANZADOS MEDIANTE SOLUCIONES BIG DATA A TRAVÉS DE LA APLICACIÓN DE MODELOS ESPECÍFICOS DEL DOMINIO

AUTOR **HÉCTOR GONZÁLEZ VELASCO**

DIRECTOR **RUBÉN HERADIO GIL**

CURSO ACADÉMICO 2015/2016

CONVOCATORIA SEPTIEMBRE

MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN INGENIERÍA DE SOFTWARE Y SISTEMAS INFORMÁTICOS

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

TRABAJO FIN DE MÁSTER

ITINERARIO: INGENIERÍA DE SOFTWARE

CÓDIGO: 31105128

TÍTULO: EVOLUCIÓN DEL DESARROLLO DE PRODUCTOS SOFTWARE BASADOS EN PROCESOS ANALÍTICOS AVANZADOS MEDIANTE SOLUCIONES BIG DATA A TRAVÉS DE LA APLICACIÓN DE MODELOS ESPECÍFICOS DEL DOMINIO

AUTOR: HÉCTOR GONZÁLEZ VELASCO

DIRECTOR: RUBÉN HERADIO GIL

CURSO: 2015/2016

CONVOCATORIA: SEPTIEMBRE

Hoja de calificaciones

Impreso TFdM05_Autor

Autorización de publicación y difusión del TFdM para fines académicos

Autorización

Autorizo a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del Autor

Resumen

El trabajo fin de máster se centra en investigar la forma de aplicar diferentes conceptos avanzados de la ingeniería de dominio y el desarrollo dirigido por modelos a un tipo concreto de productos software basados en la construcción de procesos de minería de datos y análisis de grandes volúmenes de información, con el objetivo de proponer un planteamiento que permita facilitar y mejorar la forma en la que se desarrollan actualmente este tipo de soluciones.

Debido a sus propias características variables y al acoplamiento con los propios datos a analizar, el desarrollo de estas soluciones se realiza habitualmente desde cero y de forma particular, adaptando el software a cada caso concreto, sin realizar una reutilización sistemática de los componentes disponibles o desarrollados con anterioridad dentro del mismo contexto.

La solución propuesta se basa en la utilización de los propios conceptos del dominio a analizar, expresados a través de una serie de ontologías, para dirigir la especificación de características de estos procesos analíticos y por tanto, del producto software a construir. Además de esto, para facilitar el desarrollo se ha diseñado y construido un lenguaje específico del dominio que permite expresar dichas características de una manera simple y directa.

El resultado constituye una herramienta funcional y suficientemente genérica que permite adaptar fácilmente estas técnicas a la construcción de diferentes sistemas. Integrando de forma sencilla y efectiva el conocimiento disponible del dominio, con la información a analizar y las técnicas o herramientas analíticas avanzadas para su procesamiento y comprensión de cara a la toma de decisiones.

Palabras clave

Minería de datos, Analítica Avanzada, Extracción de Conocimiento, Aprendizaje Automático, Ciencia de datos, Análisis de grandes volúmenes de datos, Ingeniería de Software, Metodologías de Desarrollo de Software, Ingeniería dirigida por modelos, Modelado específico del dominio, Diseño dirigido por el dominio, Minería de datos dirigida por el dominio, Lenguajes específicos del dominio, Desarrollo de Software Generativo, Reutilización de Código.

Abstract

The Master's degree final project is focused on researching how to apply different advanced concepts on domain engineering and model-driven development to a particular type of software products based on the construction of data mining processes and large volume data analysis, with the aim of proposing an approach that can facilitate and improve the way that solutions are currently developed.

Due to their particular and variable features and the coupling with the information to be analyzed, the development of these solutions is usually done from scratch, adapting the software ad hoc to each unique case, without a systematic reuse of available or previously developed components in the same context.

The proposed solution is based on the extended use of the domain concepts, manifested through a collection of ontologies, to drive the abstract feature model definition of the analytical processes, and therefore, the planned software product. In addition, a domain specific language that can be used to express these characteristics in a simple and direct way, has been designed and built in order to make the development easier.

The result is a full functional and generic tool which supports the adaptation of these techniques to a particular system development. This solution successfully integrates the available domain knowledge, with the information to analyze and the advanced analytical techniques or big data tools for processing it and actionable insight discovering.

Keywords

Data Mining, Advanced Analytics, Knowledge Discovery, Machine Learning, Data Science, Big Data, Software Engineering, Software Development Methodologies, Model Driven Engineering, Domain Specific Modeling, Domain Driven Design, Domain Driven Data Mining, Domain Specific Languages, Generative Software Development, Code Reuse.

Índice

Lista de figuras

Lista de tablas

1. Introducción

1.1. Ámbito

El análisis de la información disponible, con el objetivo principal de extraer el conocimiento relevante y dirigir la toma de decisiones dentro de una organización, no es para nada una necesidad nueva. Existen desde hace años una multitud de estrategias, técnicas y soluciones que se han desarrollado con el propósito de agilizar y mejorar este tipo de tareas.

Debido al aumento exponencial de los datos digitales recopilados por las empresas, que provienen de diversas fuentes interconectadas, sistemas móviles, servicios en tiempo real, medios y redes sociales, sensores IoT (*Internet of Things*), etc. Los esfuerzos durante la última década se han centrado además en proporcionar plataformas que permitan almacenar y procesar este volumen masivo de información en un tiempo razonable. Este complejo y heterogéneo entorno en el que se encuentra el análisis de información es ya conocido mediante el popular término "Big Data".

Estas plataformas y herramientas disponibles constituyen una base perfecta sobre la que los ingenieros y científicos de datos pueden construir sus procesos software analíticos. Sin embargo, como ha venido ocurriendo a lo largo de la historia, la mejora y evolución en el desarrollo de cualquier tipo de software se basa en diferentes principios, entre los que merece la pena destacar el aumento en el nivel de abstracción, la modularización y la reutilización de código. Para conseguir incrementar sus capacidades y reducir el coste de desarrollo, el futuro de los productos software basados en procesos analíticos y soluciones Big Data debe pasar también por aplicar estas características.

Para intentar ilustrar el ámbito en el que se desarrolla este trabajo se va a describir una posible situación real dentro del entorno empresarial actual. Nos centraremos en una empresa dedicada a proporcionar servicios analíticos a otras organizaciones, por ejemplo, en el sector de las telecomunicaciones. Imaginemos que esta empresa ha cerrado un acuerdo de colaboración muy importante con un cliente, para realizar un producto software encaminado a descubrir y predecir los patrones de comportamiento de los usuarios a través de los datos que la empresa de telecomunicaciones ha ido almacenando durante el último año. Esta información representa un volumen muy considerable.

Tras una fase inicial de toma de requisitos, comprensión del contexto y examen parcial de los problemas a resolver, comienza el desarrollo de la solución, que va a integrar diferentes procesos de extracción y transformación de información, procesamiento y realización de modelos matemáticos. Todo ello basado en herramientas y marcos de desarrollo Big Data.

A medida que avanza el análisis de la información disponible y con ello el desarrollo del producto, aparecen diferentes complejidades, debidas entre otras causas a la diversidad de los datos y a que los requisitos planteados son demasiado específicos. Además, como los resultados intermedios no son los deseados inicialmente, se deciden modificar el diseño de las

funcionalidades y la forma de abordar algunos planteamientos para cumplir con los tiempos de desarrollo pactados.

Todas estas decisiones hacen que se desarrolle una gran parte de la solución a medida para cumplir con las necesidades concretas del cliente, centrándose en la parte analítica funcional y olvidando en cierta medida algunos principios de la ingeniería de software. La solución técnica desarrollada es muy completa, pero está muy ligada a la información disponible, acoplada a las técnicas y herramientas utilizadas, y funcionalmente orientada a los pasos que se deben seguir para resolver cada proceso concreto. A pesar de las dificultades, el proyecto ha obtenido un buen resultado, por lo que el diseño y las decisiones aplicadas parece que han sido correctas.

Un tiempo después surgen nuevas posibilidades de realizar diversas propuestas para proyectos muy similares pero en otros clientes del mismo sector. Como es lógico, la empresa decide utilizar su producto software y su experiencia previa para abordar estas oportunidades.

Tras analizar su solución en base a los nuevos requisitos, la empresa que ha desarrollado el producto software se da cuenta de que, a pesar de manejar los mismos conceptos (usuarios, productos, tarifas, cobros, campañas, etc.) y de que el conocimiento adquirido durante el desarrollo es de gran utilidad, sólo un porcentaje muy bajo del código fuente y de los componentes disponibles se pueden reaprovechar para el nuevo proyecto. Por tanto, el presupuesto de desarrollo necesario para las nuevas funcionalidades vuelve a ser elevado. Además de esto, el coste se eleva en el mantenimiento de aquellas partes a reutilizar, para intentar asegurar la compatibilidad con el sistema que ya está en producción.

De cara a preparar las presentaciones de su propuesta para los nuevos clientes, se utiliza un nuevo equipo de personas que debe revisar el planteamiento realizado anteriormente. A pesar de que la documentación disponible es bastante completa, la descripción está muy centrada en las funcionalidades y procesos generales a realizar, mientras que el código se centra en la parte operacional (parámetros, algoritmos, resultados). Por tanto, se dan cuenta de que es complejo enlazar las diferentes partes del producto para entender los procesos analíticos desde un punto de vista de negocio, de cara a poder explicar a los nuevos clientes en sus propios términos, cuáles son las estrategias y planteamientos que se van a aplicar para resolver sus problemas.

A pesar de las dificultades inherentes a este tipo de soluciones software, el planteamiento propuesto durante la realización de este trabajo trata de mejorar este tipo de situaciones. Intentando acercar la definición de estos procesos analíticos de minería de datos a la forma en la que se desarrollan otros tipos de productos software.

Mediante la aplicación de técnicas metodológicas relativas a la ingeniería y diseño dirigido por modelos cercanos al dominio, la solución alcanzada pretende proporcionar una nueva forma de desarrollo, que permita obtener un mayor nivel de abstracción y trazabilidad, separación efectiva entre conceptos y responsabilidades, así como una mayor modularización. Haciendo que los productos software desarrollados sean más mantenibles y se pueda incrementar el nivel de reutilización de sus componentes.

1.2. Necesidades

Para profundizar en las necesidades que han suscitado el desarrollo de este trabajo se intentan recoger a continuación de forma resumida los principales problemas y dificultades que vienen derivadas de la propia naturaleza de los sistemas basados en procesos de análisis y minería de datos, así como de la forma en la que estos se suelen desarrollar.

En general este tipo de aplicaciones contienen una serie de diferencias sustanciales respecto a otros tipos de productos software. Es por esto, que su metodología y forma de desarrollo difieren también en gran medida de los estándares actuales en ingeniería de software, dificultando su industrialización y por consiguiente la posible reducción de costes que facilitaría una mayor evolución y agilidad en los desarrollos.

En primer lugar, los requisitos iniciales para este tipo de aplicaciones y productos son a menudo muy difusos, no es posible su especificación directa o directamente no existen. En realidad, uno de los primeros pasos dentro de un proceso encaminado a la extracción de conocimiento a partir de información, es entender correctamente la naturaleza del problema planteado. Esto lleva, por tanto, a un desconocimiento parcial de las funcionalidades que el software debe incluir o que es necesario desarrollar inicialmente. Por otra parte, a medida que se van esclareciendo este tipo de necesidades de negocio, se incrementa la dependencia entre las funcionalidades o componentes a implementar y los propios datos a analizar.

Aunque muchas fases de los procesos suelen ser comunes y a menudo se repiten entre aplicaciones con objetivos similares, las características de los datos a analizar (origen, significado, tipología, representación, estructura, formato, periodicidad, nivel de agregación, etc.) hacen que los requisitos sean variables o necesiten una serie de desarrollos previos antes de decidir su inclusión o no en el producto final a implantar.

Todo esto puede llegar incluso a cambiar completamente la definición de los requisitos. Por ejemplo, si se demuestra que con la información disponible no va a ser posible llegar a ninguna conclusión precisa, ni ratificar o refutar las cuestiones que han suscitado el análisis. En estos casos extremos se podría llegar incluso a redefinir el problema, sus premisas, o la forma de abordar el análisis. Impactando totalmente en las funcionalidades necesarias.

Como se puede apreciar, la variabilidad produce uno de los principales problemas en este tipo de elementos software, la escasa reutilización de código. Tratando de buscar mayor agilidad y velocidad en el desarrollo, se tiende en muchas ocasiones a implementar procesos a medida para cada nueva situación o escenario planteado. Sin detenerse a reflexionar si se pueden adaptar o reutilizar alguno de los componentes incluidos en otra aplicación realizada previamente.

Además de esto, la propia metodología seguida en los procesos de minería de datos lleva a la necesidad de realizar un proceso puramente iterativo con diferentes fases: generación, comprensión, análisis, predicción, validación, etc. Esto tiene como resultado la construcción de diferentes versiones del software, incluyendo y excluyendo las características desarrolladas.

Esto hace muy necesario un férreo control de los componentes y módulos que se van desarrollando en el marco del producto software completo, para evitar posibles fallos, descoordinación, repetición de parte del análisis, generación de duplicidades en los datos, etc.

La dependencia entre los datos y en muchas ocasiones, entre los diferentes resultados intermedios que se van obteniendo, implica también una dependencia entre las funcionalidades o características de las aplicaciones de este tipo. Complicando además la forma de priorizar y estructurar las tareas a realizar por parte del equipo de desarrollo.

Al realizarse habitualmente el proceso de análisis de una forma iterativa, otra necesidad habitual en estos casos es la posibilidad de repetir de forma exacta los pasos seguidos durante una parte del análisis o la totalidad de un estudio intermedio de la información. Es decir, tener una trazabilidad de la ejecución, de los componentes y funcionalidades implicadas, que permita determinar con total exactitud cómo se han generado ciertos resultados intermedios o se ha llegado, por ejemplo, a una conclusión concreta.

La trazabilidad es sumamente necesaria en este tipo de procesos, para la validación de los resultados obtenidos. Pero también a la hora de realizar un traspaso o compartición de conocimiento entre los diferentes actores implicados en la elaboración del software. Un caso concreto que se produce habitualmente es la explicación a un cliente o usuario de negocio sobre los pasos seguidos en las diferentes fases del proceso, junto con el estado actual y su futura evolución. En este punto, es sumamente crítico haber establecido y engranado correctamente los conceptos del propio dominio y términos cercanos al mismo, dentro del proceso de desarrollo, para permitir obtener una comunicación o comprensión lo más fluida posible. Esta necesidad también se produce entre los propios miembros del equipo de desarrollo, a modo de documentación o lenguaje interno. Cuando es necesario evolucionar o mejorar una parte del análisis, revisar la forma en que está implementado un componente, o si por ejemplo, se asigna una nueva persona a una tarea relacionada con el desarrollo o pruebas del producto.

Otro momento de carencia, suele ocurrir cuando se debe generar una aplicación final a partir de los componentes del producto software realizados, como por ejemplo, si se requiere de una puesta en producción para realizar periódicamente la ingesta de datos, parte del análisis, la predicción de nuevos valores o la visualización de los resultados. En estos casos, es posible que haya sido necesario implementar diferentes versiones del proceso y se hayan realizado múltiples intentos o estudios intermedios distintos dentro del desarrollo. Los cuales, en realidad, podemos asimilar a la implementación y activación de diferentes características o funcionalidades dentro del software.

En estos casos, disponer de una forma sencilla y rápida de automatizar la generación de diferentes ejemplares del producto es otra característica que resulta muy apreciada. Del mismo modo, debido a los ajustados tiempos de mercado, se impone también la necesidad de agilizar el proceso de incorporación de funcionalidades para producir una línea de productos software completa que pueda encajar en casos de negocio similares.

Desde una visión más técnica, es una práctica cada vez más habitual la necesidad de seleccionar e integrar en el desarrollo diferentes tecnologías, lenguajes de programación y plataformas para llevar a cabo el proceso de análisis e interpretación de la información de la mejor forma posible. Esto hace nuevamente que se incremente la complejidad de estas soluciones y pone aún más de manifiesto la necesidad de una arquitectura de componentes consistente, que permita incorporar componentes individuales con un origen o implementación diferente.

Por último, se incluye a modo de resumen una tabla con las principales necesidades detectadas que motivan la realización del trabajo. La tabla 1 se complementa con una columna que muestra el posible nivel de impacto en cada necesidad (en una escala incremental 1 a 3) propiciado por la evolución o mejora de los mecanismos de desarrollo de este tipo de soluciones.

Necesidad	Impacto
Soporte para especificación de requisitos del proceso analítico	1
Habilitar mecanismos para variabilidad de los requisitos	1
Desacoplar funcionalidades y los datos a analizar	1
Desligar los requisitos de la información específica	1
Reformulación de requisitos en base a resultados intermedios	2
Realización de un proceso de desarrollo iterativo	3
Control de los componentes intermedios desarrollados	3
Trazabilidad de los resultados intermedios y diferentes versiones	3
Enlazar conceptos de dominio con desarrollo de software	3
Facilitar la comunicación de resultados y un lenguaje ubicuo	2
Proporcionar mecanismos de documentación del proceso	2
Mejorar y automatizar la forma de puesta en producción	2
Facilitar la integración con diferentes tecnologías y plataformas	3

Tabla 1: Resumen de necesidades en el desarrollo de productos analíticos

1.3. Conceptos

De cara a mejorar la comprensión del contexto en el que se enmarca este trabajo antes de formular su planteamiento y alcance, se considera necesario introducir de una forma general una serie de conceptos relacionados con la temática del mismo y con los principios en los que se basa el desarrollo de la solución alcanzada.

Por una parte, debido al propio ámbito relacionado con los productos software considerados, es preciso conocer en cierta medida el entorno y algunas de las principales características específicas de los procesos de análisis y minería de datos a través de soluciones Big Data. Por otra parte, además de estos conceptos ligados a la parte más analítica, durante el desarrollo del trabajo y a lo largo de la memoria se han utilizado otro conjunto de fundamentos y

planteamientos más cercanos a la ingeniería del software, basada en el análisis del dominio y dirigida por modelos específicos del dominio.

Algunos de los términos manejados poseen una definición muy amplia y suelen ser empleados en muchas ocasiones para hacer referencia a un conjunto más amplio de conceptos interrelacionados. Por esto, se intentará precisar su utilización dentro de este ámbito. En la medida de lo posible, se intentará además proporcionar ejemplos ilustrativos e introducir brevemente el estado y uso actual de los mismos.

1.3.1. Ciencia de Datos

El término Ciencia de Datos, del inglés "*Data Science*", hace referencia de forma general a un campo de especialización que engloba diferentes disciplinas relacionadas con los sistemas y procesos para la extracción de conocimiento y conclusiones a partir de los datos disponibles. Como se puede ver en la figura 1, la ciencia de datos es de alguna forma una prolongación de otras disciplinas o campos más específicos, entre los que se encuentran las matemáticas, estadística, informática, computación, analítica predictiva, modelos de probabilidad, programación, bases de datos, minería de datos, aprendizaje automático, inteligencia artificial, etc.

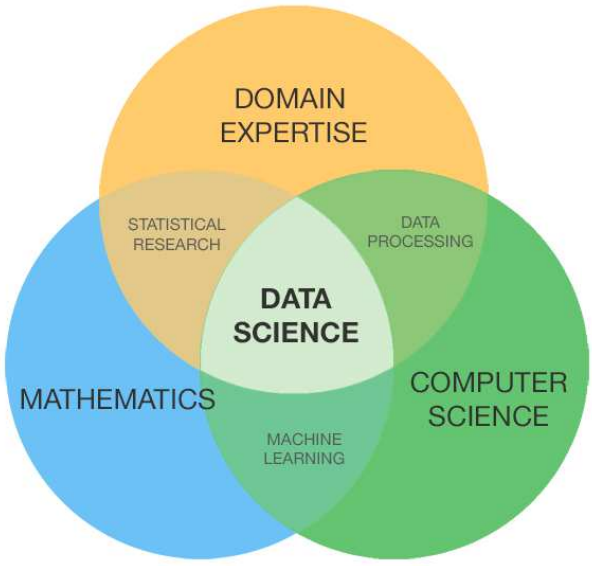

Figura 1: Disciplinas incluidas en la Ciencia de Datos

Este campo entraña una gran complejidad, al necesitar la combinación de una serie de técnicas avanzadas y requerir una gran cantidad de habilidades. En realidad, no se trata únicamente de habilidades técnicas: validar y analizar los datos, utilizar algoritmos concretos y programar los procesos o soluciones; es necesario además comprender el dominio de aplicación y conseguir comunicar de forma efectiva los resultados obtenidos.

La ciencia de datos ha adquirido además una enorme importancia en la actualidad. La extracción de conclusiones que puedan ser accionables a partir del análisis de la información disponible es una necesidad que está siendo muy demandada en todos los ámbitos. Sobre todo dentro de las empresas, que necesitan conocer y predecir los gustos, opiniones y comportamiento de sus clientes, para ofrecer mejores productos o servicios y adelantarse a la competencia.

La realización de proyectos y el desarrollo de productos específicos en este campo suele realizarse siguiendo un flujo de trabajo o metodología bastante particular. Aunque en muchos campos puede ser ligeramente diferente, los pasos más habituales son:

1. **Comprensión**, de la información disponible, dominio de actuación y objetivos.
2. **Preparación**, de los datos que se van a utilizar en el análisis.
3. **Modelado**, creación de modelos estadísticos y algoritmos de analítica avanzada.
4. **Evaluación**, de los resultados obtenidos y extracción de conclusiones.
5. **Implantación**, puesta en producción del sistema.
6. **Monitorización**, control y seguimiento de los resultados a lo largo del tiempo.

Figura 2: Etapas del flujo de trabajo habitual en proyectos de Ciencia de Datos

Como se puede apreciar en la figura 2, este flujo de trabajo es normalmente cíclico. Esto es debido fundamentalmente a dos motivos: por una parte, los propios resultados y acciones llevadas a cabo en base a éstos pueden volver a alimentar la entrada de datos. Y por otra parte, se necesita un control para asegurar que los efectos conseguidos siguen siendo significativos.

Por poner un ejemplo concreto, si el objetivo del proceso es la realización de un modelo de propensión de compra, que mida la probabilidad que tienen los clientes de adquirir un producto en un determinado momento, entonces el resultado obtenido (es decir, si finalmente se produce o no la compra) se podrá utilizar, pasado un tiempo, para re-entrenar el modelo si se detecta que éste ha perdido su poder predictivo.

1.3.2. Análisis de grandes volúmenes de datos

El procesamiento analítico de grandes volúmenes de información se suele definir en la actualidad a través de los términos "*Big Data*" o "*Big Data Analytics*". Estos hacen referencia a los casos en los que el volumen de datos es tan alto o éstos son tan complejos que no se pueden aplicar herramientas o técnicas tradicionales de procesamiento y son necesarias nuevas soluciones.

Como se puede ver en la figura 3, en los últimos años se ha producido un incremento exponencial de la cantidad de información almacenada digitalmente. Esto es debido, entre otras causas, al continuo crecimiento y abaratamiento de los dispositivos interconectados que recogen los datos: móviles, sensores, cámaras, micrófonos, etc. Y a la utilización cada vez más masiva de los canales de comunicación, redes sociales, servicios en la nube, transacciones a través de internet, etc. Esta tendencia es cada vez mayor, por lo que las necesidades de análisis de información y el mercado también se incrementan.

Figura 3: Incremento exponencial en la predicción de datos almacenados

Los sistemas y procesos de este tipo se suelen describir mediante una serie de características denominadas popularmente como las "uves" del Big Data. Estas características son conocidas así por asociarse a un término que comienza por la letra V. Inicialmente se consideraron únicamente tres características principales, aunque posteriormente se han extendido a cinco y más recientemente hasta siete. A continuación se hace un resumen de todas ellas:

1. **Volumen**. Hace referencia a la magnitud o tamaño de información generada y procesada, aunque estos valores pueden ser relativos.
2. **Velocidad**. Hace referencia a la gran velocidad de generación de los datos y a las necesidades de procesamiento demandadas en el menor tiempo posible.
3. **Variedad**. Hace referencia a la naturaleza y a los distintos tipos de datos a analizar, tanto estructurados como no estructurados.

4. **Variabilidad**. Hace referencia al continuo cambio en el significado de los datos, lo que puede producir que éstos no sean siempre consistentes.
5. **Veracidad**. Hace referencia a la necesidad de obtener unos resultados precisos y rigurosos, aunque la información de entrada no tenga la calidad esperada.
6. **Visualización**. Hace referencia a la necesidad de presentar y visualizar los resultados de forma accesible, de forma éstos que sean entendibles.
7. **Valor**. Hace referencia al hecho de que la información por sí misma carece de valor, éste reside en el análisis efectivo y la extracción de conclusiones.

Desde un punto de vista más técnico, el problema principal que se necesita abordar en este tipo de planteamientos es ver cómo se debe almacenar toda la información y buscar los mecanismos para que ésta se pueda procesar en el menor tiempo posible. Esto pasa necesariamente por hacer uso de paradigmas de computación paralela y distribuida. De manera que los cálculos necesarios para realizar una tarea se puedan dividir y paralelizar entre diferentes unidades de computación. Para conseguir esto es imprescindible el uso de soluciones Big Data específicas que posibiliten el análisis de los datos.

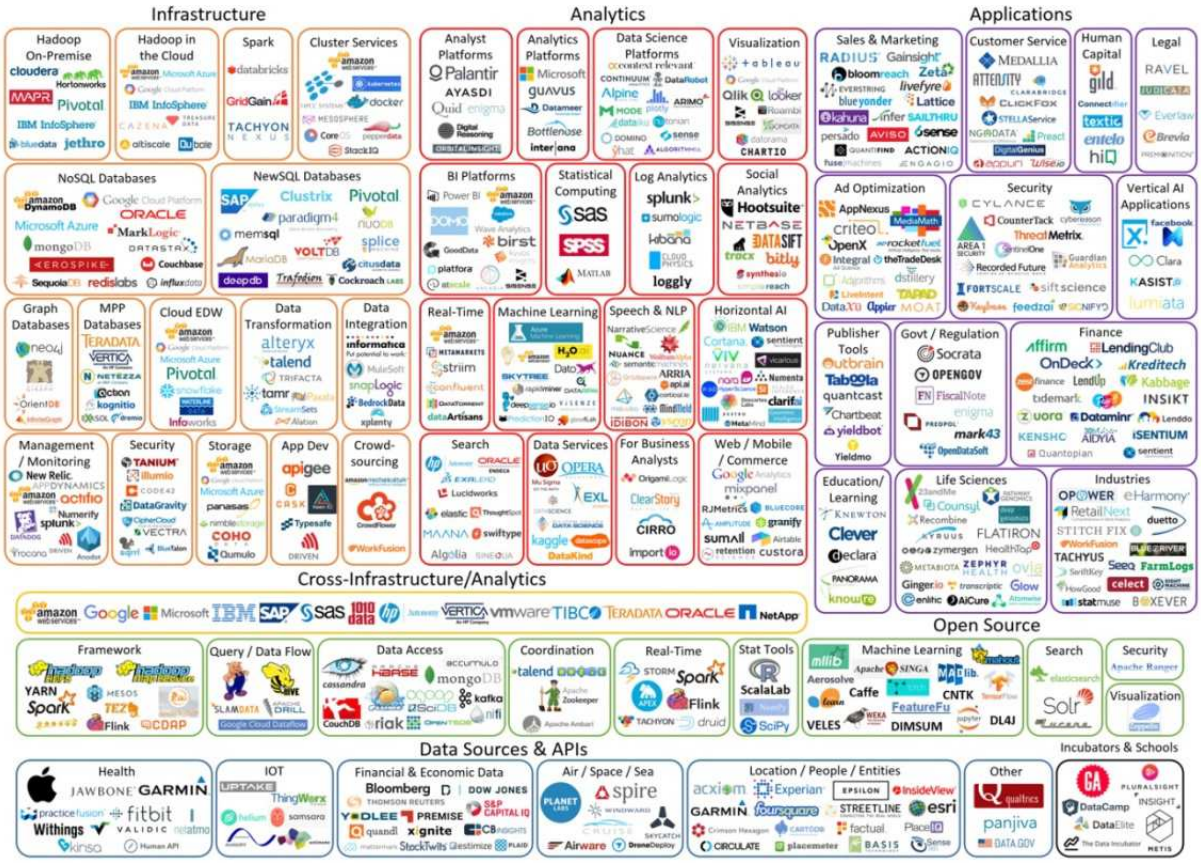

Figura 4: Panorama de soluciones basadas en tecnologías Big Data

Al igual que existen múltiples complejidades posibles implicadas en la elaboración de un proceso analítico de este tipo, también existen cientos de lenguajes, herramientas y plataformas que se han desarrollado para facilitar cada tarea. Tal y como se puede ver en el ejemplo de la figura 4.

Se vuelve por tanto también imprescindible el seleccionar en cada caso aquellas que mejor apliquen e integrar las diferentes soluciones para formar un único sistema o producto final.

Dentro de las herramientas analíticas más utilizadas en la actualidad se pueden destacar principalmente los lenguajes R y Python, junto con todas las librerías, algoritmos y utilidades implementadas bajo sus marcos de trabajo.

Por otra parte, dentro de las posibles plataformas, Apache Hadoop merece una mención especial. Se trata de una implementación de código abierto que incluye una serie de componentes y recursos base para el almacenamiento y procesamiento distribuido de información, sobre un cluster de máquinas económicas estándar. Inspirado inicialmente por dos artículos publicados por Google, se ha convertido en la actualidad en el estándar de facto en la industria.

Sobre Apache Hadoop se ha creado además todo un ecosistema de soluciones técnicas y analíticas más específicas. En este contexto, Apache Spark constituye un buen ejemplo de un motor para el procesamiento de datos a gran escala de carácter generalista.

1.3.3. Ingeniería de dominio

La ingeniería de dominio es una disciplina planteada originalmente con el objetivo de mejorar la calidad de los productos software a través de la reutilización de componentes. En la ingeniería de dominio se captura la información y se representa el conocimiento sobre un determinado dominio con el fin de crear artefactos reutilizables en el desarrollo de nuevos productos, posiblemente dentro de la misma familia.

Se ha demostrado que muchos de los nuevos productos desarrollados son en realidad variantes de otros sistemas dentro del mismo contexto. Con la ingeniería de dominio se utilizan los conceptos y las implementaciones ya realizadas, lo que permite mejorar la productividad, reducir los costes y el tiempo de mercado.

El término “componente software” es bastante amplio y se suele utilizar dentro de la industria de maneras muy diferentes. La definición más común suele referirse a “una pieza funcional de software que es liberada independientemente de otras y que proporciona acceso a sus servicios a través de interfaces” [1] [2].

A pesar de que suele ser el propósito más habitual, la ingeniería de dominio no sólo permite la reutilización de componentes software. Otra serie de artefactos pueden ser también objeto de dicha reutilización, como por ejemplo: las especificaciones de requisitos, los diseños y patrones, los procesos y algoritmos, las arquitecturas, los modelos de datos, etc.

Un proceso de ingeniería de dominio se estructura normalmente en tres fases principales:

1. **Análisis**: En esta fase se estudia el dominio, se definen sus conceptos y se produce un modelo de dominio que representa los aspectos comunes y variables de los sistemas dentro de este contexto.
2. **Diseño**: La fase de diseño se utiliza definir la arquitectura genérica que soportará la realización de sistemas dentro del dominio.
3. **Implementación**: En esta última fase se construyen los componentes, procesos y herramientas necesarios para la construcción de los productos software.

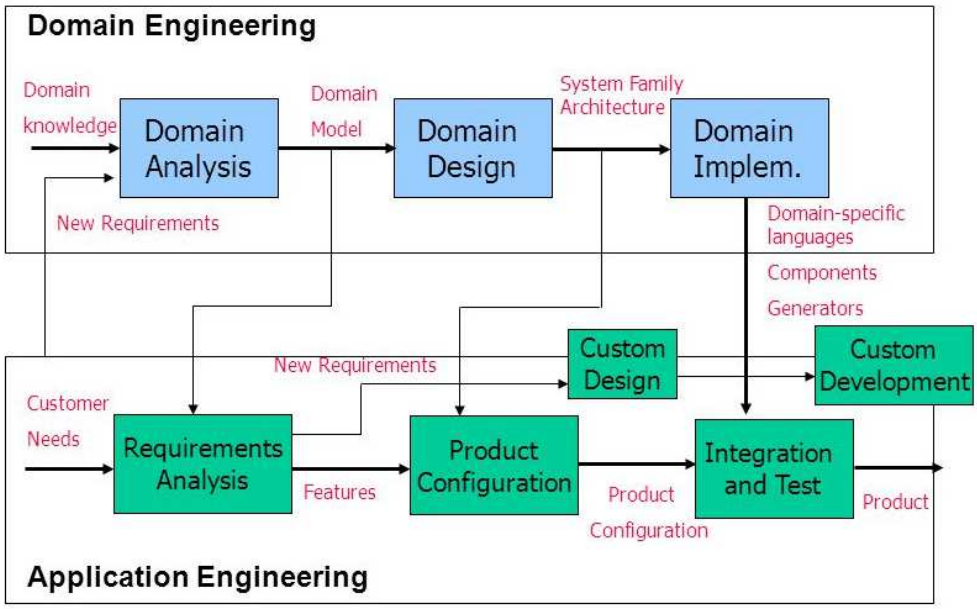

Figura 5: Fases de la ingeniería de dominio frente a ingeniería de aplicación

Desde una perspectiva conceptual, la definición de un modelo de dominio constituye un conjunto de abstracciones que describen todos los aspectos seleccionados dentro de una esfera de conocimiento, influencia o actividad [3] [4]. Éste modelo debe representar de manera precisa todos los conceptos y relaciones, e incorporar además el conocimiento y comportamiento específico. De una manera más coloquial, un modelo de dominio representa los conceptos del mundo real que se necesitan incorporar al software para poder resolver los problemas de negocio dentro del dominio.

Muy ligado a esta definición, se encuentra también el concepto de modelo de dominio utilizado para hacer referencia a los modelos de información específicos, que representan de forma estructurada los datos disponibles dentro de un contexto particular. A lo largo del trabajo, se considera habitualmente que el modelo de información será una particularización del modelo de dominio general que representa el conocimiento existente. Por ejemplo, se puede pensar en esta separación de manera similar a la diferencia que existe entre el modelo de clases y el modelo de datos de una aplicación.

En muchos casos, crear y representar un modelo de dominio puede llegar a convertirse en una tarea bastante compleja. Se deben tener en cuenta diferentes factores, como el correcto balance entre genericidad y especificidad, la separación efectiva de los conceptos, la delimitación y mapeo de los diferentes contextos dentro del dominio, la definición de restricciones, la preservación de la integridad de los mismos, etc.

Estas consideraciones se enlazan con otros campos de estudio muy relacionados, como son la Representación y Razonamiento del Conocimiento ("*Knowledge representation and reasoning*", KR) dentro de la Inteligencia Artificial ("*Artificial Intelligence*", AI) y sobe todo con la Ingeniería Ontológica ("*Ontology Engineering*", OE).

Esta última disciplina, estudia los mecanismos y metodologías para construir esquemas conceptuales, denominados ontologías, que constituyen una representación formal de un grupo de conceptos dentro de un dominio y de las relaciones entre dichos conceptos. En la figura 6 se puede ver un esquema de estas relaciones.

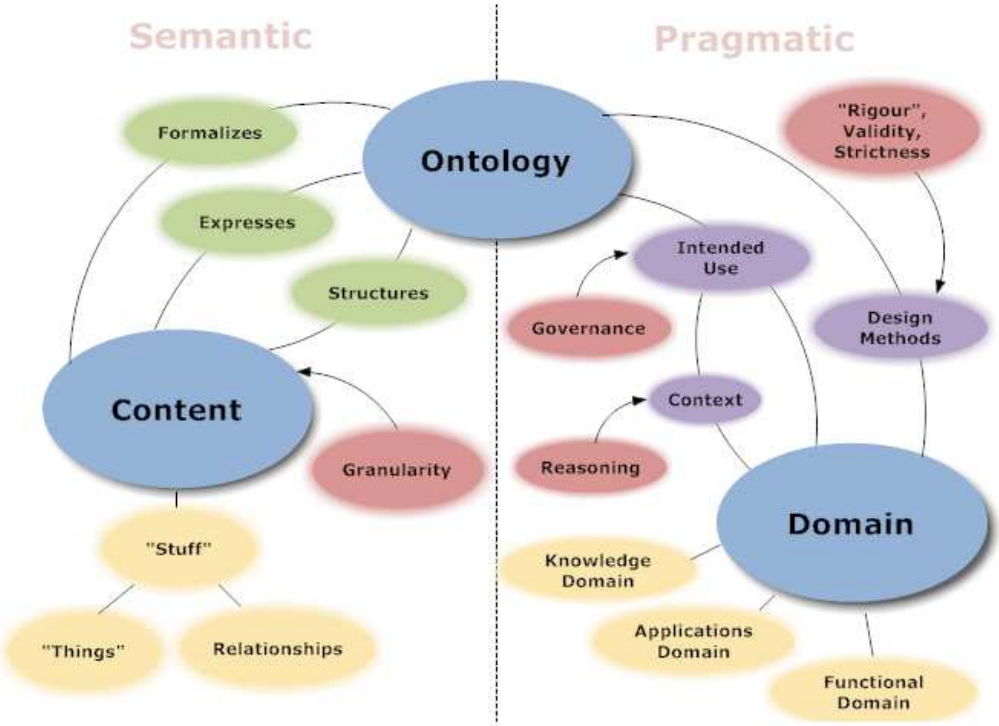

Figura 6: Mapa de dimensiones dentro del desarrollo ontológico

Durante los últimos años se ha incrementado en gran medida el soporte de ontologías dentro de algunos tipos de sistemas de información, encaminados sobre todo al procesamiento automático y a la interpretación de información. Hasta llegar a ser una de las características más utilizadas en ciertos campos, como por ejemplo la web semántica. Esto ha fomentado también el aumento de las herramientas y lenguajes que permiten realizar la especificación de ontologías, como por ejemplo: OWL, RDF, CycL, OntoUML, etc.

1.3.4. Ingeniería dirigida por modelos

Tomando como base la ingeniería de dominio y el desarrollo de modelos de dominio, la posible aplicación de estas características dentro de la ingeniería y el desarrollo de software, ha resultado en la definición de diferentes paradigmas y metodologías relacionadas entre sí.

Una de estas aproximaciones es la conocida como Diseño Dirigido por el Dominio ("*Domain Driven Design*", DDD). La principal característica es su enfoque centrado en utilizar siempre el modelo y la lógica de dominio, en los que se deben basar los diseños complejos del sistema. De esta forma se realiza una colaboración entre los expertos técnicos y del dominio, que comparten un mismo lenguaje ubicuo ("*Ubiquitous Language*", UL), para abordar los diferentes problemas a resolver.

Pensar y centrar las decisiones en el dominio ayuda a crear una arquitectura e implementación del sistema más robusta y fiable a largo plazo. En la figura 7 se pueden ver las fases habituales del ciclo de vida completo de un sistema desarrollado de esta forma.

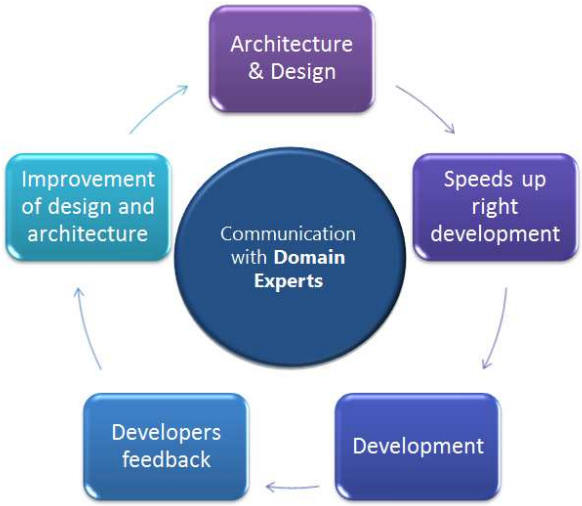

Figura 7: Ciclo de vida habitual en el Diseño Dirigido por el Dominio

Aunque sería deseable conseguir un único modelo que represente de forma estructurada todos los conceptos del dominio, en la práctica los sistemas complejos no escalan de una forma adecuada con este planteamiento. Utilizando una visión más estratégica, además del modelo principal o núcleo, dentro de DDD se suelen definir diferentes modelos para contextos delimitados ("*Bounded Contexts*", BC) del dominio, que hacen referencia a un subconjunto de conceptos relacionados.

Un ejemplo clásico de la necesidad de estos contextos se produce con algunos términos como una "cuenta". Dentro del dominio este concepto puede hacer referencia a una "cuenta bancaria" en el contexto de la realización de pagos o cobros, o a una "cuenta de usuario" en el contexto del acceso y la seguridad del sistema.

Una de las necesidades principales dentro de diseño guiado por el dominio, es intentar asegurar la integridad de los modelos. Para ello, suele ser necesario también crear un Mapa de Contexto ("*Context Map*") que relacione unos contextos con otros, en los que se vean sus dependencias y se expongan los conceptos del dominio que son equivalentes pero en diferentes contextos. En la figura 8 se puede ver un ejemplo concreto de este mapeo.

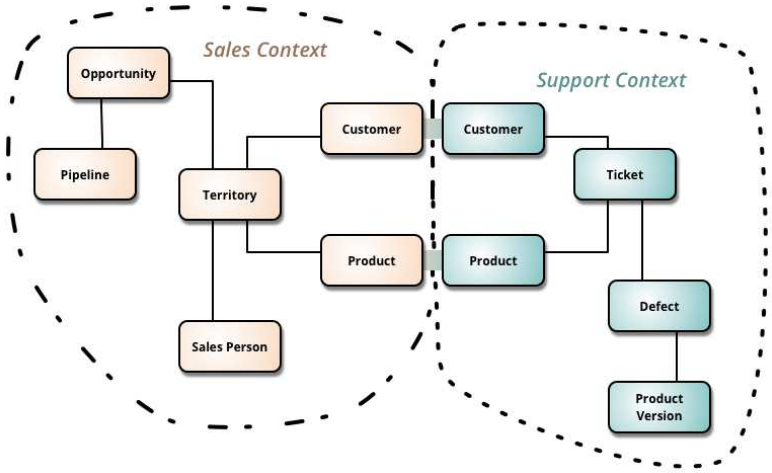

Figura 8: Mapeo de contextos delimitados en el diseño dirigido por el dominio

Otra aproximación diferente es la contemplada dentro de la Ingeniería Dirigida por Modelos ("*Model Driven Engineering*", MDE). Se trata también de una metodología de desarrollo de software centrada en la creación y uso de diversos modelos de dominio, que tratan de resaltar las representaciones abstractas del conocimiento y las actividades que gobiernan un dominio de aplicación frente a los conceptos más técnicos o computacionales. El objetivo que se pretende conseguir en este caso es elevar el nivel de abstracción en la especificación de los productos y proporcionar los mecanismos de automatización necesarios para generar los sistemas finales a partir de los diferentes modelos construidos.

Esta metodología combina la especificación de los modelos de forma declarativa a través de algún tipo de Lenguaje de Modelado Específico del Dominio ("*Domain-specific Modeling Language*", DSML), descrito a su vez mediante un metamodelo que recoge las relaciones entre los posibles elementos; junto con la aplicación de paradigmas generativos, mediante la incorporación de motores de transformación de los modelos y generadores automáticos de código.

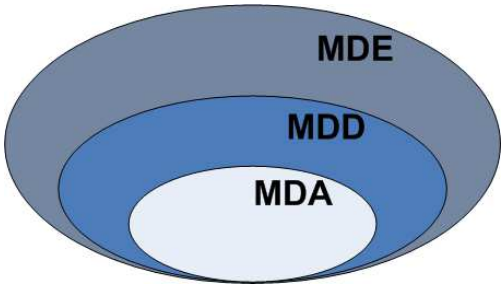

Figura 9: Relación entre las metodologías dirigidas por modelos

En la figura 9 se puede ver la relación con otras dos aproximaciones basadas en los conceptos de MDE que han sido estandarizadas por el Object Management Group (OMG), como son la Arquitectura Dirigida por Modelos ("*Model Driven Architecture*", MDA) y el Desarrollo Dirigido por Modelos ("*Model Driven Development*", MDD).

En realidad, la Arquitectura Dirigida por Modelos es un tipo de estrategia de diseño de software dentro de la ingeniería de dominio que soporta el desarrollo dirigido por modelos a través de tres puntos de vista diferentes. Dentro de MDA, el contexto y los requisitos del sistema son fijados utilizando un Modelo Independiente de la Computación (CIM), mientras que la operativa y características se definen en un Modelo Independiente de la Plataforma (PIM). Mediante la realización de mapeos y transformaciones estos modelos se traducen en uno o varios Modelos Específicos de la Plataforma (PSMs) que se ensamblarán y utilizarán para producir el sistema final.

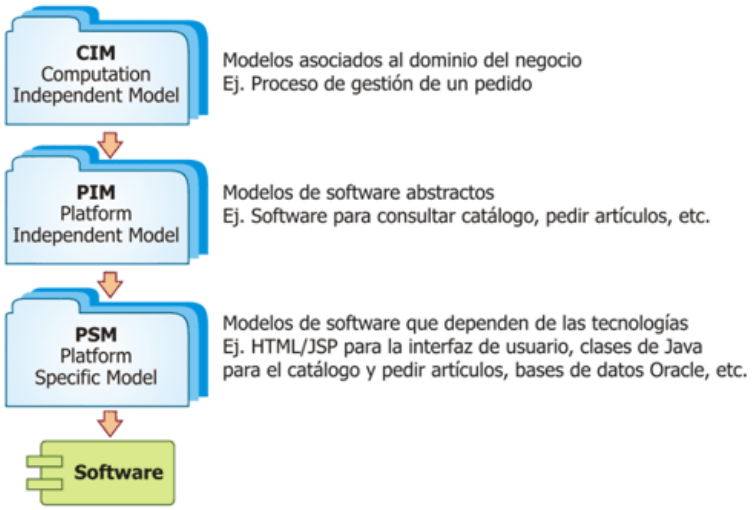

Figura 10: Modelos definidos dentro de la Arquitectura dirigida por Modelos

Dentro de estos enfoques, la utilización de un Lenguaje Específico del Dominio ("*Domain Specific Language*", DSL) suele ser una característica fundamental que aporta enormes beneficios. Por definición, un lenguaje específico de dominio es un lenguaje informático especializado en un dominio de aplicación particular. En la práctica esto supone que incorpora internamente las abstracciones de los conceptos importantes del dominio en su propia estructura y definición.

Los lenguajes específicos de dominio se suelen clasificar en todos tipos principales en función de su naturaleza. Por una parte, se encuentran los DSL independientes ("*stand-alone*") que constituyen lenguajes completos desde el inicio y que incorporan su compilador o intérprete específico. En segundo lugar, estarían los DSL embebidos ("*embedded*") que se han creado como huésped de otro lenguaje de propósito general.

También relacionado se encuentra el denominado Modelado Específico del Dominio ("*Domain Specific Modeling*", DSM). Una metodología utilizada para diseñar y construir sistemas software que implica el uso sistemático de lenguajes específicos de dominio para modelar o

representar los diferentes aspectos del mismo. En este enfoque el código fuente es también generado, pero directamente a través de los modelos basados en los DSLs.

La principal diferencia de DSM con los enfoques anteriores es la utilización de estos lenguajes, creados internamente o a medida para las soluciones. Mientras que MDA y otros enfoques suelen hacer uso de lenguajes de modelado de propósito general, como por ejemplo el Lenguaje de Modelado Unificado ("*Unified Modeling Language*", UML). Lo que permite una mayor interoperabilidad y el uso de herramientas estándar en el mercado.

1.4. Alcance

Al analizar el ecosistema de desarrollo de los productos software basados en procesos de análisis de datos mediante soluciones Big Data, una de las principales conclusiones que se pueden extraer es que nos encontramos ante una auténtica revolución en este ámbito. Existen un número casi ilimitado de posibilidades y posibles áreas de aplicación dentro de este tipo de soluciones analíticas. Por lo que se ha considerado fundamental fijar y delimitar el alcance inicial del trabajo a una serie de problemas concretos.

A pesar de la amplia experiencia en la realización de este tipo de productos, realizar esta tarea no ha resultado trivial. Se necesitan conocer a fondo los planteamientos más comunes y seleccionar para incluir aquellas características necesarias que hagan que la solución propuesta se pueda aplicar para el desarrollo de procesos lo suficientemente genéricos pero interesantes. Para fijar este alcance, se han tenido en cuenta tres aspectos fundamentales de la solución:

- La definición de relaciones y conceptos del dominio.
- La utilización de características analíticas avanzadas.
- La integración con las plataformas y herramientas más comúnmente utilizadas.

En primer lugar, se ha decidido deliberadamente no limitar la definición de elementos y conceptos del dominio. Permitiendo que la solución se pueda aplicar a prácticamente cualquier entorno o sector de negocio. Aunque pueda parecer que no se está limitando el alcance sino añadiendo mayor complejidad, lo que se pretende con este requisito es precisamente separar la definición de conceptos del dominio de la definición de características del producto software. Por tanto, la solución únicamente va a soportar la importación de dichos conceptos y no su completa definición. Es decir, se va a utilizar un lenguaje externo de definición o modelado de propósito general para los conceptos del dominio.

En segundo lugar, ha sido necesario poder delimitar de alguna forma las posibilidades analíticas que se pueden incluir en los diferentes procesos. En lo que se refiere por ejemplo a definición de los datos a analizar, operaciones soportadas de consulta y transformación sobre dichos datos, generación de nueva información derivada, aplicación de algoritmos matemáticos, realización de modelos estadísticos, etc.

En este sentido, se ha optado por soportar un proceso metodológico clásico de análisis predictivo de datos, que incluye fundamentalmente la realización de modelos estadísticos predictivos ("*Predictive Modeling*", PM). Estos se enmarcan directamente dentro del campo del aprendizaje automático supervisado ("*Supervised Machine Learning*", SML).

En resumen, el objetivo del proceso será poder entrenar un modelo para deducir una función que permita predecir el valor de una variable objetivo, en función de los valores de una serie de variables de entrada. Este problema tan concreto tiene multitud de aplicaciones posibles y sigue siendo la base de la mayoría de productos analíticos.

Las necesidades básicas para la construcción de estos modelos son las utilizadas para limitar el soporte de las siguientes características:

- Definición del conjunto de datos de entrenamiento, a partir de los datos históricos disponibles basados en los conceptos y relaciones existentes en el dominio.
- Definición de las variables de entrada y el target, permitiendo algún tipo de transformación básica y operaciones para generar información derivada.
- Definición del tipo de modelo a utilizar, soportando diferentes tipos de técnicas o algoritmos y sus parámetros de entrada o control para el entrenamiento.

En tercer y último lugar, debido a las múltiples plataformas y tecnologías de desarrollo Big Data disponibles (aunque con muchas similitudes, se podría decir que existe una diferente por cada vendedor o fabricante), se ha decidido acotar también el trabajo desde un punto de vista más técnico.

Se ha decidido incluir soporte únicamente con plataformas de desarrollo de código abierto (evitando soluciones cerradas o únicamente comerciales) en base a su posibilidad de integración de una forma más directa y mediante tecnologías que sean las más utilizadas de forma predeterminada en el mercado. Basándose en estas premisas, de cara al planteamiento de la solución se ha considerado adecuado tener en cuenta fundamentalmente aquellas tecnologías construidas sobre la plataforma y framework de desarrollo Apache Hadoop.

1.5. Planteamiento

Para tratar de resolver las necesidades comentadas dentro del alcance descrito anteriormente, el planteamiento que se ha realizado se basa fundamentalmente en la idea de definir los principios de una solución que permita desarrollar los procesos analíticos de una forma más simple, ágil y modular.

Para ello, la principal premisa es conseguir aplicar un conjunto de conceptos y técnicas más cercanas a la ingeniería del software y los paradigmas de desarrollo dirigido por modelos del dominio, que ya se aplican sistemáticamente y de forma exitosa en otros tipos de desarrollos de software.

El diseño de la solución alcanzada tiene como pieza central un modelo de objetos genérico, que soporta la definición de las características necesarias dentro de los procesos analíticos que componen el producto software a construir y que sirve al mismo tiempo como base para un Lenguaje Específico del Dominio (DSL), que facilita esta definición mediante la generación de los elementos y el establecimiento de los atributos apropiados.

Además de la propia definición de procesos, el DSL posee también una característica fundamental en este planteamiento. Permite incorporar al modelo utilizado todos aquellos conceptos (entidades y relaciones) específicos del dominio a analizar a través de una serie de ontologías. Cada una de estas ontologías se podría comparar con un contexto delimitado dentro del dominio global. De esta forma, se habilitarán una serie de mecanismos para que sean los propios conceptos del dominio los que dirijan la definición de características.

Para introducir de mejor forma este planteamiento y comprender sus elementos diferenciales, se realizará una comparación con otro posible tipo de lenguaje de definición con los mismos objetivos, pero realizado con un enfoque funcional más tradicional. Cuyo tipo de diseño y estructura se encuentran habitualmente en este tipo de desarrollos.

Imaginemos que en el ejemplo planteado se poseen unos datos de ventas de productos que se pretenden analizar. Este es el objetivo principal del proceso de análisis de información y por tanto, del sistema a construir. Esta información se encuentra almacenada de forma tabular en un fichero de gran tamaño identificado mediante el nombre “Ventas.txt”.

En este ejemplo, el lenguaje de definición de características soporta una serie de operaciones analíticas limitadas:

- Definir el tipo y origen de la fuente de datos.
- Definir un proceso basado en una lista de transformaciones sobre los datos.
- Las posibles transformaciones a aplicar son:
    - Filtrar los registros por el valor de una columna
    - Agregar los registros aplicando una función

A continuación se muestra una posible implementación de un lenguaje específico del dominio basado en las operaciones disponibles y la meta-información asociada a cada una para definir un proceso completo.

```
datasource { file="Ventas.txt", output="data1" }
process {
  transformation {
     filter { input="data1", expression="producto=X", output="data2" }
  }
  transformation {
     aggregate { input="data2", column="beneficio",
       function="custom", output="result" }
  }
}
```

Desde el punto de vista funcional, se puede apreciar cómo se define el origen de los datos en la primera sentencia. Dentro de la definición del proceso, se realizan dos transformaciones. En primer lugar, se filtran los datos a través de una expresión, limitando los registros a aquellos asociados al producto "X". Mientras que en la segunda transformación se agrega la columna "beneficio" aplicando una posible función a medida.

Centrándonos en la estructura y formato específicos de este lenguaje, se aprecia cómo está basado en una estructura de objetos jerárquica definida explícitamente, a la que se añaden diferentes parámetros e información contextual en cada posible elemento. Las palabras reservadas se corresponden con las posibles tareas a realizar (datasource, process, transformation, etc.). Esto pone de manifiesto que el foco del mismo es la especificación de dichas operaciones. Una posible ejecución o interpretación de este lenguaje podría dar como resultado intermedio la generación del árbol de objetos asociados a cada una.

Volviendo al enfoque planteado en este trabajo, los elementos centrales de la definición serán los propios conceptos del dominio. En este caso, tendríamos conceptos tales como: "Ventas", "Productos", "Beneficios", etc. Que estarían relacionados entre ellos, por ejemplo de la siguiente manera: "Una Venta corresponde a uno o varios Productos y tiene como resultado un Beneficio". Imaginemos que esta información del dominio ya se encuentra definida dentro de una ontología y que ésta está serializada en un fichero denominado "Ventas.owl".

Al cargar esta ontología a través del DSL, lo que provoca es que automáticamente se puedan utilizar sus conceptos como objetos definidos dentro del propio lenguaje. Siguiendo una serie de técnicas basadas en la meta-programación, se puede conseguir que estos objetos adquieran una serie de propiedades y métodos que serán los utilizados para definir el proceso. Este comportamiento acerca el lenguaje a un enfoque más orientado a objetos y menos procedimental.

A continuación se presenta un posible ejemplo del proceso a construir, definido mediante un DSL con las características descritas anteriormente.

```
Domain.load("Ventas.owl")
Ventas.datasource("file", "Ventas.txt")
Ventas.process([
  Ventas.filter(Producto, "X"),
  Ventas.aggregate(Beneficio).impl(package.operations, fn_custom)
])
```

La primera sentencia es específica del lenguaje y se utiliza para cargar la ontología y definir con ello los conceptos disponibles en el dominio. Los objetos denominados "Ventas", "Producto" y "Beneficio" se corresponden ahora con elementos propios que pueden ser identificados al interpretar el DSL. En la segunda sentencia, se realiza la definición del origen de datos utilizando la propiedad o método "datasource" asociado al propio objeto de dominio. Como parámetros se establecen el tipo y el nombre del fichero. De un modo similar, para definir el proceso se parte directamente de las "Ventas" y se establece la lista de transformaciones que lo forman. Estas transformaciones utilizan nuevamente las propiedades y los elementos del dominio para guiar la especificación.

En la última sentencia mostrada, se puede encontrar además un ejemplo de otra de las singularidades de la solución planteada. El lenguaje está diseñado de forma que tiene la capacidad para hacer una referencia directa a los componentes que implementan las características definidas. Nos referimos en este ejemplo, a la función a medida denominada "fn_custom" que se encuentra dentro del módulo "package.operations". De esta forma, se consigue realizar una separación efectiva entre la definición de funcionalidades y la implementación de los componentes reutilizables que forman parte de la solución.

Dentro de la arquitectura de la solución, este diseño permite habilitar un mecanismo similar al conocido como Inversión de Control ("*Inversion of Control*", IoC), en los que el lenguaje y su modelo de datos forman un marco de desarrollo al que se ha incorporado la capacidad de enlazado o ejecución de los componentes ya disponibles. Todo ello orquestado a través de la interpretación del lenguaje específico de dominio.

Además de esto, para complementar el planteamiento realizado se han incorporado al diseño dos mecanismos específicos encaminados a permitir la extensión y consulta del modelo datos y la utilización del mismo para realizar generación automática de código. En este sentido, el modelo de objetos en sí mismo posee un interface de programación (API) para el acceso a la meta-información acerca de la definición del proceso analítico que éste almacena. Esto permite que la información esté accesible a componentes externos o incluso desde el propio lenguaje específico de dominio.

La segunda característica encaminada a la generación de código, hace uso del patrón de diseño Visitor para permitir recorrer todos los objetos del modelo, desacoplando su estructura interna del algoritmo de procesamiento. Este mecanismo más avanzado se puede utilizar de manera directa para la generación de otra serie de productos (documentación, monitorización, generación de código fuente dinámico, etc.) que complementen los resultados el análisis de datos llevado a cabo.

En la figura 11 se puede ver un diagrama general de la solución y del marco de desarrollo propuesto. Se puede apreciar cómo el uso del lenguaje específico de dominio constituye la pieza central que enlaza el módulo principal con los componentes software reutilizables del sistema, haciendo uso de los propios conceptos del dominio. Estos componentes implementarán las características definidas, que permiten analizar los datos y generar los resultados del proceso analítico.

A través de la interpretación del DSL se genera el modelo de objetos, que incluye el API que permite a los componentes acceder a la especificación del propio proceso (configuración, parámetros, etc.). Estas capacidades pueden ser utilizadas también por los posibles generadores de código que construyen otra serie de elementos que forman parte o hacen uso a su vez de los resultados de producto.

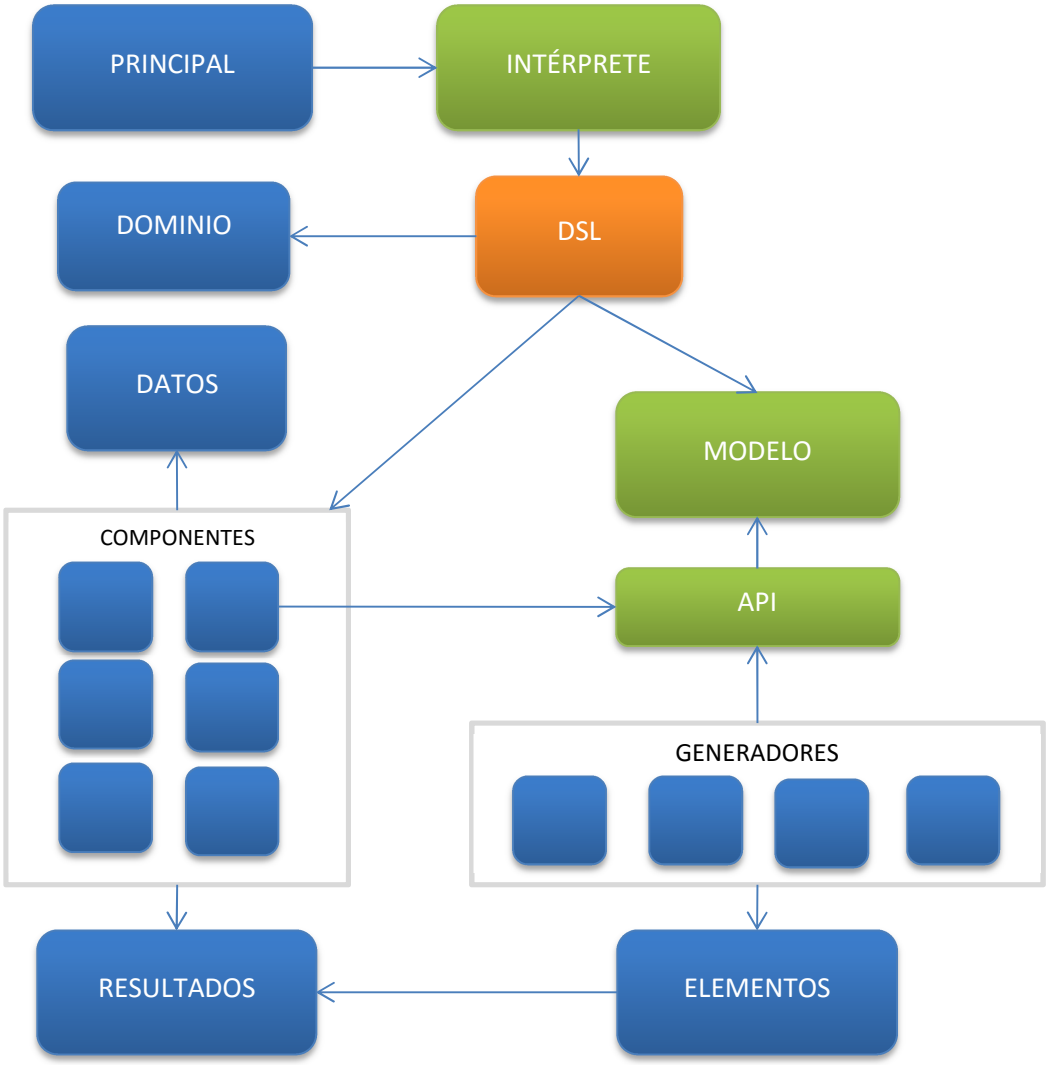

Figura 11: Diagrama general de la solución planteada

Finalmente, el planteamiento realizado se ha complementado con la construcción de un prototipo que permite validar que la aplicación de estas mejoras se puede llevar a cabo de una manera efectiva. De cara a realizar la implementación se han evaluado las alternativas que mejor se ajustan a los objetivos del trabajo. Como resultado, se ha seleccionado y utilizado fundamentalmente el lenguaje de programación Python para su desarrollo.

Python es un lenguaje orientado a objetos con un carácter generalista, simple y de alto nivel. Inspirado en otros lenguajes, ofrece múltiples características muy avanzadas de cara tanto al desarrollo de lenguajes específicos de dominio, como para la integración con diferentes librerías y plataformas. A pesar de tener ya un largo recorrido, estas características lo han puesto nuevamente en primera línea en los últimos años. Muchas de las soluciones analíticas actuales se han implementado y portado a Python, o incluyen soporte para su uso a través de este lenguaje.

1.6. Objetivos

A modo de resumen, se enumeran a continuación los objetivos concretos que persigue el trabajo fin de máster a través del planteamiento de la solución realizada:

1	Investigar sobre las necesidades de evolución y mejora en el desarrollo de productos software basados en procesos analíticos y de minería de datos que utilizan tecnologías o soluciones Big Data.
2	Evaluar las posibilidades que se podrían aportar o incorporar en la construcción de este tipo de productos.
3	Estudiar el estado del arte en esta materia, poniendo el foco en los trabajos previos realizados y los productos o soluciones que puedan estar disponibles en el mercado.
4	Profundizar en las posibilidades de aplicación de diversos conceptos dentro de los campos de la ingeniería de software, ingeniería de dominio, el diseño y desarrollo dirigido por modelos de dominio.
5	Realizar la especificación de una solución general basada en los conceptos y planteamientos anteriores para la descripción de las características de dichos productos software.
6	Diseñar una solución concreta, apoyándose en el uso de modelos abstractos y ontologías como forma de describir el dominio de análisis.
7	Diseñar un lenguaje específico del dominio que resulte sencillo de utilizar para especificar las características del software a construir.
8	Realizar la implementación de un prototipo utilizando las herramientas y entornos de desarrollo más apropiados para la integración con las plataformas y marcos de trabajo más utilizados en la actualidad.
9	Demostrar la utilización de las soluciones y conclusiones extraídas, aplicando los resultados dentro de casos similares a los presentes en la realidad.
10	Comprobar que la solución aportada permite mejorar las metodologías de desarrollo de este tipo de productos software analíticos, permitiendo incrementar el nivel de abstracción, la trazabilidad de características, la modularización y la reutilización de componentes.

Tabla 2: Resumen de objetivos concretos del trabajo

2. Estado del arte

2.1. Situación actual

La evolución del desarrollo de software es un campo en el que se pueden encontrar múltiples líneas de investigación y gran cantidad de material publicado. Sin embargo, no se pueden encontrar demasiadas referencias al restringir la búsqueda a un ámbito más específico, basado en procesos analíticos y ciencia de datos. Históricamente, las investigaciones en este ámbito se han centrado principalmente en resolver las necesidades de mejora en lo que a algoritmos, técnicas y plataformas arquitectónicas se refiere. Olvidando en cierta medida la evolución de los procesos y metodologías de desarrollo.

Por otra parte, la denominada ciencia de datos es una práctica reciente que se encuentra muy influenciada por la industria y el devenir del mercado. Incluso su propio nombre es un término bastante comercial. Esto ha sido utilizado por diversos autores para reclamar la necesidad de aplicación de métodos más científicos dentro de esta disciplina. Aunque se han desarrollado algunos estándares concretos, la evolución y mejora de las metodologías de desarrollo se realiza muchas veces internamente en las propias empresas que construyen los productos software. Es habitual que estas empresas adopten también las formas de desarrollo propuestas por los vendedores, según la plataforma que hayan adquirido o que utilicen para construir sus procesos analíticos.

Del mismo modo, los marcos de trabajo más habituales enfocados al desarrollo analítico raramente habilitan flujos de trabajo concretos. En muchos casos se trata de un conjunto de librerías de componentes y algoritmos de minería de datos para el aprendizaje automático, accesibles a través de interfaces de programación mediante lenguajes de programación de propósito general. Al tener un nivel de abstracción tan bajo se consigue una mayor libertad, pero también impide su uso a mayor escala. Además de esto, algunos de los marcos de trabajo son muy específicos y están orientados a su aplicación en un tipo de problema o mercado concreto, por ejemplo: “video analytics para la predicción de fraude” o “text mining para la detección de sentimiento”.

Este campo de investigación se encuentra actualmente muy abierto y se está potenciando en gran medida, haciendo que sea otro de los motivos para desarrollar este trabajo. Como se ha podido ver, la solución planteada está influenciada por varios conceptos diferentes, extraídos de otras ramas del desarrollo de software, que ya se están aplicando en la actualidad. Es por esto que los trabajos relacionados hacen también referencia a otros dominios de aplicación diferentes, resultando difícil el poder realizar una comparación de la solución alcanzada con otras similares propuestas previamente. En definitiva, se aprecia una clara necesidad de incrementar la investigación en este campo para intentar mejorar y proponer estándares para el desarrollo de software basado en la ciencia y minería de datos. Además es fundamental también una evolución que permita eliminar la cohesión que existe con las soluciones o plataformas utilizadas.

2.2. Soluciones disponibles

Al igual que ocurre en otros ámbitos, en muchos casos la forma de desarrollar software analítico se encuentra ligada a las plataformas y marcos de trabajo disponibles más utilizados en la industria. Nos encontramos por tanto, con que las metodologías y flujos de trabajo más habituales han sido definidos por los propios vendedores de soluciones analíticas y plataformas Big Data en base a sus productos y experiencia.

Como ejemplos de estas soluciones, se encuentran en primer término una serie de grandes productos comerciales con algunas capacidades Big Data, diseñados para proporcionar a los usuarios la capacidad de explorar y analizar los datos, desarrollar modelos analíticos y diseñar flujos de trabajo analíticos, por si mismos de una forma muy visual, prácticamente sin necesidad de implementar ningún tipo de código. Entre estos destacan: IBM SPSS Modeller, SAS Enterprise Miner, Oracle Advanced Analytics, SAP Predictive Analytics, RapidMiner, Alteryx, etc.

A pesar de que estos productos están más enfocados al uso por parte de un analista de negocio o un usuario final, todos ellos poseen también versiones u opciones avanzadas para que los científicos e ingenieros de datos puedan realizar implementaciones y desarrollos a medida. Otras soluciones habituales con características semejantes que se podrían citar también son: Weka, KNIME Analytics Platform, Microsoft Revolution Analytics, Teradata Aster Platform, etc.

Aun incluyendo lenguajes de programación propios (como SAS) o de código abierto (como R) y otras herramientas de más bajo nivel, estas soluciones constituyen mayormente productos finales que utilizar para facilitar el desarrollo de los procesos analíticos. Una problemática habitual de estas soluciones es además su coste, tanto por sus necesidades de hardware e integración como por su licenciamiento, sólo asequible para grandes compañías. Muchas de estas soluciones llevan años en el mercado y aunque las últimas versiones han evolucionado para proporcionar servicios a través de entornos Big Data en la nube, sería necesaria una mayor regeneración en algunas de sus premisas.

Toda metodología debería ser por definición independiente de las herramientas utilizadas. Por lo tanto, aunque estos productos pueden aportar grandes mejoras, sirviendo de ayuda a la hora de construir procesos analíticos y productos software de este tipo, resulta complicado extrapolar las metodologías utilizadas para su generalización. Además, aunque se incluyen habitualmente procesos, interfaces y flujos de trabajo estándar, estas soluciones no incorporan planteamientos específicos que sirvan de base en la evolución de dichos desarrollos, ni para acercar su planteamiento a un modelo de ingeniería de software, que sería lo deseable en el caso que se está evaluando.

Algo parecido sucede con una nueva generación de soluciones y herramientas analíticas de código abierto basadas en Big Data que han incrementado su popularidad en los últimos años. Estas soluciones, están enmarcadas dentro del ecosistema de "*Apache Hadoop*", como por

ejemplo: "*Mahout*", "*Spark*", etc. y son utilizadas como marcos de trabajo con planteamientos más actuales, a pesar de que tampoco proporcionan características concretas enfocadas, por ejemplo, a la creación y reutilización de componentes.

Este tipo de tareas suele recaer totalmente en el tipo de metodología de desarrollo y gestión del ciclo de vida utilizados por parte del equipo encargado de su construcción. Esto lleva a muchas empresas, dedicadas a la construcción de productos basados en procesos analíticos y soluciones Big Data, a proponer y establecer sus propios procedimientos internos basados en sus necesidades concretas de desarrollo. Para poder seleccionar un framework o proponer una mejora que cumpla con las necesidades de evolución de este tipo de productos software, es imprescindible que se pueda aplicar junto con las metodologías habituales de desarrollo más utilizadas en la industria. En este caso, los procesos analíticos y la minería de datos tienen nuevamente una gran carencia en este sentido.

Una de las primeras aproximaciones en este campo en lo que a metodología y procesos de desarrollo se refiere, es la conocida como "*Knowledge Discovery Process*" (KDD) [5] [6]. Fundamentalmente, este proceso contempla los pasos generales para buscar, identificar y evaluar patrones para extraer conocimiento a partir de los datos almacenados en grandes bases de datos. En la figura 12 se pueden ver los principales pasos del proceso.

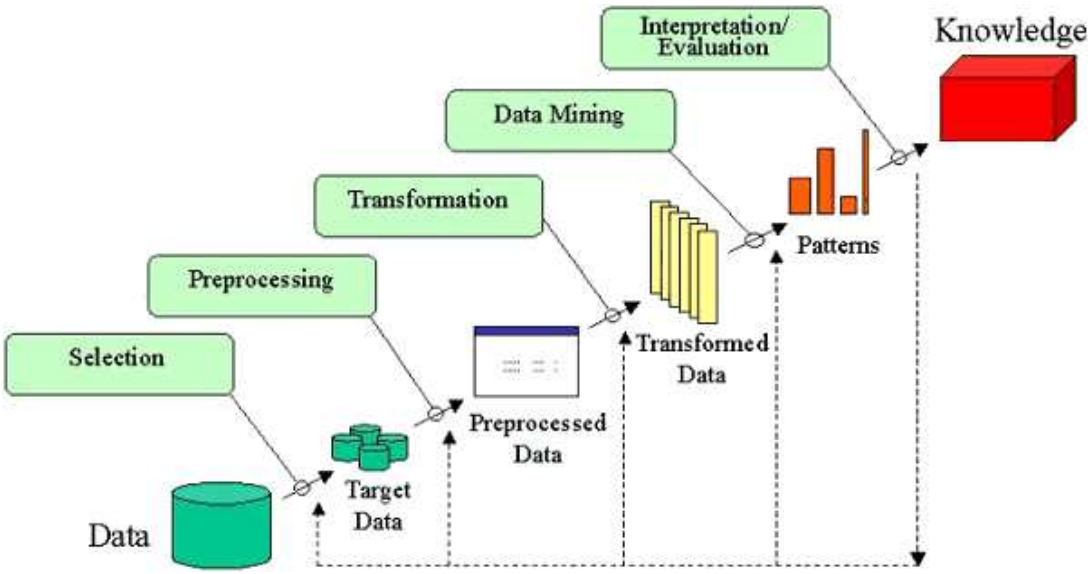

Figura 12: Pasos incluidos en un Knowledge Discovery Process

Otro planteamiento muy similar es el denominado SEMMA [7], acrónimo de "*Sample, Explore, Modify, Model, and Assess*", que ha sido propuesto por el SAS Institute y se encuentra incluido dentro de la metodología propuesta para desarrollo de aplicaciones mediante sus productos. Aunque en muchos casos se ha considerado SEMMA como una metodología general de minería de datos, es en realidad un listado de pasos o tareas recomendables a la hora de realizar un proceso analítico de minería de datos.

La metodología estándar más relevante en el campo de la minería de datos es la denominada "*Cross Industry Standard Process for Data Mining*" (CRISP-DM) [8] [9]. Este modelo de procesos fue concebido originalmente en 1996 y publicado en 1999 como la principal metodología para la minería de datos y analítica predictiva. Durante 2006 y 2008 hubo planes para revisar y actualizar este estándar, aunque los esfuerzos nunca tuvieron lugar. En la figura 13 se puede ver un diagrama general del proceso completo.

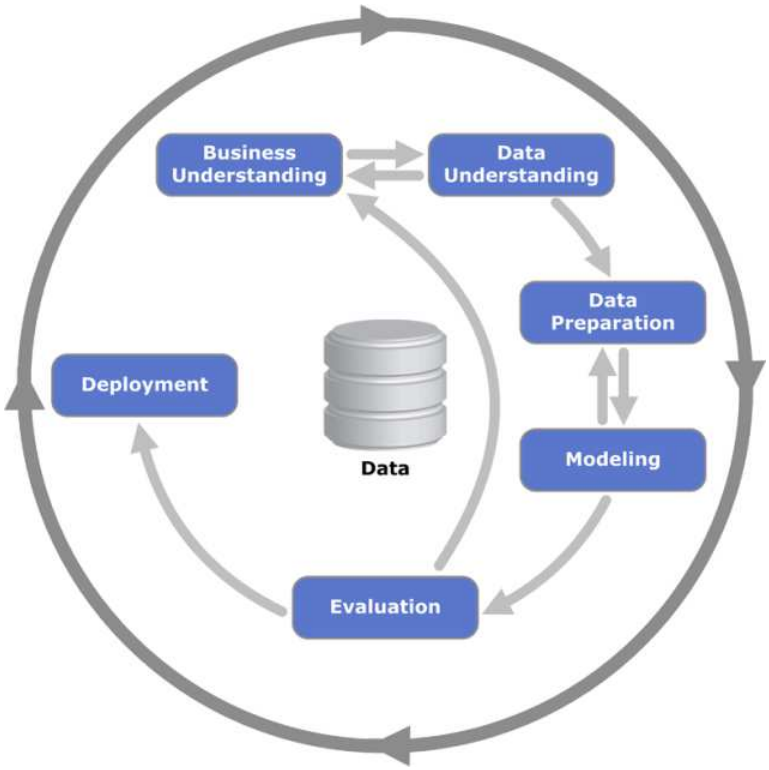

Figura 13: Cross Industry Standard Process for Data Mining

Los puntos fuertes de CRISP-DM están en la parte más analítica del desarrollo, sin embargo, la metodología no cubre aspectos operativos o relativos a la infraestructura de implementación. Posee además muy pocas actividades y tareas relacionadas con la parte de gestión y respecto a la fase de despliegue del producto. En estos puntos, sería deseable que se hubieran incorporado también plantillas o algún tipo de guía de aplicación.

Actualmente tanto el sitio web como el consorcio que realizó su elaboración original ya no se encuentran activos. A pesar de su desaparición, diferentes encuestas llevadas a cabo durante la última década siguen situando esta metodología al frente en su utilización dentro del sector. Lo que puede dar una clara indicación de la falta de alternativas y la necesidad de nuevos planteamientos en este sentido. La figura 14 recoge los resultados de una de las principales encuestas realizadas [10].

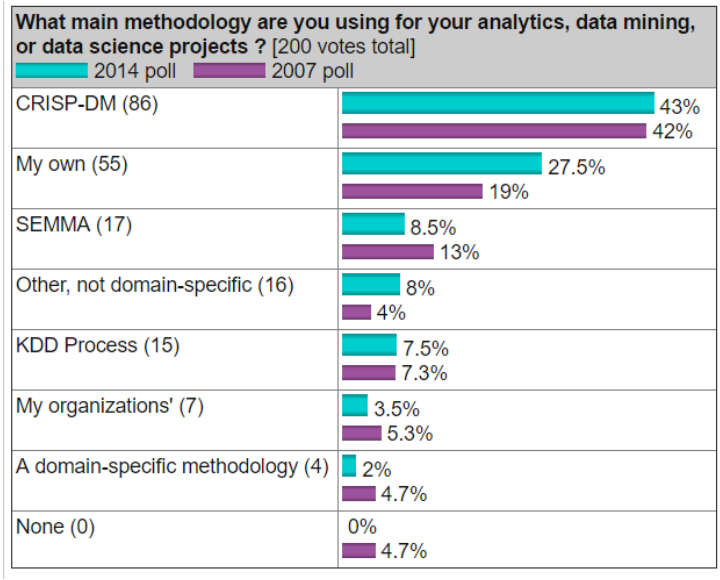

Figura 14: Encuestas sobre metodologías en proyectos de ciencia de datos

En los últimos años, los principales esfuerzos por mejorar este estándar y proporcionar algún tipo de metodología estándar se han desarrollado dentro de la compañía IBM. Como resultado se ha publicado en 2015 la aparición de una metodología que han denominado "*Analytics Solutions Unified Method for Data Mining/Predictive Analytics*" (ASUM-DM) [11] que se considera una extensión y refinamiento de CRISP-DM.

Estos esfuerzos han dado lugar también a otra serie de artículos publicados en torno a la necesidad de una metodología estándar de código abierto: "*OpenSource Foundational Methology for Data Science*" [12]. En la figura 15 se recogen las principales fases de esta nueva metodología.

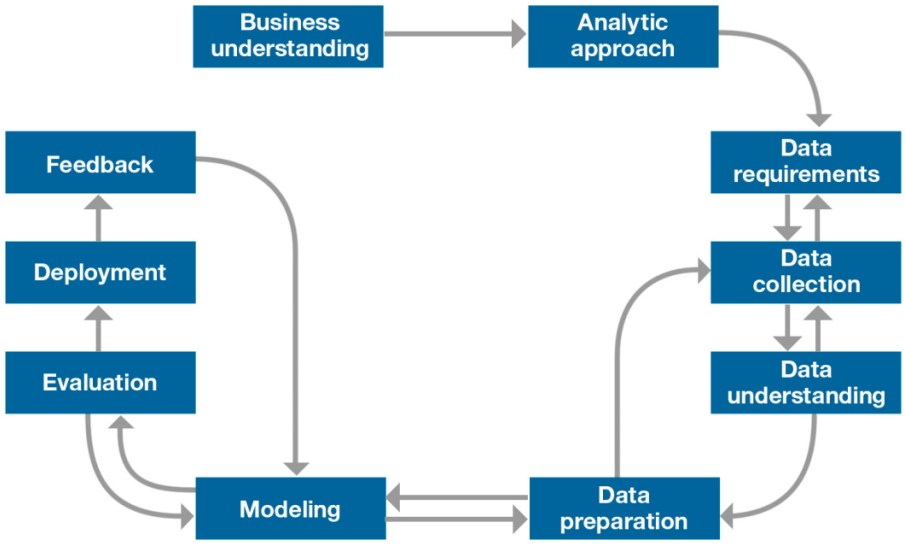

Figura 15: IBM OpenSource Foundational Methology for Data Science

Otras compañías como Microsoft también han ido realizando diferentes esfuerzos por definir aproximaciones que se puedan utilizar junto con sus soluciones en análisis y minería de datos. En este caso, sobre el motor analítico disponible en SQLServer, han definido diferentes extensiones encaminadas a ofrecer alternativas a la hora de desarrollar este tipo de procesos, como por ejemplo: "*Data Analysis Expressions*" (DAX), "*Multidimensional Expressions*" (MDX) y "*Data Mining Extensions*" (DMX).

Además de esto, Microsoft también ha ido realizando diferentes movimientos en los últimos años para situarse a la altura de sus competidores en el ámbito de la analítica avanzada. Esto se puede ver con la creación de "*Azure*" para dar servicios analíticos Big Data en la nube, el lanzamiento de "*Cortana Intelligence Suite*" como producto especializado en analítica avanzada y aprendizaje automático, el desarrollo del lenguaje de programación probabilística incluido en "*Infer.NET*" o la adquisición de la compañía "*Revolution Analytics*" que basaba su negocio en el desarrollo del lenguaje R para el análisis de información y la creación de modelos estadísticos.

Aunque todas estas soluciones están basadas en sus propios productos y aún no se ha publicado ningún estándar concreto, se puede ver a través de su documentación cómo la intención de Microsoft es intentar mejorar los marcos de trabajo y definir flujos de trabajo concretos para el desarrollo de aplicaciones analíticas. Un buen ejemplo reciente es la definición realizada del denominado "*Team Data Science Process*" (TDSP) [13], una aproximación que permite a los científicos de datos colaborar en las diferentes actividades para convertir las aplicaciones con una base analítica en productos, tal y como se puede ver en la figura 16.

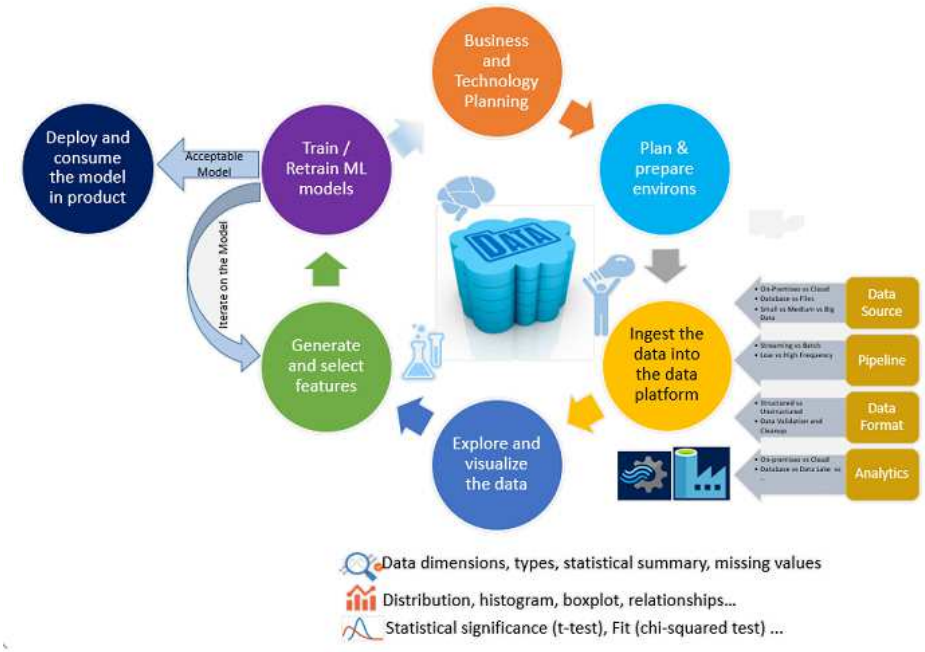

Figura 16: Microsoft Team Data Science Process

Otros organismos independientes dedicados a la elaboración de estándares tampoco han realizado aún ninguna propuesta relacionada con los procesos de desarrollo analítico y metodologías de minería de datos basados en modelos o que incluyan específicamente información relativa al dominio de aplicación. La mayoría de esfuerzos se han centrado en el acceso a la información y la interoperabilidad de los sistemas.

En estos ámbitos, se pueden destacar por ejemplo "*Common Warehouse Metamodel*" (CWM) propuesto por el Object Management Group (OMG). Del que además existe un producto completo que realiza una implementación concreta denominado XELOPES (Prudsys). Por otra parte, el Data Mining Group (DMG) ha propuesto los estándares "*Predictive Model Markup Language*" (PMML) y más recientemente "*Portable Format for Analytics*" (PFA), que permiten compartir meta-información entre sistemas sobre los procesos analíticos, transformación de datos y modelos estadísticos realizados dentro de diferentes aplicaciones.

Todas las soluciones comentadas proporcionan marcos de trabajo específicos en algunos casos, o metodologías generales en otros, para intentar agilizar y mejorar el proceso de desarrollo de sistemas basados en analítica, ciencia o minería de datos. Este ámbito tiene aún un largo recorrido y necesita la definición de propuestas genéricas más avanzadas, independientes de la plataforma o herramientas utilizadas, para elevar el nivel de abstracción y fomentar la reutilización de componentes.

2.3. Trabajo relacionado

Durante los últimos años se puede apreciar en diferentes publicaciones y congresos como existe una tendencia creciente a realizar investigaciones en líneas muy relacionadas con los planteamientos expuestos en este trabajo. Se han publicado diferentes intentos de evolución de las metodologías de desarrollo y marcos de trabajo de minería de datos a través de planteamientos cercanos a la utilización de modelos de dominio, sin embargo, los resultados alcanzados se han enfocado mayoritariamente en otros ámbitos y necesidades.

Uno de los primeros planteamientos propuestos es la posible utilización de la Ingeniería dirigida por Modelos para realizar modelos gráficos que permitan desarrollar software basado en procesos de minería de datos. Los trabajos de investigación realizados por Zubcoff, J. y Trujillo, J. [14] [15] [16] han propuesto aproximaciones concretas para la realización de meta-modelos y procesos de transformación, a partir de modelos independientes de la plataforma, con el objetivo de automatizar la generación de diferentes técnicas de procesamiento y análisis de datos.

Esta línea de investigación se ha continuado mediante los trabajos realizados también por Pardillo J. y Mazón J.N. [17] [18] [19] en los que se aplican los conceptos y técnicas de la Arquitectura Dirigida por Modelos (MDA) a la realización de tareas de depósito, almacenamiento, consulta y transformación de datos ("*data warehousing*").

Este tipo de planteamientos refleja la necesidad de separar la realización de modelos conceptuales, que permitan especificar formalmente las actividades de minería de datos de una forma sencilla y cercana a la forma de pensar de los analistas y científicos de datos. Esta separación se consigue en dichos trabajos mediante la creación de los modelos gráficos basados en estándares MDE (UML) y herramientas de diseño. Mientras que el producto software se genera a través de diferentes transformaciones a representaciones más concretas (otros modelos o texto) y finalmente al código final (por ejemplo: consultas SQL).

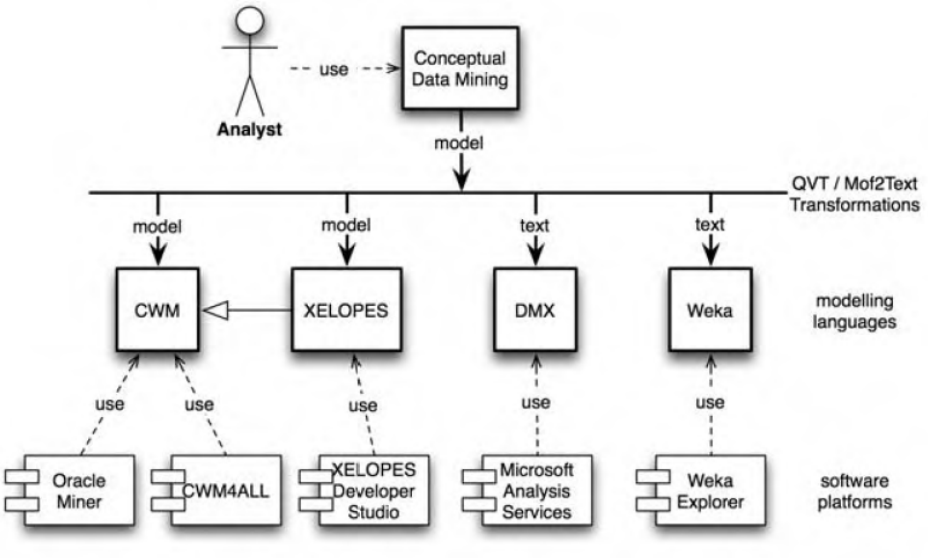

Figura 17: Arquitectura de transformación de modelos para minería de datos

Otros planteamientos se han centrado también en ver la forma de aplicar las técnicas de MDE a través de Lenguajes de Modelado Específicos del Dominio (DSML), como por ejemplo los realizados por Breuker D. [20]. En estos casos, el esfuerzo se ha enfocado en proponer una forma inicial de modelado gráfico a través de un lenguaje de dominio. Este modelo almacenará la información necesaria para realizar una serie de tareas concretas de aprendizaje automático mediante modelos probabilísticos (Inferencia Bayesiana), que se podrán ejecutar en diferentes plataformas (por ejemplo, mediante el lenguaje Microsoft Infer.NET). Este tipo de planteamientos están muy influenciados también por los paradigmas "*Model-Based Machine Learning*" y "*Probabilistic Programming Languages*" descritos en los estudios realizados por Winn J., Bishop C. y Diethe T. [21].

Con el aumento de potencial de las soluciones y plataformas Big Data, otra serie de planteamientos similares se han especializado en acercar la MDA a las arquitecturas Big Data. Un buen ejemplo es el trabajo "*Towards A Model-Driven Design Tool for Big Data Architectures*" realizado por Guerriero M., Tajfar s., Tamburri D. y Di Nitto, E. [22], en la que se utilizan varios meta-modelos diferentes para realizar tanto la especificación de los procesos analíticos, como el diseño de la arquitectura necesaria en soluciones Big Data basadas en el paradigma Hadoop Map-Reduce. Este tipo de soluciones se engloban dentro de las caracterizadas por hacer uso de modelos abstractos para aislar la plataforma de ejecución de las características del proceso a realizar.

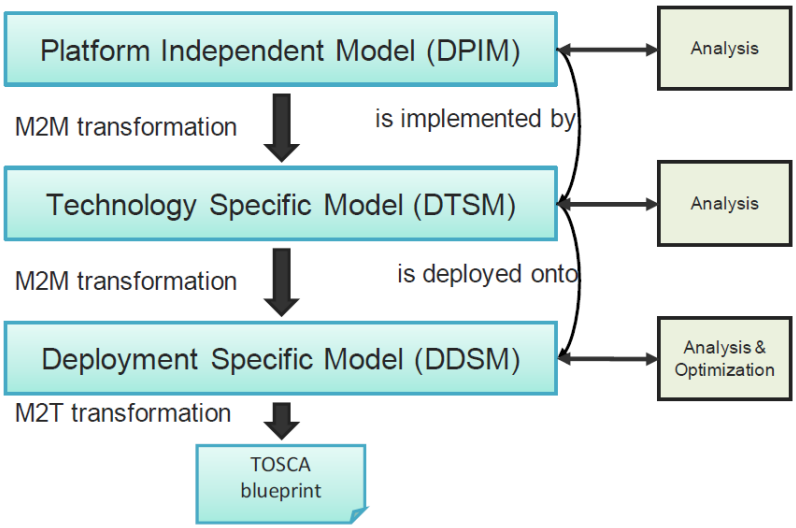

Figura 18: Diseño de la arquitectura MDA para sistemas Big Data

El objetivo fundamental de todos estos planteamientos es aumentar la productividad y la velocidad a la hora de desarrollar los procesos analíticos de minería de datos. Para ello se realiza un proceso de desarrollo basado en MDD, mediante el uso de soluciones estándar y modelos gráficos genéricos, tal y como se puede ver en la figura 18. Aunque es un objetivo difícil debido a la naturaleza matemática de las soluciones, en las que los conceptos estadísticos se encuentran muy correlacionados, se intenta agilizar de alguna forma el desarrollo mediante la generación automática y mejorar la calidad permitiendo incrementar el tiempo en estudiar el dominio y el problema a resolver.

Siguiendo con este objetivo, muchos autores y organizaciones han llegado también a la conclusión de la enorme importancia que tiene incluir los conceptos de dominio en los procesos analíticos [23] [24] [25] [26] [27]. El valor que juega el conocimiento específico del dominio es fundamental en la mayoría de los casos, siendo una de las capacidades más demandadas y requeridas en los profesionales del sector. Si vamos más allá, varios investigadores apuntan a que debería ser la propia información contextual disponible la que guíe el descubrimiento de conclusiones y transforme este nuevo conocimiento adquirido en decisiones accionables. Destacan sobre todo los trabajos realizados por Cao L. y Zhang C. en la definición de la metodología denominada "*Domain Driven Data Mining*" (D3M) [28] [29] [30] [31].

Este planteamiento nace de la necesidad de mejorar las metodologías disponibles, tratando de acercar la minería de datos y los procesos de descubrimiento de conocimiento a los problemas y objetivos de negocio reales. El paradigma considera una evolución desde el Descubrimiento de Conocimiento en Bases de datos ("*Knowledge Discovery in Databases*", KDD) hacia un planteamiento enfocado al Descubrimiento de Conocimiento Accionable ("*Actionable Knowledge Discovery*", AKD). Con esto, lo que se pretende es definir y alcanzar una serie de objetivos guiados por la información de dominio, que pueden ser adoptados como acciones de negocio de forma directa.

En su base, D3M realiza una serie de definiciones que permiten especificar los problemas centrados en el descubrimiento de patrones en los datos como sistemas basados en AKD. Para ello se tiene en cuenta principalmente la incorporación y aplicación de la inteligencia ubicua que se encuentra disponible en el dominio. Aunque estas definiciones son de inicio muy teóricas, han sentado las bases para su aplicación en distintos casos de uso reales, trazando además líneas de investigación muy prometedoras en diferentes ámbitos. Un resumen de estos trabajos se puede ver en los artículos "*Domain-Driven Data Mining: Challenges and Prospects*" [32] también de Cao L. y en "*Introduction to the Domain-Driven Data Mining Special Section*" de Zhang C., Yu P. S. y Bell D [33].

Destaca además el planteamiento realizado de un framework práctico concreto, como alternativa a CRISP-DM, para aplicar los conceptos de D3M a un proceso real de minería de datos a través del reconocimiento de patrones. Este proceso denominado "*Domain-driven In-depth Pattern Discovery*" (DDID-PD) se ha definido formalmente en el artículo "*Domain-Driven Actionable Knowledge Discovery in the Real World*" de Cao L. y Zhang C [34].

Uno de los puntos fundamentales en la Minería de Datos Dirigida por el Dominio es la forma de incorporar la propia información de dominio, es decir, los conceptos, relaciones y atributos que expresan el conocimiento disponible. Debido a sus ventajas, muchos de los planteamientos realizados en este sentido, utilizan como base la definición de ontologías para llevar a cabo esta representación del conocimiento específico de dominio.

El trabajo "*An Ontology Driven Data Mining Process*" de Brisson L. y Collard M. [35] trata la forma de integrar el conocimiento en el proceso de minería de datos, a través de una metodología denominada KEOPS y basada en el uso de ontologías (figura 19). Esta aproximación estudia también la posible evolución del modelo de procesos CRIPS-DM mediante la incorporación de conocimiento del dominio y la construcción de Sistemas de Información Dirigidos por Ontologías ("*Ontology Driven Information System*", ODIS).

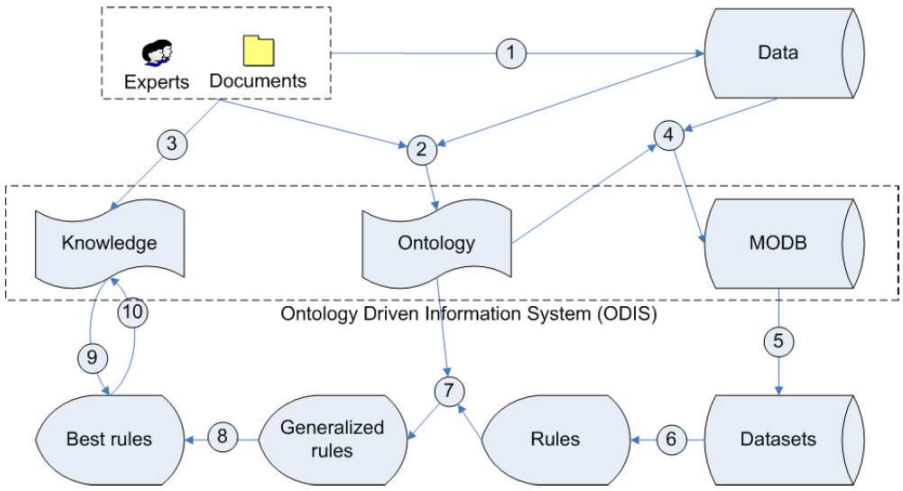

Figura 19: Modelo de procesos incluido en la metodología KEOPS

El uso de sistemas dirigidos por ontologías permite también combinar el conocimiento del dominio con el conocimiento experto en el proceso de KDD, como se puede ver en la figura 20. Esto permite mejorar y guiar las propias características del proceso analítico e incluso realimentar o actualizar la base de conocimiento disponible con los resultados obtenidos de la ejecución del proceso. Por ejemplo, si el algoritmo o los parámetros utilizados han sido o no útiles para alcanzar el objetivo esperado. De esta forma se consigue reutilizar el conocimiento de dominio adquirido, no sólo para validar los resultados o seleccionar las técnicas de pre-procesamiento más adecuadas, sino a lo largo de todo el proceso.

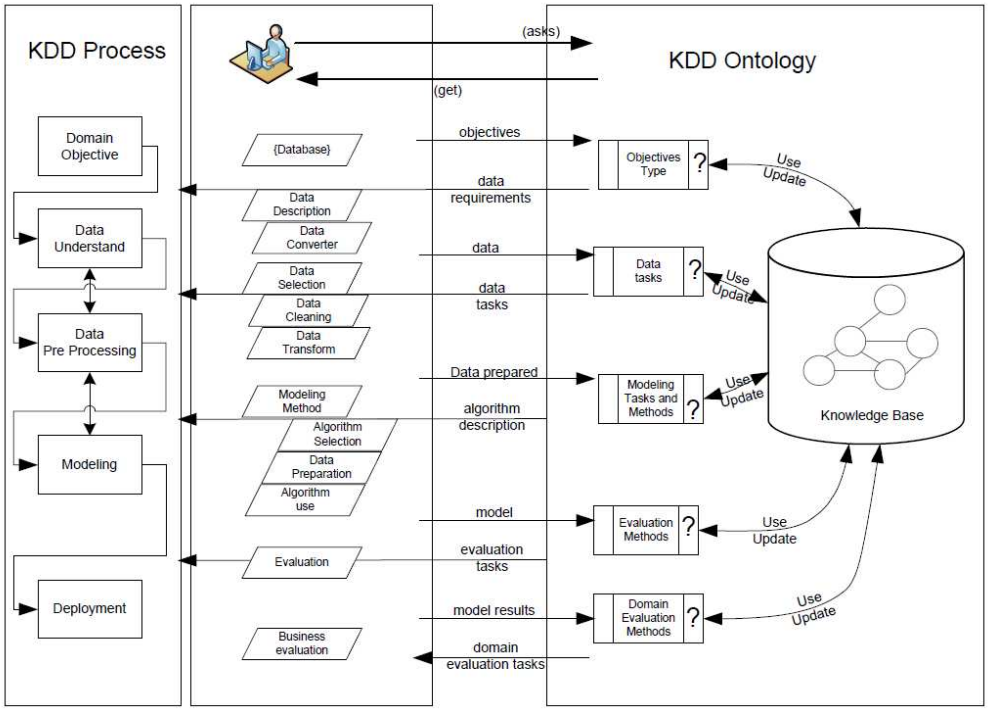

Figura 20: Aplicación de ontologías en el proceso de KDD

En el trabajo "*Considering Application Domain Ontologies for Data Mining*" realizado por Mota F. y Santos M.F. [36] se plantea una forma de mejorar cada una de las fases de estos procesos KDD a través del uso de una Ontología de Minería de Datos ("*Data Mining Ontology*", DMO). Otros ejemplos similares son los planteados por Panov P., Dzeroski S. y Soldatova L. en su definición de "*OntoDM: An Ontology of Data Mining*" [37] y también en "*Exposé: An Ontology for Data Mining Experiments*" de Vanschoren J. y Soldatova L. [38].

Este último artículo no sólo define una ontología específica para los procesos de aprendizaje automático, sino que incorpora su uso en conjunción con Bases de datos de Experimentos ("*Experiment Databases*", ExpDBs) [39] [40]. Estas bases de datos permiten almacenar y organizar en diferentes repositorios los detalles de los experimentos y flujos de trabajos realizados. De forma que se habilita una forma simple de reproducir los mismos experimentos y permite tener una trazabilidad de las pruebas que ya se han llevado a cabo.

Este tipo de planteamientos tienen como punto de partida para la mejora de los procesos, la reutilización efectiva de los flujos de trabajo que han sido exitosos en ocasiones anteriores para alcanzar soluciones a problemas similares. Un buen ejemplo se puede ver en el trabajo "*Towards Cooperative Planning of Data Mining Workflows*" de los autores Kietz J-U, Serban F., Bernstein A. y Fischer S. [41].

Esta posible automatización o selección de los mejores escenarios, pasos y operadores a utilizar tiene como fin la posible construcción de asistentes inteligentes para el descubrimiento de información ("*Intelligent Discovery Assistant*", IDA) basados en la utilización de ontologías. Esta línea de investigación propuesta por Bernstein A., Provost F. y Hill S. en su trabajo "*Intelligent Assistance for the Data Mining Process: An Ontology-based Approach*" [42] [43] posibilita que se pueda ofrecer a los usuarios la opción de seleccionar los procesos de minería de datos válidos para el problema a resolver, ordenados por diferentes criterios, de manera que se puede elegir cuál ejecutar. Este planteamiento se ha desarrollado ya en diferentes trabajos, dando lugar a sistemas inteligentes como IDEA, NExT o CITRUS.

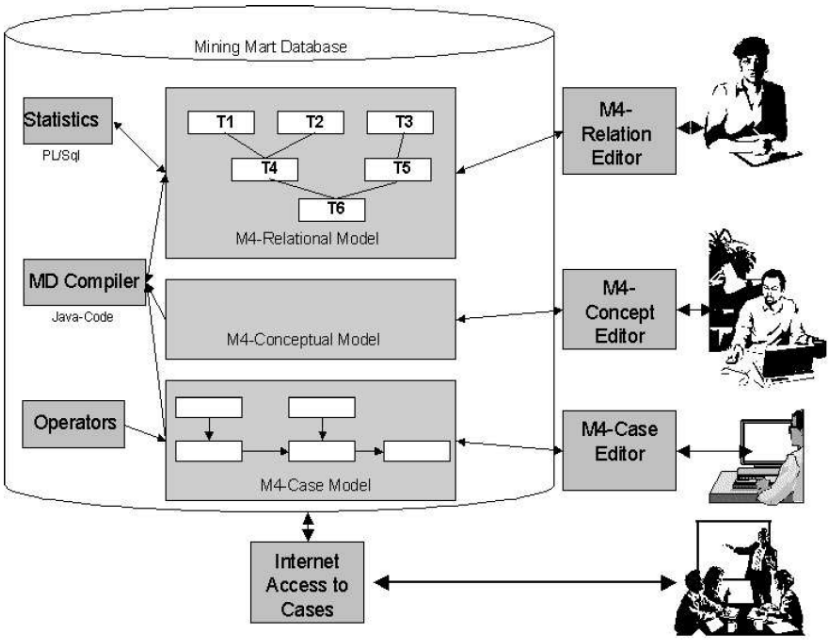

Figura 21: Esquema general del sistema MiningMart

Dentro de estos sistemas, se podría destacar también MiningMart [44], por su planteamiento en la mejora del proceso mediante el uso de modelos y la reutilización del conocimiento de dominio mediante el uso de ontologías. Esta propuesta realizada en el artículo "*The MiningMart Approach to Knowledge Discovery in Databases*" [45] de Morik K. y Scholz M., se centra principalmente en la parte operacional dentro del pre-procesamiento de los datos. El esquema general de su aplicación se puede ver en la figura 21.

Mediante la definición de un meta-modelo de meta-datos (M4) se almacena la meta-información de las operaciones y el procesamiento de los datos, mientras que las relaciones entre los datos y los conceptos del dominio se mapea a través de una ontología específica.

Todo ello se persiste en el sistema, permitiendo su reutilización posterior en la definición de nuevos procesos. Además de esto, MiningMart posee un motor de generación que construye automáticamente el código SQL a partir de la meta-información y el conocimiento almacenado.

Todos los estudios comentados anteriormente tienen una estrecha relación con la temática y el planteamiento realizado en este trabajo. Además, aunque este planteamiento difiere en buena medida de la mayoría de aproximaciones, muchos de estos trabajos han inspirado e influenciado de alguna forma la línea de investigación seguida y la solución alcanzada finalmente.

3. Solución

Este capítulo tiene como objetivo describir el planteamiento y la solución alcanzada durante la realización del trabajo. En primer lugar se procederá con una introducción de los aspectos fundamentales de la solución, para posteriormente, profundizar en el desarrollo e implementación realizados. Por último, se proporcionarán diferentes ejemplos de aplicación de la solución dentro del ámbito de la construcción de productos analíticos basados en minería de datos mediante tecnologías Big Data.

3.1. Enfoque

3.1.1. Descripción

La solución realizada constituye un marco de trabajo diseñado específicamente para el desarrollo de soluciones software basadas en procesos analíticos avanzados, mediante la utilización de tecnologías y plataformas Big Data. Está pensado para ser utilizado por los ingenieros y científicos de datos encargados de construir dichos procesos analíticos en base a su conocimiento del dominio y su estudio de la información disponible.

El objetivo principal de este marco de trabajo es habilitar una serie de mecanismos simples para la elaboración y utilización de un modelo que contiene la especificación de características de los procesos analíticos, independientemente de la implementación, plataforma o tecnología utilizadas en la construcción de dichas características. Esto permitirá utilizar una metodología de desarrollo que fomente la creación de componentes independientes que soporten todas las funcionalidades, pudiendo hacer que éstos sean adaptables, reutilizables e intercambiables entre diferentes procesos y productos software.

La definición del modelo de características se realiza dentro del marco de trabajo de una forma programática, a través de un lenguaje específico de dominio. Las propias capacidades incluidas en este DSL son las que conforman el meta-modelo utilizado, lo que proporciona una mayor agilidad en la definición de los procesos. La solución permite además integrar la ejecución del DSL y el acceso al modelo a través de una serie de interfaces de programación, permitiendo una mayor libertad y mejores capacidades de extensión durante la construcción del software.

Además de esto, el planteamiento de la solución se basa en la integración del conocimiento de dominio, permitiendo utilizar los propios conceptos del dominio del problema, que guiarán tanto la definición de características de los procesos como el análisis de los datos disponibles y la extracción de conclusiones. La forma de incorporar este conocimiento se realiza a través de un conjunto de ontologías, que deberán ser construidas reflejando las clases, relaciones y atributos existentes en los diferentes contextos del dominio de estudio.

Además del planteamiento teórico, la solución alcanzada pretende ser lo más pragmática y usable posible. Para esto, se ha realizado una implementación inicial mediante un prototipo del marco de trabajo, que permite comprobar la posible aplicación de la solución en casos de uso reales. Dentro del posible espectro tan extenso de funcionalidades, las capacidades iniciales seleccionadas y soportadas por el prototipo desarrollado se basarán en las técnicas básicas de análisis y minería de datos, así como la construcción de modelos predictivos supervisados.

3.1.2. Diferenciación

La solución elaborada es claramente diferencial frente a otros productos y plataformas disponibles y también frente a otro tipo de planteamientos o aproximaciones propuestas en otros trabajos de investigación, como los citados en el capítulo anterior referente al estado del arte.

En primer lugar, se ha intentado realizar un planteamiento lo más completo y exhaustivo posible, pero intentando evitar en todo momento que éste sea excesivamente teórico. Se ha puesto el foco en proporcionar una visión mucho más práctica y en plantear una solución que se pueda llegar a adoptar de forma más o menos inmediata. En lugar de definir una nueva metodología o describir un marco de trabajo teórico, se han seleccionado una serie de características actuales ya contrastadas, muy cercanas a la ingeniería y el desarrollo de software, para intentar evolucionar las metodologías de desarrollo actuales y cubrir algunas de las necesidades detectadas.

En segundo lugar, aunque el uso de la solución tiene como objetivo agilizar e incrementar la productividad en el desarrollo de los productos software analíticos, el marco de trabajo elaborado no pretende automatizar completamente el proceso de desarrollo, sino reutilizar los esfuerzos ya realizados. La generación automática del código de los componentes que componen el software o la creación de un asistente inteligente que decida sobre las mejores técnicas a utilizar, son capacidades que se podrían incorporar en paralelo. Puesto que por el momento es necesaria la intervención manual durante gran parte del ciclo de vida del software, el objetivo es ayudar a los ingenieros de software y científicos de datos en su gestión.

Del mismo modo, aunque la utilización de modelos proporciona nuevos paradigmas de desarrollo, no se pretende delegar en analistas funcionales o expertos del dominio la construcción de modelos gráficos de minería de datos. Los procesos analíticos avanzados basados en Big Data requieren las habilidades específicas de los científicos de datos para integrar el conocimiento de dominio con el conocimiento estadístico y matemático. Estos perfiles no están acostumbrados a un nivel tan alto de abstracción, ya que necesitan utilizar lenguajes, algoritmos y técnicas concretas.

Para facilitar además su labor de análisis y validación de las conclusiones, se han incorporado los conceptos de dominio dentro del núcleo de la solución, no como una extensión o característica adjunta. De esta manera, los productos construidos pueden evolucionar desde una perspectiva centrada completamente en los datos, a una nueva visión más centrada en el conocimiento adquirido y el dominio de aplicación.

3.1.3. Utilización

De cara a proporcionar una visión más completa del ámbito de utilización de la solución realizada, dentro de este apartado introductorio, se describirá también brevemente un posible caso de uso. Con este ejemplo no se pretende mostrar la aplicación desde el punto de vista de un usuario, sino que se pretende poner especial atención a la operativa general y los aspectos metodológicos que se pueden conseguir con el uso del marco de trabajo desarrollado, para la construcción de un producto software analítico basado en tecnologías Big Data.

Aunque a priori esta solución se puede aplicar en la resolución de problemas de diferente envergadura, las mejoras aportadas por el marco de trabajo se incrementan y perciben mejor en productos software de un tamaño relativamente considerable. Por ejemplo, en proyectos de desarrollo de un producto compuesto por decenas de procesos analíticos individuales.

El caso de ejemplo considerado se trata de la construcción de un producto analítico encargado de proporcionar recomendaciones sobre las mejores acciones de marketing para los clientes de una compañía. Dentro las posibles ofertas y acciones comerciales disponibles, el objetivo sería ofrecer la siguiente mejor acción ("*Next Best Action*", NBA) en función de la información de ventas disponibles, los datos e intereses de los clientes y las respuestas a las diferentes campañas de marketing realizadas anteriormente.

El planteamiento de este producto implica el desarrollo de varios procesos de transformación y preparación de los datos disponibles. Esto es debido al gran volumen disponible y a su diferente naturaleza, al provenir de fuentes muy dispares. Además, se contempla la generación de una serie de modelos estadísticos que clasifiquen los productos y ofertas a recomendar, agrupen a los clientes en grupos con gustos afines y den una probabilidad de compra o propensión a aceptar cada acción comercial concreta. Por último, el producto debe realizar una combinación de las salidas de todos estos modelos, incorporando las necesidades y reglas de negocio particulares, para finalmente, establecer y ordenar una lista de las acciones concretas a realizar sobre cada cliente.

Como es habitual, el equipo encargado del desarrollo está formado por varios perfiles diferentes: arquitectos e ingenieros de software, analistas y científicos de datos, expertos en el dominio de negocio, programadores, administradores, etc. Todos ellos van a utilizar una metodología concreta de desarrollo en la que será necesaria una separación de responsabilidades y un enfoque colaborativo en cada una de las fases.

Como una fase previa independiente de la aplicación de la solución, se debería realizar un análisis técnico y funcional, así como un diseño arquitectónico, en los que se debería fijar el alcance de la solución y seleccionar las plataformas y herramientas más adecuadas a utilizar. La utilización del marco de trabajo propuesto a través de esta solución, mejoraría y facilitaría el desarrollo de este producto software en varios aspectos.

En primer lugar, se ve necesario un profundo estudio del dominio del problema, para descubrir todas las necesidades existentes. Se deben definir los conceptos más importantes que intervienen y agruparlos en diferentes contextos para un manejo y una posible división del sistema completo. Con esto se pretende generar un lenguaje común, para que todos los grupos de trabajo puedan hablar un idioma común. Toda esta información de negocio contenida en el dominio de aplicación se puede recoger y documentar a través de un conjunto de ontologías.

Seguidamente, se deben estudiar los datos disponibles y diseñar los procesos analíticos que será necesario construir dentro del producto. Este es el punto fundamental en el que se utilizará el lenguaje específico de dominio, incorporado en el marco de trabajo, para realizar la especificación de las características de cada proceso. Estas características se enlazarán directamente con los propios conceptos de dominio ya definidos.

Pensar en las características necesarias y describirlas mediante el DSL ayuda a separar y agilizar el diseño e implementación de los componentes necesarios. Estos componentes tendrán como objetivo realizar una tarea con los datos, pudiendo abarcar funcionalidades muy diferentes, como por ejemplo: la lectura de un conjunto de datos desde una fuente concreta, una fase u operación concreta de transformación, la generación de información intermedia, el entrenamiento de un modelo, etc. Una vez definidos, estos componentes se podrán desarrollar y probar de manera unitaria.

La principal ventaja de utilizar este planteamiento radica en la más que probable necesidad de utilización de estos componentes en varios procesos analíticos diferentes. Por ejemplo, un mismo componente se podría encargar de unificar los identificadores de clientes y productos, si los datos provienen de sistemas transaccionales diferentes. Otro tipo posible de componentes a desarrollar se encargarían de transformar y normalizar ciertos atributos: Eliminar valores erróneos o atípicos (por ejemplo, la edad de un cliente con valores negativos), estandarizar las magnitudes o medidas (por ejemplo, precios en diferentes monedas o datos temporales en distintos usos horarios). Además de esto, sería posible reutilizar también implementaciones de ciertos algoritmos, por ejemplo, si se utilizar el famoso “K-means” para generar grupos de clientes similares por sexo y edad, o si los grupos de clientes se quieren realizar también por categoría de los productos que han adquirido.

Finalmente, a la hora de ensamblar los diferentes procesos e integrarlos dentro del sistema completo, el marco de trabajo ofrece una serie de ventajas diferenciales. La interpretación de los DSL permite generar los modelos de características de cada proceso. Y mediante los interfaces de programación disponibles, estos modelos se pueden utilizar para la ejecución de

los procesos, el acceso a la meta-información o la generación de otros componentes y elementos de forma automática.

Como se puede apreciar, la utilización de este marco de trabajo, y en general, de la solución planteada en este trabajo, permite una evolución y estandarización en las metodologías y los procesos de desarrollo de este tipo de productos software analíticos, agilizando la implementación, fomentando la modularización y la reutilización de los componentes y los conceptos del dominio del problema.

3.2. Características

En este apartado se profundiza, desde una perspectiva más técnica, en las características generales de la solución y del prototipo o marco de trabajo realizados. Poniendo además especial hincapié en describir las decisiones tomadas y las ventajas aportadas en cada caso.

3.2.1. Framework

El principal resultado extraído de la solución alcanzada es la elaboración de un marco de trabajo o "framework" de desarrollo que está compuesto de tres piezas fundamentales:

- MC – Modelo de Características (meta-modelo)
- DSL – Lenguaje Específico de Dominio
- API – Interfaces de Programación

El modelo de características se refiere a la especificación del meta-modelo con el que se definen las funcionalidades que pueden incluir los procesos analíticos de minería de datos a construir.

El lenguaje específico de dominio es la implementación del lenguaje que permite realizar la especificación de características de los procesos, o en otras palabras, mediante el cual se pueden generar y popular los objetos del modelo de características.

Los interfaces de programación son aquellos elementos que permiten manejar y utilizar el marco de trabajo para acceder principalmente a la meta-información contenida en el modelo de características.

Estos tres componentes se encuentran en realidad bastante enlazados entre sí, tanto desde el punto de vista conceptual como dentro de la implementación realizada. Esto se traduce en diferentes dependencias entre sí, al haberse diseñado cada uno para dar soporte y habilitar las funcionalidades del resto.

Toda la implementación de estos tres componentes se ha realizado utilizando el lenguaje de programación Python. En su construcción se han seguido los estándares y convenios de desarrollo habituales definidos para este lenguaje.

3.2.2. Lenguaje específico

El lenguaje específico de dominio que se ha construido constituye la pieza central del marco de trabajo y por tanto, de la solución alcanzada. Este DSL permite la creación de un modelo que define las características de un proceso analítico de análisis y minería de datos, basado en tecnologías Big Data.

Por tanto, el dominio en el que se considera específico este lenguaje es el relacionado con las funcionalidades que pueden albergar este tipo de procesos. Como el conjunto completo de características posibles sería inabarcable, se ha reducido este dominio para definir de forma inicial una serie de características básicas, pero que son al mismo tiempo suficientemente interesantes y necesarias para la resolución de muchos tipos de problemas.

La implementación de este componente posee algunos puntos relevantes. Por una parte, se ha decidido construir un DSL embebido dentro del propio lenguaje de programación utilizado (Python), frente a la construcción de un lenguaje nuevo independiente. Esta decisión se ha tomado con el objetivo principal de agilizar el proceso de desarrollo del prototipo.

Las dos ventajas principales que se pueden considerar al utilizar un lenguaje embebido son el incremento en la velocidad de desarrollo y en la potencia de mano del lenguaje. Esto es debido fundamentalmente a que no es necesario crear un compilador o intérprete específico, ya que se aprovecha el proporcionado por el propio lenguaje utilizado. Además de esto, el DSL utiliza y proporciona de inicio a los usuarios todas las características disponibles en el lenguaje anfitrión. La única posible desventaja es que los usuarios deben conocer también de antemano este lenguaje de programación.

Por otro lado, como ya se ha comentado, el DSL incluye una serie de capacidades para incorporar automáticamente el conocimiento del dominio de análisis o dominio del problema. Esto se traduce en que los conceptos (entidades, relaciones y atributos) que describen y ponen nombre a los conjuntos de datos a analizar, serán reconocidos como elementos del propio lenguaje de definición. Por ejemplo: “Venta”, “Cliente”, “Producto”, etc.

Otro punto a destacar es el diseño y construcción del DSL directamente sobre el modelo de objetos que define el meta-modelo de características. Se han aprovechado todas las capacidades posibles de la programación orientada a objetos y el diseño dirigido por modelos para definir una estructura de datos que pueda almacenar la información del modelo y en la que, al mismo tiempo, los métodos de estos objetos sean los que implementen las funcionalidades del lenguaje.

3.2.3. Python

Python es un lenguaje de alto nivel, generalista, minimalista, interpretado, interactivo y orientado a objetos. Sus principales cualidades se basan en su simplicidad: con pocas palabras reservadas, una estructura simple y una sintaxis bien definida; lo que hace que sea muy fácil de leer y de aprender.

Este lenguaje de programación se ha utilizado para el desarrollo del marco de trabajo y más concretamente para la implementación del lenguaje específico de dominio embebido, tal y como se ha visto en el apartado anterior. Al realizar un DSL embebido y aprovechar el compilador e intérprete del lenguaje original, la estructura y sintaxis del DSL siguen las mismas reglas que aplican en el lenguaje Python.

Python como lenguaje de programación proporciona muchas características avanzadas (scripting, meta-programación, introspección, sobrecarga, redefinición, etc.) que favorecen la creación de un DSL de este tipo. Pero, además de esto, todas las funciones, operaciones, y capacidades incluidas en el núcleo de Python estarán disponibles de antemano para el usuario también dentro del DSL.

Aunque para utilizar el DSL sean necesario un conocimiento básico de Python, en este caso, esta capacidad también se ha valorado de forma positiva y es una de las razones por las que se ha utilizado este lenguaje de programación en todo el desarrollo. En la actualidad existe una tendencia que ha situado a Python al frente (junto con los lenguajes R y Scala), como uno de los más utilizados en las tareas de análisis, minería de datos, modelado predictivo y aprendizaje automático, entre otras. Por lo tanto, muchos posibles usuarios ya tienen un conocimiento básico de este lenguaje y están familiarizados con sus capacidades y sintaxis concreta.

Al utilizar Python, se reducen también las dificultades de integración con todas las librerías, algoritmos y herramientas de análisis de datos disponibles para este lenguaje (por ejemplo: Pandas, NumPy, Matplotlib, etc). Del mismo modo, su utilización enlaza perfectamente con otros marcos de trabajo y plataformas Big Data actuales, que proporcionan interfaces de programación específicos para Python (por ejemplo: Apache Spark, Storm, Kafka) e incluso con otros lenguajes (C, C++ o Java).

Como se puede apreciar, la motivación para decidir utilizar Python en el desarrollo de la solución va más allá de las propias capacidades del lenguaje para el desarrollo del DSL y el marco de trabajo. Su utilización encaja perfectamente en este ámbito analítico y permite cubrir muchas de las necesidades actuales.

3.2.4. Modelos

La utilización de modelos específicos del dominio es una de las piezas centrales, más importantes y diferenciales de la solución realizada. No sólo en el propio desarrollo, sino también en las posibilidades de utilización del marco de trabajo y las ventajas aportadas para los usuarios.

En este punto es clave diferenciar claramente las relaciones entre los meta-modelos y dominios diferentes que intervienen en la solución:

- Por una parte, se encuentra el modelo del dominio del problema o dominio de análisis. Este modelo es más cercano al negocio y es sobre el que hacen referencia los datos que se pretenden analizar. El conocimiento de este dominio varía en función del tipo de sector y ámbito de aplicación. Por ejemplo, los conceptos de un dominio financiero o bancario ("transferencia", "depósito", "hipoteca"…) son totalmente diferentes a los incluidos en un dominio médico o de salud ("paciente", "medicamento", "dosis"…)
- Por otra parte, se encuentra el modelo de dominio de la solución o del proceso a realizar. Este modelo es el definido dentro del dominio de análisis y minería de datos, aprendizaje automático, modelos estadísticos predictivos, etc. Los modelos concretos generados variarán en cada proceso, aunque los conceptos del dominio son siempre los definidos en el marco de trabajo ("origen de datos", "transformación", "modelo", "variable"…)

El primer modelo define los conceptos que guiarán el desarrollo y los diferentes pasos que componen los procesos analíticos en los que se basa el producto software a construir a través del marco del trabajo. Mientras que el segundo modelo define los conceptos necesarios para realizar la especificación de características de un proceso analítico concreto. Este modelo almacenará la información necesaria para que se pueda ejecutar dicho proceso dentro del sistema completo.

Frente a otras posibilidades, como pueden ser la utilización de lenguajes de modelado de propósito general, la utilización de modelos gráficos (UML) o la definición a través de otro tipo de lenguajes basados en esquemas (XML), se ha planteado una solución en la que se realiza una definición programática del modelo de características de los procesos, facilitada por la utilización del DSL. Esta solución se antoja más práctica y productiva de cara a ser utilizada por los usuarios, ingenieros y científicos de datos, muy cercanos al mundo de la programación, uso de APIs y configuración de las herramientas analíticas utilizadas.

Por otra parte, el conocimiento específico del dominio del problema no se definirá dentro del sistema o a través del DSL, sino que se incorporará como meta-información mediante el uso de ontologías.

3.2.5. Ontologías

La utilización de ontologías dentro del marco de trabajo realizado permite condensar y utilizar todos los conceptos importantes para el dominio del problema de una manera simple. Mediante este tipo de aproximaciones, se ha conseguido habilitar una forma de trabajo para la definición de los procesos analíticos guiada directamente por el modelo de dominio.

Aunque es obligatorio que los expertos del dominio realicen un proceso ontológico previo para crear estas ontologías, esta decisión proporciona una mayor flexibilidad en el marco de trabajo, permitiendo también la reutilización de las mismas entre diferentes productos.

En este planteamiento, la implementación del lenguaje específico de dominio es directamente la que permite hacer uso de estos conceptos de dominio, utilizando sus atributos y comprendiendo sus relaciones, para realizar el mapeo con la información a analizar. Para ello, el sistema proporciona un mecanismo de importación de una lista de ontologías, que deben haber sido generadas mediante el lenguaje de ontologías web (OWL 2.0). Éstas deben estar almacenadas como ficheros independientes siguiendo el formato de representación XML para OWL.

El parseo de las ontologías se realiza mediante la utilización del módulo de Python denominado OWLReady (previamente Ontopy). Esta librería proporciona un conjunto de mecanismos y utilidades para leer y cargar las ontologías como objetos Python. De esta forma, las clases y entidades relativas al dominio se pueden manejar como si se trataran de instancias de clases definidas mediante el propio lenguaje de programación.

Además de realizar ciertas modificaciones en el código fuente de OWLReady, por encima del mismo, se han implementado una serie de mecanismos que permiten enriquecer las clases generadas automáticamente, añadiendo los campos y métodos necesarios, de forma que estos elementos están disponibles para ser utilizados por los usuarios dentro del DSL de definición de procesos.

3.2.6. Interfaces

Los interfaces de programación de aplicaciones (APIs) son elementos fundamentales en la creación de cualquier marco de trabajo de este tipo. Estos interfaces permiten a los usuarios del sistema interactuar y utilizar las características ofrecidas por el mismo.

Para el desarrollo del prototipo se ha fijado la creación de interfaces en tres puntos principales del marco de trabajo:

1. Ejecución del DSL.
2. Acceso a los metadatos.
3. Recorrer el modelo de características.

El primer punto tiene que ver con la necesidad de interpretar el código fuente realizado a través del DSL para generar la especificación de características de los procesos analíticos. La necesidad de realizar la ejecución de los procesos desde un componente externo al marco de trabajo, por ejemplo un componente principal o un script de ejecución, hace imprescindible habilitar un interface específico.

En segundo lugar, una vez ejecutado el DSL y generado el modelo de características, es imprescindible que los componentes independientes y reutilizables del espacio de la solución puedan acceder a los metadatos que están almacenados en dicho modelo. Por poner un ejemplo concreto, si tenemos en cuenta la posible implementación de un componente que lea de un origen de datos que podría ser un fichero distribuido, una tabla de una base de datos, un servicio en la nube, etc. Este componente va a necesitar una serie de meta-información para poder realizar su tarea: la ruta o nombre del fichero en el sistema de ficheros distribuidos, el nombre de la tabla y esquema en la base de datos, la dirección IP y el puerto en el que se encuentra desplegado el servicio para realizar la petición, etc.

Por último, en este contexto se considera también necesario un interface de programación sencillo y genérico para permitir recorrer e iterar todos los componentes del modelo de características de un proceso analítico. Este mecanismo se podría utilizar con diferentes objetivos, siempre relacionados con la transformación del propio modelo, como por ejemplo para realizar una generación automática de código.

3.3. Diseño

3.3.1. Funcionalidades

Desde un punto de vista funcional, el marco de trabajo ofrece una serie de capacidades concretas a los desarrolladores de productos software basados en procesos analíticos, con el objetivo principal de centralizar y estructurar la especificación de características de dichos procesos.

Además de esto, el caso habitual de utilización del marco de trabajo implica que forme parte de la construcción de un sistema software más complejo, en el que lo procesos analíticos sean la parte fundamental, pero que incluya también otra serie de capacidades alrededor de los mismos. Un posible ejemplo es la construcción de una aplicación web que permita ejecutar, monitorizar y finalmente visualizar los resultados de los procesos analíticos. Por tanto, a la hora de fijar el alcance de las funcionalidades ofrecidas por el marco de trabajo, no sólo se han tenido en cuenta las capacidades internas de definición de los procesos, sino también los mecanismos de integración y acceso a la información desde módulos externos.

En este caso, el esfuerzo principal se ha dedicado a plantear y desarrollar estas capacidades, por lo que no se han fijado requisitos no funcionales sobre el marco de trabajo.

En la tabla 3 se puede ver un resumen de las funcionalidades generales del marco de trabajo desarrollado.

1.1	Definición de características de los procesos analíticos
	Permitir la realizar la definición de características de los procesos analíticos y de minería de datos basados en tecnologías big data.
1.2	Incorporación de conceptos del domino en la definición de procesos
	Permitir incorporar en la definición de los procesos los conceptos importantes relativos al dominio del problema y a los datos a analizar.
1.3	Generación del modelo de características de los procesos analíticos
	Permitir generar un modelo abstracto de objetos que almacene de forma estructurada la definición de características de los procesos.
1.4	Acceso a la información contenida en el modelo de características
	Permitir acceder a la meta-información almacenada en el modelo de objetos que describe y contiene todos los pasos de los procesos analíticos.
1.5	Recorrido por la estructura del modelo de características de los procesos
	Permitir recorrer la estructura del modelo de características para aplicar ciertos algoritmos que faciliten las tareas de generación automática.

Tabla 3: Lista funcionalidades generales del framework

Partiendo del primer requisito funcional de la tabla anterior, relativo a la definición concreta de los procesos analíticos, se puede desglosar para llegar a las diferentes capacidades concretas del lenguaje específico de dominio.

Será por tanto, el DSL el que implemente una serie de características cercanas al domino analítico, que van a ser soportadas mediante un mapeo directo al modelo de objetos o características de los procesos.

Estas funcionalidades han sido definidas y acotadas por el alcance del prototipo desarrollado. En este caso, el objetivo es facilitar las labores básicas de análisis descriptivo y exploratorio de los datos, las necesidades de transformación cercanas a la minería de datos y la generación de modelos estadísticos predictivos.

En la tabla 4 se puede ver un resumen de las funcionalidades concretas soportadas por el lenguaje específico de dominio desarrollado dentro del marco de trabajo.

2.1	Incorporación de componentes externos del sistema
	El lenguaje debe permitir la importación de funciones y módulos del sistema implementados de forma externa dentro del espacio de la solución. La definición de operaciones se deben poder mapear directamente a uno de estos componentes independientes.
2.2	Definir propiedades generales de configuración
	Se debe permitir la opción de definir opciones de configuración globales que aplicarán durante todo el proceso y podrán ser consultadas por los componentes específicos.
2.3	Cargar de conceptos del dominio a través de ontologías
	El lenguaje permitirá la incorporación de los conceptos de dominio a través de una serie de ontologías que estarán serializadas y almacenadas en diferentes ficheros. Cada ontología representará un contexto delimitado diferente dentro del dominio de estudio.
2.4	Realizar operaciones básicas con el modelo de dominio
	Se debe permitir la realización de diferentes operaciones básicas con los elementos que representan los conceptos del dominio una vez cargadas las ontologías. En concreto: • Mapeo de contextos. Relacionar conceptos semejantes que se encuentran almacenados en diferentes ontologías.
2.5	Definición de orígenes de datos a partir de los conceptos de dominio
	El lenguaje debe permitir la opción de definir los diferentes orígenes de datos, en los que se encuentra la información a analizar y utilizar durante el proceso, a partir de los conceptos del dominio incorporados.
2.6	Definición de conjuntos de datos a partir de los conceptos de dominio
	El lenguaje debe permitir la opción de definir nuevos conjuntos de datos, partiendo de los conceptos de dominio y sus relaciones. Estos conjuntos de datos deben estar formados por una serie de variables de análisis.
2.7	Definición de variables en los conjuntos de datos
	Se deben permitir diferentes mecanismos de definición de variables para los conjuntos de datos manejados durante el proceso de análisis. En concreto: • Propiedades. Haciendo referencia a relaciones o atributos concretos pertenecientes a la entidad de dominio. • Transformaciones. Realizando algún tipo de operación de transformación de otras variables definidas en el conjunto de datos.
2.8	Realizar operaciones básicas con los conjuntos de datos
	El lenguaje debe soportar la definición de una serie de operaciones básicas sobre los conjuntos de datos definidos en el proceso analítico. En concreto:

	• Filtrado. Operación para eliminar registros del conjunto de datos en base al valor de alguna de las variables. • Unión. Operación para realizar la unión de los registros de dos conjuntos de datos a partir de la relación de sus entidades de dominio. • Particionado. Operación para dividir los registros de un conjunto de datos en dos, generando dos nuevos conjuntos de datos. • Secuencia. Operación para definir una secuencia de operaciones de transformación sobre las variables del conjunto de datos.
2.9	Generación de modelos estadísticos predictivos
	El lenguaje debe soportar la definición de la operación de generación de modelos estadísticos predictivos a partir de un conjunto de datos que contenga una variable objetivo. Se deben poder establecer diferentes parámetros o argumentos dinámicos que serán utilizados por el componente específico encargado de crear y entrenar el modelo.
2.10	Predicción de valores de variable objetivo a través de un modelo
	Se debe permitir definir la operación de predicción o scoring de la variable objetivo partiendo de un conjunto de datos y mediante la aplicación de un modelo predictivo definido durante el proceso analítico.

Tabla 4: Lista funcionalidades concretas del DSL de definición de procesos

3.3.2. Arquitectura

El diseño arquitectónico realizado se debe ver también desde dos perspectivas diferenciadas. En primer lugar, la arquitectura general de la solución engloba todos los componentes que intervienen en el desarrollo de los productos analíticos y que hacen uso del marco de trabajo, así como, las relaciones que tienen los elementos incluidos en el marco de trabajo con los componentes desarrollados específicamente en el espacio de la solución. En segundo lugar, se encuentra la arquitectura interna planteada en el desarrollo del prototipo del marco de trabajo.

Dentro de la arquitectura general de la solución, el sistema en construcción hace uso de los componentes reutilizables implementados dentro del espacio de la solución, y además de esto, el sistema contiene los DSL concretos realizados, que definen cada uno de los procesos analíticos que componen el producto completo. El lenguaje específico de dominio utilizado enlaza directamente con el marco de trabajo y es el que puede hacer referencia también a los componentes independientes que implementan cada tarea del proceso.

Por otra parte, dentro del marco de trabajo se encuentra el modelo de objetos general, que contiene la especificación de características de los procesos analíticos y soporta la implementación del lenguaje específico de dominio. Una instancia concreta de este modelo se generará durante la interpretación de cada uno de los DSLs incluidos en el sistema.

El acceso a la información contenida en el modelo de características se habilita mediante los interfaces de programación incluidos también dentro del marco de trabajo. Los componentes del sistema harán uso de estos interfaces para consultar los meta-datos necesarios para llevar a cabo cada tarea específica.

En la figura 22 se puede ver un diagrama simple que refleja la arquitectura general seguida en el desarrollo de los productos software a través de la utilización y el funcionamiento del marco de trabajo propuesto.

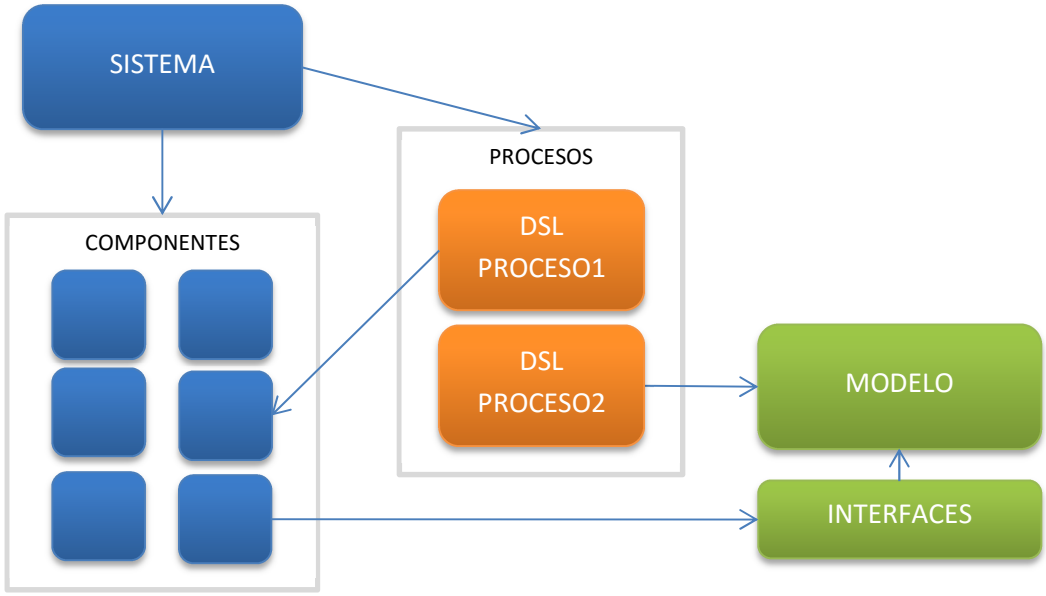

Figura 22: Arquitectura general y funcionamiento del marco de trabajo

Como se puede apreciar, el diseño del marco de trabajo y del DSL no se ha realizado para intervenir únicamente durante la fase de desarrollo y definición de los procesos analíticos, sino que también guían la arquitectura del sistema en construcción, la ejecución de los propios procesos y en cierta medida la forma de implementación de los componentes del producto. En este sentido, el diseño del marco de trabajo se ha realizado con el objetivo de poder facilitar la utilización de un procedimiento similar a la inversión de control.

La inversión de control es un principio de diseño de software en el que cada uno de los componentes implementados dentro del sistema recibe el flujo de ejecución desde el marco de trabajo utilizado. En las soluciones cuya arquitectura sigue este tipo de diseño, el control de ejecución se encuentra invertido si se compara con la programación procedimental tradicional.

Es decir, en lugar de ser los componentes reutilizables los que se vayan invocando entre sí de manera enlazada y con puntos de interacción con el marco de trabajo, es el propio framework el que tiene el control y realiza las llamadas a los componentes independientes para realizar las tareas incluidas en el proceso.

En las figuras 23 y 24 se puede ver un ejemplo gráfico de esta comparativa de flujos de control entre una arquitectura de componentes procedimental tradicional y la arquitectura de componentes con inversión de control utilizada.

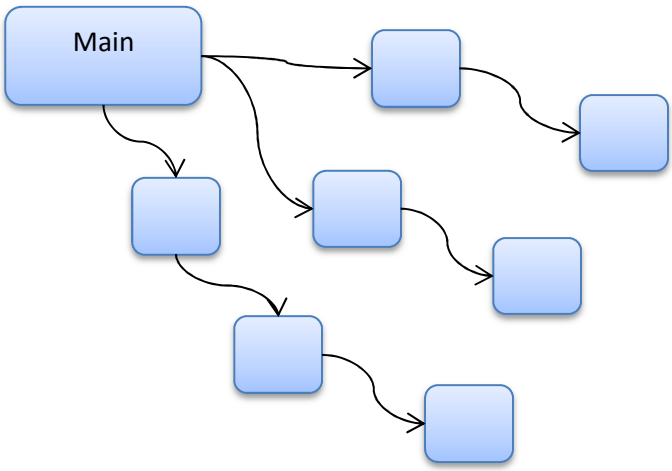

Figura 23: Arquitectura procedimental tradicional de componentes

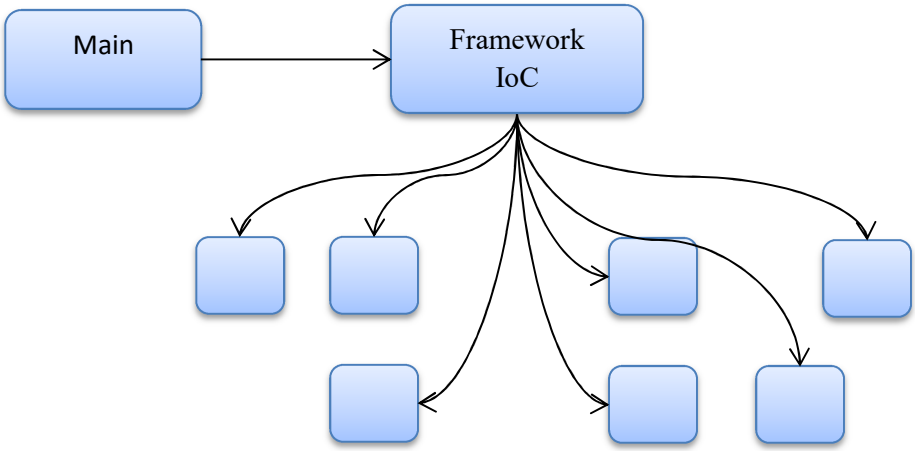

Figura 24: Arquitectura de componentes con Inversión de Control

El marco de trabajo soporta dentro del DSL la importación del interface de entrada a cada componente independiente, permitiendo definir su relación con las tareas correspondientes del proceso. En otras palabras, para cada tarea concreta se puede definir qué componente es el encargado de llevarla a cabo (donde está implementada).

De esta forma, la generación del modelo de características mediante la interpretación del DSL se puede utilizar también para invertir el control y dirigir la ejecución de los procesos.

La arquitectura interna del marco de trabajo desarrollado como parte del prototipo es bastante simple. El sistema se ha estructurado en diferentes módulos, siguiendo los convenios de desarrollo habituales en Python y realizando una separación lógica de responsabilidades a la hora de implementar cada una de las funcionalidades.

En la figura 25 se puede ver un diagrama de componentes básico en el que se representan los módulos principales del marco de trabajo y sus relaciones de dependencia.

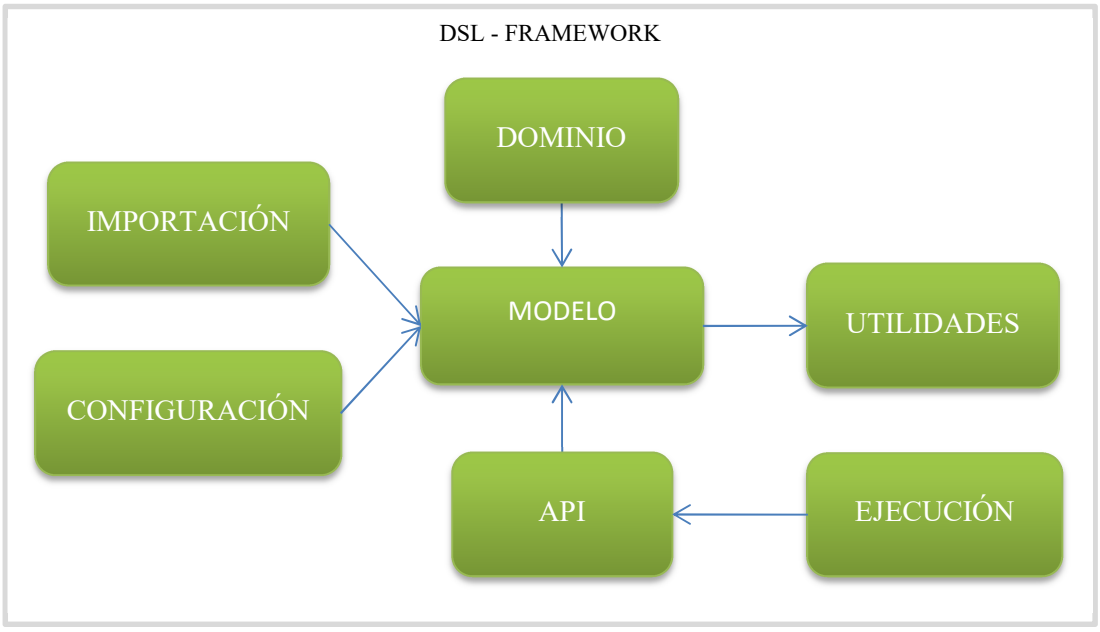

Figura 25: Arquitectura interna y módulos del marco de trabajo

3.3.3. Modelo de dominio

Una vez analizado el diseño funcional y arquitectónico de la solución, en este apartado se describe el modelo de dominio de características para la definición de los procesos analíticos.

Este es un modelo conceptual genérico que representa los principales elementos que componen el dominio de la solución y que sirve de base para el diseño del marco de trabajo. A partir de este modelo, en la fase de desarrollo se ha construido el modelo de clases y objetos, adaptado a las necesidades concretas de implementación, que soporta la definición de cada una de las características que están incluidas en un proceso.

En la figura 26 se puede ver un diagrama UML de clases que permite visualizar y entender de mejor forma la estructura de este modelo de dominio para la definición de características.

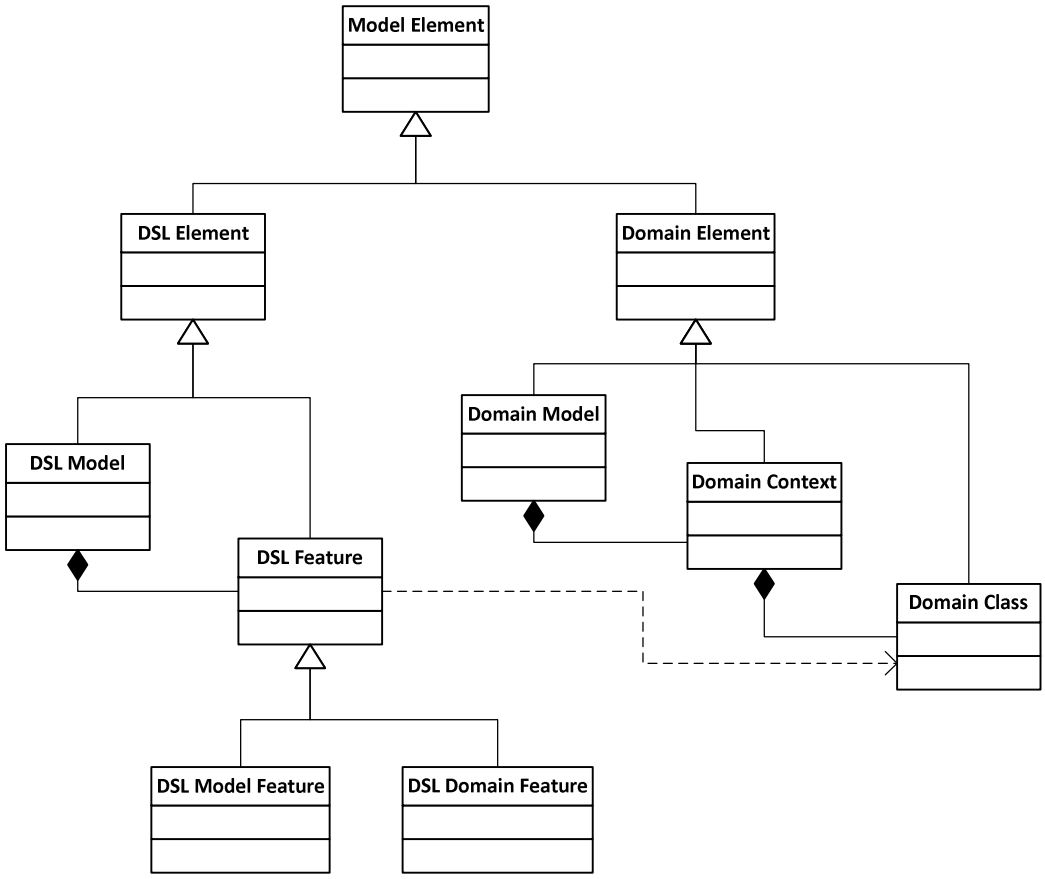

Figura 26: Diagrama de componentes del modelo de dominio

En primer término se encuentra la clase padre de la que extenderán el resto y que define un elemento del modelo. Estos elementos de diferencian entre los elementos propios del DSL y los elementos relacionados con el dominio del problema, formando lo que se ha denominado como el modelo del lenguaje y el modelo del dominio.

El modelo de dominio está compuesto por una serie de elementos que se identifican con los diferentes contextos (importado cada uno a través de una ontología) y que contienen a su vez una serie de clases del dominio, que hacen referencia a cada uno de los conceptos concretos.

Por su parte, el modelo del DSL está compuesto por un conjunto de características ("*features*") que dependen y tienen siempre una relación directa con la clase de dominio a la que hacen referencia. Estas características, que hacen referencia a las funcionalidades, operaciones y tareas del proceso definidas dentro del DSL, se dividen a su vez entre las que se basan en los elementos del dominio y las que hacen referencia a los elementos del lenguaje.

De estos últimos componentes heredarán la mayoría de las operaciones concretas soportadas.

3.3.4. Estructura del DSL

Tras la descripción del modelo genérico de características, a continuación se completará el diseño realizado con una descripción de las decisiones alcanzadas en lo que respecta a la estructura del lenguaje específico de dominio planteado en esta solución.

Las tres causas principales que definen la estructura seguida en la especificación del lenguaje son las siguientes:

1. Se ha concebido un DSL en forma de lenguaje embebido utilizando como base Python, por lo tanto, su estructura está muy influenciada por las características de este lenguaje de programación.
2. El DSL se diseñado además para utilizar directamente los propios objetos del modelo de definición de características de los procesos. Esto hace que la estructura del DSL esté muy acoplada a este modelo de objetos.
3. A esto se le debe sumar la enorme influencia que significa el incorporar los conceptos del dominio como elementos a utilizar dentro del propio lenguaje.

Estos motivos hacen que la estructura y forma de utilizar el DSL sea muy cercana a la programación con cualquier otro lenguaje orientado a objetos. La mayoría de las sentencias se refieren a la creación de uno de los objetos del modelo o la invocación de métodos concretos de una clase o instancia. Además de esto, como ya se ha comentado, se podrán utilizar todas las características y funcionalidades inherentes al uso de Python.

Para poder utilizar el DSL simplemente se necesita conocer la estructura completa del modelo de dominio para la especificación de características implementado que se encuentra por debajo, junto con las operaciones permitidas, algunos convenios simples y funciones específicas. Dentro de los convenios básicos implícitos al uso del DSL, destacan las siguientes tres características:

- La definición de una nueva operación o tarea concreta del proceso analítico se debe realizar siempre y hace siempre referencia a un elemento del modelo de dominio del problema, ya sea de forma directa o por herencia de su entidad padre. Por ejemplo, para definir un nuevo conjunto de datos se deben enlazar dichos datos con la clase que define “qué son esos datos”, es decir, si son datos referentes a “Ventas”, datos referentes a “Productos”, etc. Si posteriormente se define una operación de filtrado sobre un conjunto de datos que hace referencia, por ejemplo a “Transferencias”, la nueva operación de filtrado también hará referencia implícitamente a dicho concepto.

- La especificación de una característica del proceso genera siempre un objeto concreto del modelo de dominio de la solución, que se podrá utilizar de forma enlazada para realizar la definición de la siguiente operación del proceso. Es decir, la operación de definición de un origen de datos genera un objeto de tipo “Datasource” que permite invocar directamente a las operaciones soportadas sobre dicho conjunto de datos.

- La parametrización de las operaciones definidas, por regla general, se podrá realizar de tres formas: mediante el uso de argumentos concretos, uno que define el tipo y otro el valor a utilizar, mediante el uso de un diccionario, o una lista de objetos. En el primer caso, ambos argumentos serán de tipo cadena de caracteres, pudiendo contener cualquier tipo de valores o expresiones complejas, según las necesidades de implementación de los componentes de la solución. En el segundo caso, el objeto de tipo diccionario es un tipo de dato de Python similar a un mapa clave/valor, donde ambos elementos serán también cadenas de caracteres.

Para clarificar estos conceptos sobre la estructura del lenguaje, a continuación se proporcionan algunos ejemplos concretos de funcionalidades junto con el formato de definición dentro del DSL.

Por ejemplo, en lo que se refiere a las propiedades de configuración generales del proceso, el lenguaje proporciona una sentencia específica para indicar los pares clave – valor que se almacenarán dentro del módulo de configuración. El valor puede ser un objeto de cualquier tipo.

```
config.property("clave", "valor")
```

De un modo similar, la importación de ontologías que contienen los conceptos relativos al dominio del problema, se puede realizar mediante la siguiente sentencia.

```
domain.load(["contexto1.owl", "contexto2.owl"])
```

Además de la lista de nombres, esta operación permite también especificar como parámetro la ruta al directorio del sistema de ficheros en el que se encuentran las ontologías.

Si por ejemplo, una de estas ontologías contiene una clase denominada “Cuentas” que hace referencia a un concepto específico del dominio, a través de la siguiente sentencia del lenguaje se podría definir el origen de datos en el que se encuentra la información relativa a las cuentas que se quiere analizar.

```
domain.Cuentas.datasource("file", "/files/cuentas.dat")
```

Como se puede apreciar, la propia clase “Cuentas” es un elemento válido del lenguaje y es ésta la que sirve como guía para la especificación de las características del proceso.

En el caso anterior, el primer parámetro define el tipo del origen de datos y el segundo es el valor concreto que identifica el elemento deseado. El DSL no define ni limita el formato de estos valores, que serán interpretados y utilizados por el componente encargado de implementar la funcionalidad de lectura de los datos en base a la plataforma específica utilizada.

```
domain.Cuentas.datasource("table",
"Driver=ODBC;User=*****;Password=*****;Database=db;Table=cuentas")
      .impl(database_module.custom_odbc_datasource)
```

Por ejemplo, la sentencia anterior realiza la especificación de un origen de datos similar al anterior, pero utilizando otro tipo de formato similar a una cadena de conexión ODBC sobre una base de datos. Un punto también importante en este segundo ejemplo es la utilización de la operación "impl" directamente sobre el "datasource" recién definido. Esta operación permite hacer referencia al módulo y componente concreto del producto que realiza la implementación de esta característica.

La generación de un conjunto de datos a partir de la realización de transformaciones sobre los datos originales es también una tarea habitual en este tipo de procesos analíticos. A continuación se muestra un ejemplo de sentencia de definición de un "dataset" sobre la clase "Cuentas" anterior.

```
domain.Cuentas.dataset("dataset_name", [
      domain.Cuentas.property("id"),
      domain.Cuentas.property("saldo").transform(function).variable("s")
])
```

En este caso, el primer argumento de la definición se corresponde con el nombre único o identificador del conjunto de datos, mientras que el segundo es una lista de transformaciones. La primera sentencia selecciona simplemente la propiedad o atributo "id" de la cuenta, mientras que la segunda utiliza la propiedad "saldo" para realizar una transformación mediante la función denominada "function". Esta transformación genera una nueva "variable" que se ha denominado "s".

Como se puede apreciar, tanto la lectura como la comprensión del DSL son muy sencillas. La estructura del lenguaje es bastante simple, lo que sumado a la incorporación de los conceptos del dominio y la sintaxis minimalista de Python, hacen que su utilización sea muy ágil y directa.

3.4. Desarrollo

3.4.1. Fases

El desarrollo del prototipo se ha realizado siguiendo un proceso iterativo simple separado en tres fases diferentes. El objetivo ha sido implementar en cada fase una de las partes generales del marco de trabajo: el modelo de dominio, el lenguaje específico de dominio y los interfaces de aplicación; basándose cada uno en los elementos generados durante la fase anterior.

En la primera fase se ha construido el modelo de objetos que permitirá especificar las características concretas de los procesos analíticos definidos. Este modelo está compuesto por una estructura jerárquica de clases, siguiendo el diseño realizado para el modelo de dominio, junto a los campos que permiten almacenar la información y parámetros relativos a cada tarea. Además de esto, se ha implementado todo el soporte para incorporar al modelo todos los conceptos de dominio. Esto último implica soportar la lectura y parseo de las ontologías.

Durante la segunda fase, se ha realizado la implementación del lenguaje específico de dominio, a partir del modelo de objetos anterior. Se han incorporado a cada clase los métodos necesarios para poder crear y relacionar los diferentes objetos del modelo. La implementación del DSL incluye la sobrecarga de algunos operadores y métodos avanzados para facilitar y agilizar la definición de los procesos analíticos.

Por último, en la tercera fase se han implementado los interfaces de programación de aplicaciones para el acceso y consulta de la información contenida en el modelo de dominio. Esto incluye todas las características necesarias para la aplicación del DSL y la estructura del patrón de diseño Visitor para recorrer y aplicar un algoritmo a los objetos del modelo.

3.4.2. Módulos

La implementación del marco de trabajo se ha realizado estructurando y agrupando los componentes en diferentes módulos, según sus características y funcionalidades comunes. Todos los módulos Python desarrollados se encuentran dentro del paquete denominado “unedtfm”.

A continuación se proporciona un listado de los módulos que componen el marco de trabajo junto con una breve descripción.

api
Contiene los mecanismos relativos a los interfaces de programación de aplicaciones que permiten el acceso a los componentes del modelo de dominio generado tras la utilización de un DSL.
config
Implementa las capacidades para dar soporte a la definición de una configuración global de los procesos analíticos, permitiendo establecer propiedades y valores que podrán ser utilizados durante las diferentes tareas.
domain
Incluye los componentes necesarios para lectura e importación de las ontologías que contienen los conceptos del dominio y que serán generados como elementos que se puedan utilizar a través del lenguaje específico de dominio.
exec
Contiene diferentes clases y funciones que permiten y facilitan la ejecución de los procesos a través del lenguaje específico de dominio, haciendo uso del resto de funcionalidades incluidas dentro del marco de trabajo.
expr
Implementación de algunas de las expresiones y componentes de utilidad para el manejo de expresiones dentro del lenguaje específico de dominio.

import
Incluye diferentes funciones de utilidad para la importación o integración del marco de trabajo dentro de los productos software que basan sus funcionalidades en los procesos analíticos definidos a través del DSL.
model
Contiene todas las clases que forman el modelo de dominio del marco de trabajo para la definición de características de los procesos analíticos y que soportan la implementación del lenguaje específico de dominio y de los interfaces de programación de aplicaciones.
util
Incluye una serie de clases y funciones de utilidad genéricas que son utilizadas en la implementación del resto de módulos del marco de trabajo.
visitor
Implementación siguiendo la estructura del patrón de diseño visitor que proporciona los mecanismos y clases base para realizar el recorrido por todos los elementos del modelo de dominio y aplicar sobre ellos las operaciones o algoritmos específicos.

Tabla 5: Listado de módulos que componen el marco de trabajo

La estructura de estos módulos sigue la arquitectura especificada durante la fase de diseño del marco de trabajo y sus dependencias no varían respecto al diagrama general incluido en la figura 25.

3.4.3. Clases

En el apartado de diseño se ha descrito en profundidad el modelo de dominio y sus principales características. Se han podido ver los principales elementos que constituyen este modelo y que dan soporte a la implementación del lenguaje específico de dominio, de los interfaces de programación de aplicaciones, y en general, del marco de trabajo desarrollado.

Partiendo de este diseño de elementos y de la especificación de funcionalidades a incluir, la implementación del prototipo se ha realizado aplicando todos los paradigmas de desarrollo descritos, relativos a la ingeniería de dominio y el diseño dirigido por modelos, entre otros, pero intentando siempre minimizar todo lo posible su complejidad.

A continuación se proporciona un listado resumido con las principales clases de las que está compuesto el modelo de dominio del prototipo de marco de trabajo desarrollado. Para evitar un listado excesivamente extenso, se han incluido únicamente aquellas clases más relevantes y con un significado dentro del dominio, filtrando otras clases más relacionadas con las necesidades concretas de implementación.

domain_model
Representa la meta-información del modelo de dominio completo
domain_ontology
Representa un contexto del dominio a través de una ontología
domain_class
Representa una clase o concepto del modelo de dominio
datasource
Representa la definición de un origen de datos para una clase del dominio
dataset
Representa la definición de un conjunto de datos de una clase del dominio
dataset_transformation
Representa una operación de transformación sobre un conjunto de datos
dataset_filter
Representa una operación de filtrado de registros de un conjunto de datos
dataset_join
Representa una operación de unión de dos conjuntos de datos
dataset_partition
Representa una operación de particionado de un conjunto de datos
dataset_partition_item
Representa un elemento de la partición de un conjunto de datos
dataset_chain
Representa una secuencia de operaciones sobre un conjunto de datos
dataset_variable
Representa una variable definida en un conjunto de datos
dataset_operation
Representa una operación sobre las variables de un conjunto de datos
property
Representa una propiedad de una clase del dominio
property_variable
Representa una propiedad de una clase de dominio dentro de un conjunto de datos
transform
Representa una operación de transformación sobre una variable
dsl_function
Representa la implementación de un operación de transformación
model
Representa la definición de la tarea de generación de un modelo predictivo
model_prediction
Representa la operación de predicción mediante un modelo predictivo
dsl_configuration
Representa la configuración general de un proceso
dsl_configuration_property
Representa una propiedad de configuración
dsl_visitor
Representa un visitor que puede recorrer los elementos del dominio

Tabla 6: Listado de las principales clases que componen el marco de trabajo

Si se atiende a la implementación concreta de las clases se puede ver cómo todas extienden una serie de clases padre comunes, que sirven para describir algunos comportamientos y campos comunes a todas ellas.

La composición, vínculo y dependencia de unas clases con otras queda perfectamente definida según la estructura utilizada en el propio DSL. En la figura 27 se incluye un diagrama de clases UML que muestra las relaciones entre las clases descritas anteriormente.

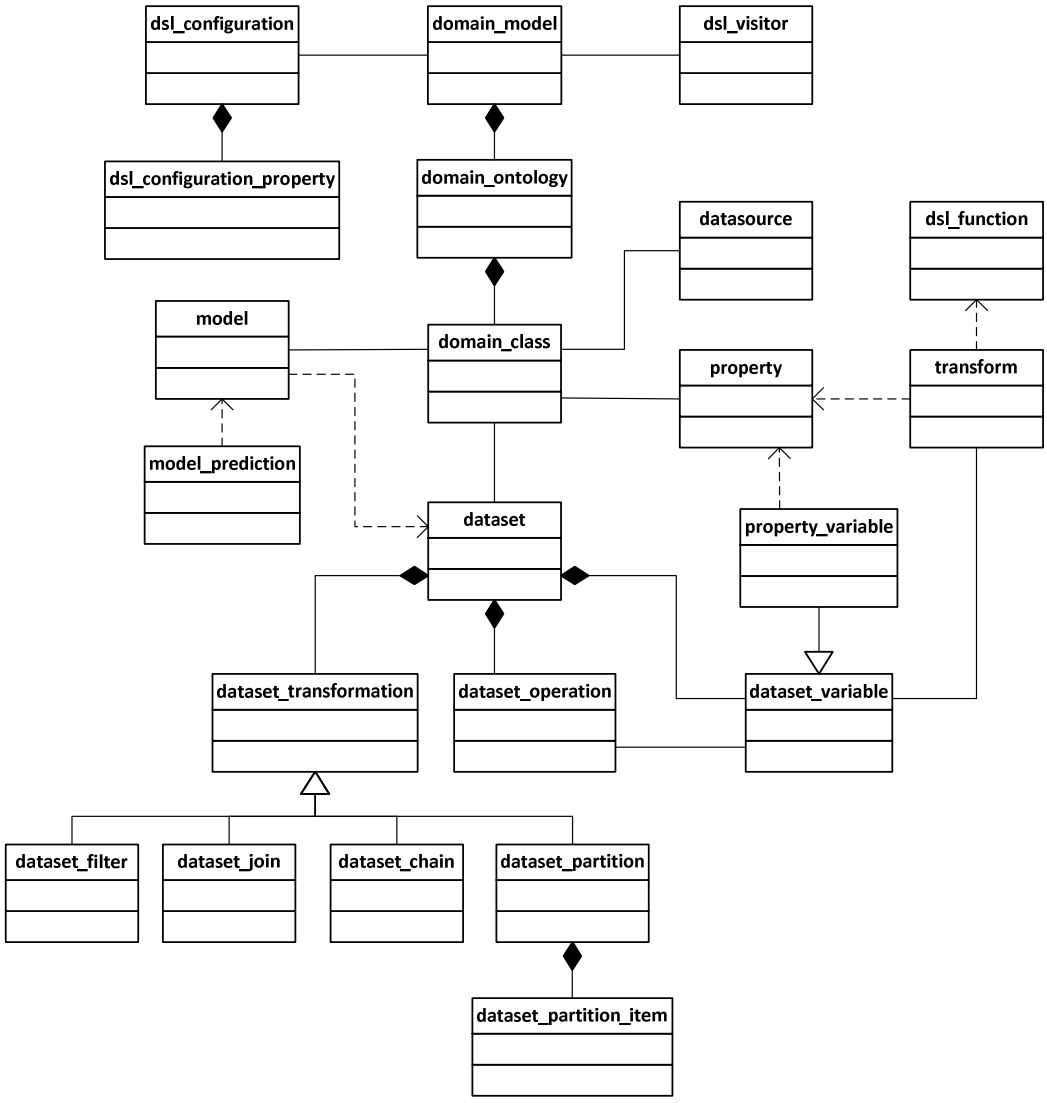

Figura 27: Diagrama de clases del modelo de dominio

Como es de esperar y a la vista del diagrama, las clases del dominio son la pieza central del dominio de características, puesto que guían toda la definición de los procesos. Del mismo modo, los conjuntos de datos son el elemento principal para la definición de las operaciones y transformaciones.

3.4.4. Interfaces

En lo que respecta a los interfaces de programación de aplicaciones desarrollados dentro del prototipo del marco de trabajo, se encuentra en primer lugar el módulo API, que permite el acceso al modelo de dominio generado tras la ejecución del DSL. En la implementación realizada, este módulo es bastante simple y delega las operaciones de acceso en las propias clases del modelo a través de variables estáticas globales.

Por tanto, serán directamente los elementos del modelo de dominio los que proporcionen el acceso a la definición de características de los procesos analíticos. A través por ejemplo, de los campos públicos de las clases, que contienen los valores establecidos en la interpretación del lenguaje, así como las listas y diccionarios para el almacenamiento de las relaciones entre elementos y las estructuras de datos más complejas. Del mismo modo, se pueden encontrar diferentes métodos de utilidad en las clases del modelo que permiten un acceso sencillo a la información.

Esta implementación tan simple está diseñada para no incorporar mayores complejidades en la implementación base del marco de trabajo. Al exponer el propio modelo se evita a los usuarios la necesidad de manejar dos APIs diferentes: una para el uso del DSL y otra para el uso del API. Esto permite además, habilitar un mayor número de puntos de acceso y extensión de las características actualmente desarrolladas.

El siguiente fragmento de código Python puede servir para ilustrar la implementación realizada y la forma de utilización de este interface de programación. En este caso, se hace uso de la sintaxis específica de comprensión de listas para recorrer y mostrar una serie de valores.

```
[print(v.definition.name)
for v in api.domain.get_model().get_ontology('test').get_class('Test')
.get_dataset('test_dataset').variables]
```

El objetivo es acceder a los nombres de las variables de un conjunto de datos concreto denominado “test_dataset”. Para ello, a partir del API se recupera el modelo de dominio y de éste la clase denominada “Test”, que se encuentra dentro de la ontología “test”. De cada una de las variables, se accede a un definición concreta (“definition”) y dentro de esta estructura se mostrará su nombre (“name”). Otros ejemplos serían muy similares, pero haciendo uso de otros elementos y métodos diferentes.

Como se puede apreciar, el interface de programación para el acceso a la información contenida en el modelo implica simplemente conocer la estructura de objetos y los métodos de consulta proporcionados. Estas capacidades del marco de trabajo se pueden y deben utilizar desde los componentes del producto software a construir para ejecutar las tareas concretas de los procesos analíticos.

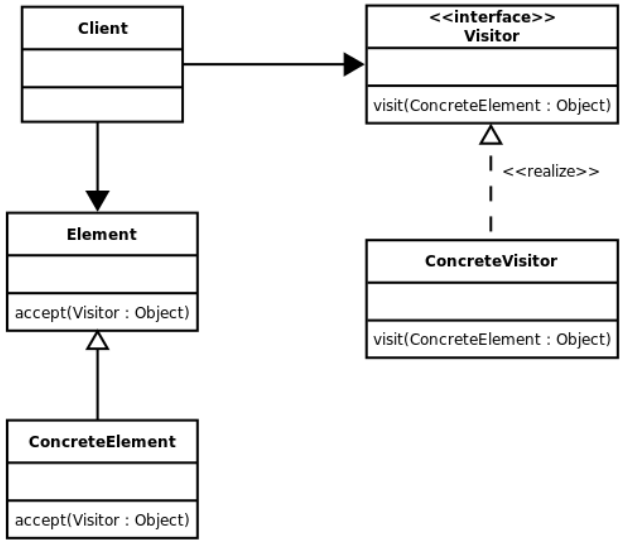

Figura 28: Estructura del patrón de diseño Visitor

Además de este API, dentro del marco de trabajo se ha desarrollado también una funcionalidad específica para permitir la implementación de algoritmos que recorran toda la estructura de objetos del modelo de dominio. Esto permite realizar transformaciones del modelo o generar código automáticamente de una forma directa. Como ya se ha comentado, esta característica se ha construido haciendo uso del patrón de diseño Visitor, siguiendo la estructura mostrada en la figura 28.

El patrón de diseño Visitor se ha considerado el más apropiado en este caso, puesto que su objetivo es la definición de una operación sobre los objetos de una jerarquía de clases sin necesidad de la modificación de estos objetos.

Siguiendo la estructura del patrón de diseño, los elementos del modelo deben definir un método que “acepta la visita”, es decir, que recibe como parámetro un objeto que implementa el interface visitor. La implementación del método realiza a su vez la llamada al método concreto “visit” del visitor, pasando como parámetro el propio elemento. Además, para cada uno de sus elementos hijos o elementos relacionados se invocará a sus métodos “accept” pasando nuevamente el visitor como parámetro. De esta forma, el visitor consigue realizar un recorrido completo por toda la estructura de elementos.

El desarrollo realizado en el marco de trabajo sigue esta forma de proceder e incorpora mayores funcionalidades para los usuarios, aprovechando las características propias del lenguaje Python. En este caso, al no existir los interfaces como en otros lenguajes, se ha realizado una clase abstracta denominada “dsl_visitor” que es la aceptada por todos los objetos del modelo de dominio. Esta clase define una serie de métodos “visit” diferentes por cada tipo de elemento. La distinción se realiza utilizando los decoradores de Python, que son funciones o clases que permiten anotar otras funciones, de forma que se puede ejecutar un código concreto ante la invocación a la función anotada.

Esta implementación permite una gran flexibilidad a la hora de utilizar el interface de programación para construir un nuevo visitor, ofreciendo la posibilidad de sobrescribir únicamente aquellos métodos que sean de interés para la operación a realizar, por ejemplo: si únicamente se quieren tratar ciertos elementos. Además de esto, el decorador implementado recibe como parámetro el tipo de objeto o el nombre del tipo de elemento que visita la función anotada. De esta forma, puede realizar una búsqueda recursiva para encontrar un método válido que implemente la operación para cada elemento concreto. Esta búsqueda se realiza en dos sentidos: sobre la jerarquía de clases del modelo de dominio y sobre una posible jerarquía de clases visitor.

```
@visitor(domain_ontology)
def visit(self, ontology):
    if self.logging:
        print("Domain Ontology: %s " % ontology.name)

@visitor(domain_class)
def visit(self, domain_class):
    if self.logging:
        print("BaseClass: %s" % domain_class.__name__)
```

Este comportamiento se puede ilustrar de mejor forma con un ejemplo. Dentro de la jerarquía del modelo de dominio nos fijaremos en las clases “dataset_transformation” y en sus clases hijas “dataset_filter” y “dataset_join”. Se podría implementar un visitor específico que visite cualquier transformación (“transformation”) de un conjunto de datos y otro visitor hijo que implemente una operación específica sobre una tarea de filtrado (“filter”). Tras aceptar la visita un elemento de alguno de estos tipos (filter o join), el decorador buscará primero si el visitor posee un método para visitar ese tipo concreto. En caso de no existir, se buscaría recursivamente si existe algún método para visitar alguna de sus clases padre (transformation). Si no existe ningún método para manejar la jerarquía de elementos, entonces se buscaría en las clases base del visitor, es decir en el visitor padre.

El objetivo de esta implementación es facilitar de forma transparente la utilización del interface visitor dentro del marco de trabajo, de forma que los usuarios sólo necesiten implementar el mínimo código posible para realizar las operaciones o tareas requeridas.

3.5. Aplicación

Existen diferentes modalidades de aplicación de la solución propuesta, a través de la utilización de todos los componentes del marco de trabajo descritos con anterioridad.

El diseño e implementación seguidos facilitan las siguientes formas de aplicación dentro del desarrollo de un producto software, donde las dos primeras constituyen la base de utilización del marco de trabajo y las tres últimas se refieren a la integración con el resto de componentes.

1. Utilizar el lenguaje específico de dominio para realizar la descripción de un proceso analítico.
2. Interpretar el DSL implementado para obtener el modelo de características del proceso.
3. Incluir dentro del DSL construido las referencias a los componentes que implementan cada característica, para poder aprovechar así la interpretación del DSL y permitir ejecutar el propio proceso descrito.
4. Implementar una pieza o componente específico que utilice el API del marco de trabajo para acceder a la información del modelo generado y realice las operaciones oportunas.
5. Implementar un componente de tipo visitor que recorra el modelo generado tras la interpretación del DSL y aplique algún tipo de transformación o algoritmo al recorrer los elementos.

Para profundizar en las diferentes formas de aplicación, en este apartado se describe la ejecución o interpretación del DSL dentro del marco de trabajo y las dependencias necesarias para poder realizar esta tarea.

Además de esto, se proporciona un ejemplo de un caso de uso completo para la aplicación del marco de trabajo. Este ejemplo se puede extrapolar a la mayoría de los problemas reales incluidos dentro del alcance de la solución.

3.5.1. Ejecución

La integración del marco de trabajo dentro del desarrollo de un producto software requiere que la descripción de características de los procesos analíticos sea realizada mediante la utilización del lenguaje específico de dominio y que estos fragmentos del DSL se puedan compilar o interpretar y ejecutar para generar el modelo de objeto que almacena toda la información especificada.

En este caso, no es necesario conocer unos mecanismos específicos de compilación y ejecución, debido a que el DSL se ha implementado como un lenguaje embebido dentro de Python. Las propias características y funcionalidades ofrecidas por el lenguaje Python se pueden utilizar para integrar la ejecución del DSL dentro de los productos software. En realidad, esta tarea es casi directa si para el desarrollo del producto también se utiliza Python como lenguaje de programación. En otro caso, será necesario un componente que actúe de puente para permitir la ejecución de código Python.

La forma más sencilla para indicar al motor de ejecución (“runtime”) de Python que debe interpretar un DSL es mediante la importación del fichero que contiene el código. En el ejemplo siguiente se importa el fichero “dsl.py” que se encuentra dentro del módulo “test”.

```
import test.dsl
```

Si no se quiere establecer una dependencia directa con el código del DSL de definición, Python ofrece también la posibilidad de leer, compilar y ejecutar el lenguaje de forma dinámica, mediante el uso de las funciones compile() y exec(). El ejemplo siguiente permite realizar estas acciones para el mismo fichero anterior.

```
with open("dsl.py") as f:
    dsl = compile(f.read(), "dsl.py", 'exec')
    exec(dsl, globals(), locals())
```

Una vez ejecutado el código del DSL y generado el modelo de características del proceso analítico, éste se podría operar o transformar mediante la utilización, por ejemplo, de un visitor implementado específicamente.

```
v = dsl_visitor()
domain.accept(v)
```

Finalmente, el acceso al modelo de dominio se encuentra siempre disponible también a través de todas las características del API. Los métodos de consulta se pueden utilizar directamente desde Python para obtener la información necesaria.

3.5.2. Dependencias

Para poder realizar la ejecución del DSL e integrar por tanto el marco de trabajo dentro de un producto software son necesarias algunas dependencias concretas. Obviamente, la primera necesidad es la instalación del entorno de ejecución de Python. Este entorno contiene el compilador e intérprete de Python, así como los principales módulos y librerías estándar incluidas dentro del lenguaje de programación.

En segundo lugar, puede ser necesario establecer algún tipo de variable de entorno o variable de configuración para permitir que Python sepa incorporar las dependencias al motor de ejecución. Por ejemplo, la variable de entorno PYTHONPATH se puede utilizar para definir con las rutas a los directorios que contienen los módulos del marco de trabajo y el resto de librerías o componentes independientes utilizados. El siguiente ejemplo muestra una posible configuración de esta variable de entorno bajo un sistema Windows.

```
set PYTHONPATH=%cd%
set PYTHONPATH=%PYTHONPATH%;C:\unedtfm
set PYTHONPATH=%PYTHONPATH%;C:\python\lib\site-packages
```

El marco de trabajo desarrollado únicamente tiene dependencia con un módulo externo de terceros, nos referimos a la librería OWLready para el manejo de ontologías OWL. Este módulo debe ser instalado y accesible desde el entorno de ejecución. Para conseguir esto se puede utilizar directamente “pip”, un sistema de gestión de paquetes para Python que mantiene un índice de paquetes (“PyPI”) entre los que se encuentra la versión 0.2 de OWLready, que es la utilizada para el desarrollo del prototipo.

3.5.3. Caso de uso

De cara a completar la forma de aplicación y proporcionar una visión completa de la solución, se ha realizado un caso de uso completo, que incluye el desarrollo de un producto software analítico muy sencillo, pero en el que se han seguido todas las decisiones, metodologías y herramientas actuales necesarias para ajustarse lo más posible a la realidad.

A pesar de la simulación del problema y de la información disponible, este ejemplo permite demostrar la utilización del marco de trabajo en el entorno de un proyecto realista, para poder comprobar su forma de integración dentro de la metodología de desarrollo de los procesos analíticos y verificar así los beneficios obtenidos de su aplicación.

El objetivo principal del caso de uso es, por tanto, la construcción de un producto software basado en la realización de un proceso analítico y que se pueda aplicar al negocio de una compañía. Este proceso se debe diseñar en torno al análisis y transformación de la información histórica disponible, para la generación de un modelo estadístico y su posterior utilización con el fin de extraer un beneficio de negocio.

En este ejemplo, se ha seleccionado un problema de negocio genérico en el que el software se va a utilizar en la posible recomendación del mejor producto disponible a un cliente, a partir de la información que la empresa dispone de éste. El producto recomendado debería ser el que más le puede interesar y por lo tanto el que mayor probabilidad de compra tiene. Para intentar construir el modelo, se desea explotar la información transaccional disponible relativa a las ventas de productos que los clientes han adquirido con anterioridad.

A la hora de definir el dominio del problema se ha intentado buscar un ejemplo que se adapte a este caso de uso y que pueda ser al mismo tiempo lo más simple y entendible posible. Por esto, se ha decidido utilizar un ejemplo académico clásico cómo es la venta de pizzas. De esta forma, la compañía que encarga el desarrollo del sistema podría ser una cadena de pizzerías, que ofrece la posibilidad a los clientes registrados de realizar pedidos online a través de su página web. Estos pedidos online serán la principal fuente de información disponible.

El resultado esperado del modelo estadístico será la recomendación de un tipo de pizza concreto para cada cliente. Estas recomendaciones se podrían explotar posteriormente dentro de la estrategia de marketing online de la compañía, a través de una serie de campañas o acciones de retargeting personalizadas en redes sociales. Por ejemplo, si se detecta que un cliente ha accedido a la web de la compañía, pero no ha formalizado ningún pedido, se pujará por los anuncios que vea posteriormente al navegar por los medios sociales y se le mostrará una oferta o descuento en base a su pizza recomendada.

Los principales conceptos del dominio que maneja una compañía de este tipo son bastante habituales y muy fácilmente entendibles. Por una parte, se encuentra todo el conocimiento relativo a sus productos: pizzas, bases, ingredientes, etc. Y por otra parte, lo relativo al sistema transaccional de ventas: clientes, pedidos, pagos, entregas, etc.

Los dos contextos anteriores, incluidos dentro del dominio completo de análisis, contendrán todas las entidades, relaciones y atributos interesantes para el análisis de la información y la construcción del proceso analítico. Este conocimiento debe estar disponible a través de ontologías para poder ser utilizado en el marco de trabajo.

Para ello, se han adaptado y utilizado en el caso de uso dos ontologías públicas, disponibles en la librería de ontologías de Protégé (Stanford University). Protégé es una de las aplicaciones y marcos de trabajo más utilizados actualmente para la creación, edición y utilización de ontologías [46].

En concreto, las 2 ontologías utilizadas son:

- Pizzas. Es una ontología de ejemplo utilizada en el tutorial de Protégé sobre OWL.
- Resource-Event-Agent Enterprise (REA). Es una ontología que se usa para modelizar diferentes aspectos económicos en marcos de trabajo e-commerce y sistemas de información empresarial.

La ontología referente a las pizzas es muy simple, aun así, es frecuente su utilización y se considera un ejemplo muy bueno, porque muestra de una forma sencilla diferentes tipos de entidades, atributos y relaciones. Algunos de ellos no son triviales, como por ejemplo: "Una pizza vegetariana no puede contener ingredientes de tipo carne o pescado".

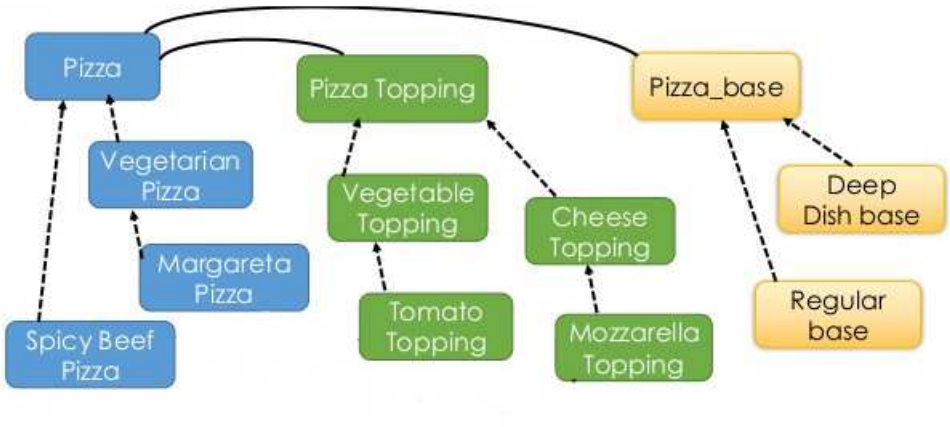

Figura 29: Ejemplo de entidades y relaciones incluidas en la ontología Pizzas

De cara a demostrar el caso de uso, esta ontología contiene todos los conceptos necesarios para entender qué es una pizza, que base e ingredientes la forman, si éstos son picantes, etc. Como se puede ver en el ejemplo de la figura 29, incluye además clases concretas de pizzas (Margarita, Caprichosa, Cuatro Estaciones, etc.) que serán las utilizadas para entrenar el modelo predictivo y generar las recomendaciones.

Por su parte, la ontología REA recoge los conceptos propuestos dentro del modelo Recurso-Evento-Agente. Este modelo propuesto originalmente por W. E. McCarthy en 1982 (Michigan State University), trata de simplificar los sistemas contables, proporcionando una estructura general diferente, en la que los objetos del modelo se pueden mapear directamente a

elementos del negocio. Aunque no es muy común ver este tipo de modelado en sistemas o bases de datos contables empresariales, sus características han influenciado y se pueden aplicar en la interoperabilidad de los sistemas de negocio electrónico.

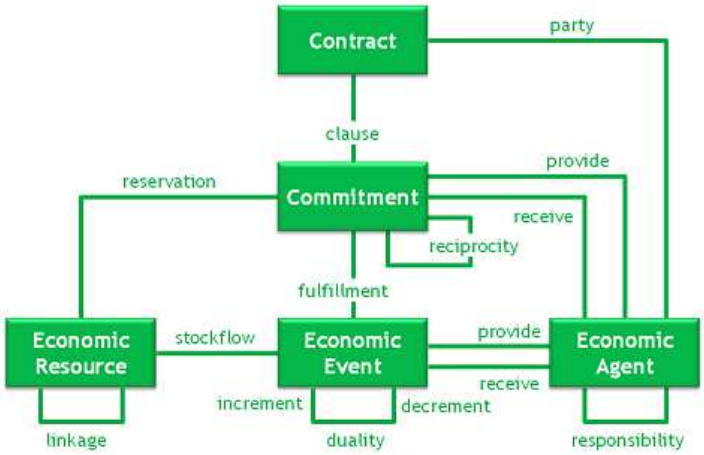

Figura 30: Meta-modelo de procesos incluido en la ontología REA

En esta ocasión, los conceptos que se van a utilizar son también bastante adecuados para la demostración del caso de uso. A continuación se muestra un listado con el mapeo realizado entre conceptos REA y los conceptos del dominio descrito:

- Recursos (*Resources*): Los productos en venta, es decir, las pizzas.
- Agentes (*Agents*): Las partes que acuerdan la venta, el cliente y la pizzería.
- Contrato (*Contract*): El concepto que identifica la venta, en este caso un pedido.
- Compromiso (*Commitment*): Cada elemento del pedido, que indica la pizza ordenada.
- Eventos (*Events*): Que se producen durante la venta: pago, envío o recogida.

Este modelo de representación recuerda en cierto modo a un sistema de venta transaccional, tal y como se puede apreciar en la figura 30. Por eso, serán estos mismos conceptos los que definan también la estructura de los datos disponibles y que se van a utilizar para la generación del modelo de recomendación. El origen de estos datos para cada entidad será definido utilizando el lenguaje específico de dominio.

Para poder generar un conjunto de datos que sirva de entrenamiento para el modelo, será necesario analizar estos datos e identificar una serie de variables de entrada que sean suficientemente representativas. Algunos ejemplos complejos de variables que se podrían llegar a considerar son: “media del coste de los pedidos realizados por el cliente en los últimos 3 meses”, “Porcentaje de pizzas pedidas con ingredientes picantes”, etc. La definición y generación de estas variables agregadas necesita que el proceso analítico incluya una serie de operaciones y transformaciones, que serán definidas también a través del DSL.

Además de las variables de entrada, el proceso debe también generar una variable objetivo (“*target*”) por cada pedido, cuyos valores serán las pizzas ordenadas. Con esta variable objetivo se podrá construir un proceso de aprendizaje supervisado, que generará el modelo

mediante la aplicación de un algoritmo de machine learning concreto. En este caso, el modelo a generar es realmente un problema de clasificación, para identificar la pizza a recomendar a partir de las variables de entrada, por lo que se utilizará un árbol de decisión para realizar la predicción.

Para la demostración del caso de uso se ha utilizado un volumen de datos reducido, aunque sí se ha seleccionado una plataforma de desarrollo y herramientas Big Data que puedan permitir manejar grandes volúmenes de información. Para realizar la implementación de todos los componentes del producto se ha utilizado principalmente Apache Spark. Esta solución proporciona una serie de capacidades Big Data, basadas en un motor de procesamiento distribuido de datos con un alto rendimiento, sobre Apache Hadoop. Que incluye también, un conjunto de librerías de minería de datos y aprendizaje automático muy completas y optimizadas.

Aunque Apache Spark se encuentra implementado utilizando el lenguaje de programación Scala, de cara a su utilización para la programación de aplicaciones, además de las opciones más naturales de uso como Java o el propio Scala, también posee un API específico diseñado para el lenguaje Python. Por lo que se considera totalmente adecuado para la integración directa con el marco de trabajo.

En el desarrollo del caso de uso se ha seguido una metodología de trabajo basada en un proceso iterativo de minería de datos (figura 31). En un caso real, la primera fase del desarrollo debe incluir principalmente la comprensión del dominio y de los datos disponibles. Es decir, se debe entender completamente el problema, los conceptos que intervienen y los detalles sobre la información que se posee. En este caso, avanzaremos hasta la segunda fase en la que el esfuerzo se centra en el desarrollo del sistema analítico, definición e implementación de los diferentes componentes software.

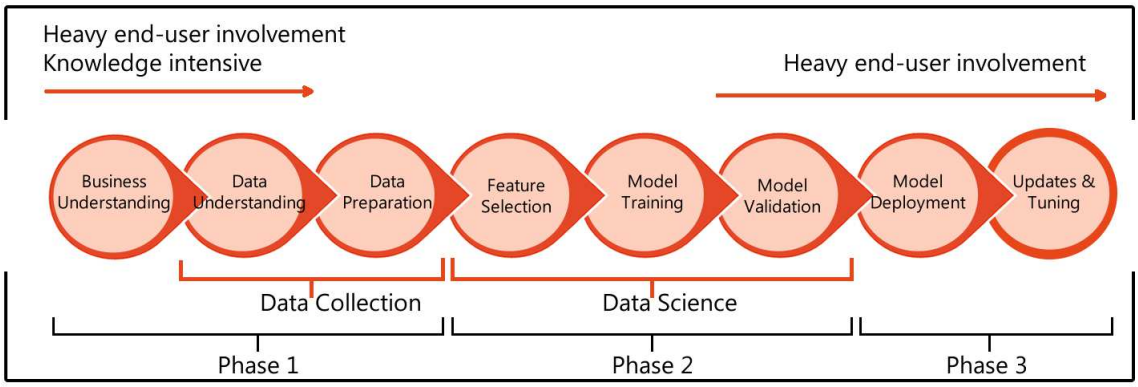

Figura 31: Proceso estándar de ciencia de datos Big Data aplicado al caso de uso

En este punto, ya se puede observar la primera ventaja aportada al adoptar la utilización del marco de trabajo. Debido a la necesidad de definición del proceso en términos del modelo de dominio, la propia solución obliga a detenerse para reflexionar sobre el posible diseño del proceso y la arquitectura de los componentes involucrados, evitando así el adentrarse sin más a implementar el código fuente necesario para cada tarea.

En este caso de uso, el diseño del proceso se ha especificado casi directamente en términos del modelo de características del marco de trabajo, mediante la definición de las siguientes operaciones y tareas concretas:

1. ***Lectura del origen de datos***. Los datos de entrada disponibles se encontrarán almacenados en diferentes directorios dentro del sistema de ficheros distribuidos. Es necesaria la implementación de un componente para realizar la lectura de dicha información.
2. ***Generación del conjunto de datos.*** Se deben aplicar una serie de funciones de transformación y agregación sobre los datos de entrada para crear un conjunto de datos de entrenamiento. Es necesaria la implementación de los componentes que incluirán estas funciones específicas en base a los conceptos del dominio de análisis.
3. ***Particionado del conjunto de datos***. Se va a separar el conjunto de datos en dos partes, para ser utilizadas en el entrenamiento del modelo ("*train*") y su posterior evaluación ("*test*"). Esta operación sobre el conjunto de datos de entrada debe implementarse como un componente aparte.
4. ***Generación del modelo***. La tarea de creación del árbol de decisión debe utilizar el conjunto de datos de entrenamiento y necesitará de una serie de propiedades de configuración para la controlar la implementación concreta del algoritmo.
5. ***Evaluación del modelo***. Se va a aplicar el modelo generado al conjunto de datos de test para evaluar su rendimiento. Esta tarea de predicción constituye otra característica incluida en el producto.

Estas fases del proceso definen todas las características del sistema a construir. Como no podría ser de otra forma, la segunda ventaja derivada de la utilización del marco de trabajo es la inclusión de esta definición del proceso analítico, dentro del propio sistema, a través de la utilización del lenguaje específico de dominio. A esto hay que sumarle además los beneficios que aporta el utilizar los conceptos del dominio de análisis para guiar esta definición.

Al entrar en los detalles de implementación del DSL, la utilización de estos conceptos del dominio requiere que el primer paso sea cargar las ontologías mencionadas anteriormente, tal y como se puede ver en el siguiente fragmento de código.

```
from unedtfm import *
domain.load(["pizza.owl", "rea.owl"], "C:\\ontologias")
```

La primera instrucción realiza la importación de las capacidades del marco de trabajo, mientras que la segunda sentencia del DSL importará las dos ontologías desde la ruta especificada, haciendo que las entidades se integren como clases del propio modelo de dominio. Estas clases se pueden utilizar para definir la primera característica del proceso, que hace referencia a la lectura de los orígenes de datos.

```
from source import datasource
domain.pizza.Pizza.datasource("file", "pizza.csv")
        .impl(datasource.datasource_from_file)
domain.rea.Contract.datasource("file", "order.csv")
        .code("source.datasource", "datasource_from_file")
```

En la segunda sentencia del ejemplo anterior se utiliza la clase “Pizza” del contexto “pizza”, que se encuentra dentro del dominio, para realizar la especificación de su origen de datos. Los datos se encuentran definidos por el tipo fichero (“file”) y están identificados por la ruta “pizza.csv”. Estos parámetros se incorporarán a los campos internos del objeto datasource generado. Además, utilizando la propiedad “impl” de este objeto, se ha especificado también la función “datasource_from_file” del módulo “datasource”, que realiza la implementación de esta característica de lectura del origen de datos.

Esta especificación se ha realizado también para el resto de las entidades que se vayan a utilizar posteriormente en el proceso. Como por ejemplo en la tercera sentencia anterior, para el elemento “Contract”, que hace referencia a los datos de pedidos de pizzas. En este caso, el componente encargado de implementar la funcionalidad se ha establecido utilizando la opción “code”, que necesita los nombres del módulo y la función. Esto evita tener que importar y hacer referencia al código de los componentes.

Como se puede apreciar, estas dos tareas ponen de manifiesto más capacidades y ventajas derivadas del uso del marco de trabajo. Por una parte, se utilizan los propios conceptos del dominio para guiar la especificación de características, y por otra parte, el marco de trabajo ayuda a modularizar la implementación del sistema, obligando a generar componentes que implementen cada una de las características definidas.

A continuación se muestra un fragmento de código extraído del componente que realiza la lectura de datos mediante la función “datasource_from_file”:

```
from source.spark import *
def datasource_from_file(*args, **kargs):
    datasource = args[0]
    if "file" == datasource.source_type:
        sc = sparkContext()
        rdd = sc.textFile(name=datasource.source_location) \
            .map(lambda line: line.split(","))
        _RDD_DATASOURCES[datasource.domain_class.name] = rdd
```

Esta función espera como parámetro un objeto “datasource” definido dentro del modelo de dominio a través de los ejemplos del DSL anteriores. Mediante esta especificación del origen de datos el componente puede realizar su implementación precisa, comprobando por ejemplo si el tipo del origen de datos es un fichero. Este mismo acceso a la información se podría realizar también a través del módulo API del marco de trabajo.

Toda la implementación se ha desarrollado utilizando las características de Apache Spark. Sin entrar en los detalles concretos y el funcionamiento de Spark, en primer lugar, se crea e inicializa el objeto SparkContext que contiene toda la configuración del contexto de ejecución de Spark y que permite realizar el procesamiento distribuido de los datos. Con este objeto se puede realizar, en segundo lugar, la lectura del fichero desde la ruta indicada mediante el método “textFile”. Se aplica además una función de mapeo (“map”) de cada una de las líneas del fichero, separando los datos de cada atributo delimitados por comas.

De esta forma se genera un objeto de tipo RDD ("*Resilient Distributed Dataset*"), que es la principal abstracción de Spark para hacer referencia a conjuntos de datos distribuidos. Por último, este objeto se almacena en un diccionario utilizando como clave el nombre de la clase de dominio, para su posterior utilización en la realización de transformaciones y operaciones.

Lo más importante de este ejemplo, es la demostración de las posibilidades de desarrollo que se pueden seguir con la forma de uso del marco de trabajo. La implementación de los componentes es directa, en este caso, mediante el uso de funciones que reciben los objetos del modelo de dominio. Esto permite independizar los componentes implementados para que puedan ser reutilizables en otros procesos y productos similares, en los que la lectura de datos se produzca desde una fuente similar. Consiguiendo así reducir el tiempo de desarrollo de nuevos sistemas. Lo mismo ocurre para las funciones de transformación o el algoritmo de generación del modelo, éstos se van a poder reaprovechar directamente para otros casos de uso dentro del mismo dominio de aplicación.

```
CT = domain.rea.Contract
AG = domain.rea.Agent
CM = domain.rea.Commitment

ds = CT.dataset("orders_dataset", [
    CT.property("id").variable("order_id"),
    CT.property("date").transform(function(lambda x: x.month))
       .variable("order_month"),
    AG("Customer", "type").property("city"),
    AG("Company", "type").property("store"),
    CM.property("resource").variable("product"),
    CM.operation(function("operations", "target")).variable("target")
])
```

El ejemplo anterior contiene la definición de un nuevo conjunto de datos denominado "orders_dataset" para la clase "Commitment". Este conjunto de datos se ha generado a través de la definición de las variables que lo componen, utilizando una serie de operaciones y transformaciones a partir de las propiedades y atributos de las entidades del dominio relacionadas.

En este fragmento de código se puede ver también otra ventaja derivada de la implementación del DSL como un lenguaje embebido dentro de Python. Todas las operaciones y características de este lenguaje están disponibles también en el DSL, como por ejemplo, se han asignado los alias a las clases de dominio en las tres primeras instrucciones, o se han utilizado funciones lambda "en línea" ("inline") para definir la transformación de la variable "date".

Sin necesidad de entrar en todas las transformaciones de los datos que han sido necesarias para generar el conjunto de datos para el entrenamiento del caso de uso, si es necesario comentar en este caso de uso la necesidad de mapear diferentes entidades del dominio. En este caso, los contextos que provienen de las ontologías "pizza" y "rea" contienen entidades independientes sobre las que es necesario crear diferentes vínculos. El marco de trabajo incluye en el DSL una característica concreta para realizar este mapeo entre clases de diferentes contextos.

```
domain.pizza.Pizza.bind(domain.rea.Resource)
```

La sentencia anterior permite indicar que el concepto “Pizza” se corresponde con un “Resource” del contexto “rea”. En otras palabras, establece un mapeo de entidades en los contextos mediante una relación del tipo “X es un Y”. De esta forma, todas las pizzas serán tratadas como si fuera un producto o ítem de la transacción definida como un pedido.

```
[train, test] = dataset.partition("random", [0.7, 0.3])

m = CT.model("model", train, {
    "type" : "classification",
    "algorithm": "DecisionTree",
    "target" : "target",
    "exclude" : ["id", "date"]
})

m.predict(test)
```

Para terminar con la explicación acerca de la especificación de características incluidas, se ha incluido en el fragmento de código del DSL anterior la descripción de los últimos pasos, correspondientes al particionado del dataset generado en dos “train” y “test”. Con estos conjuntos de datos se generará el modelo y se realizarán las predicciones para comprobar el rendimiento del mismo. El desarrollo de los componentes que implementan estas características se ha realizado siguiendo los ejemplos incluidos dentro de la documentación de Apache Spark sobre árboles de decisión [47] [48].

Como se ha podido ver, la utilización del DSL permite una definición estructurada y muy flexible, que tras su interpretación, dará lugar al modelo de dominio que contiene todas las características del proceso. Este modelo se utilizará en los componentes concretos para realizar la implementación y ejecución del producto software analítico.

Dentro del marco de trabajo, el modelo de objetos proporciona también una serie de interfaces para el acceso a la información que contiene, por lo que se puede consultar desde otros módulos del software construido para realizar diferentes tareas. Uno de los interfaces es la implementación del visitor que permite recorrer los diferentes elementos del modelo. A continuación se incluye a modo de ejemplo uno de los visitors desarrollados en este caso de uso.

```
class html_visitor(dsl_visitor):

    @visitor(domain_model)
    def visit(self, domain_model):
        print("<html>")
        print("<body>")

    @visitor(domain_ontology)
    def visit(self, ontology):
        print("<h1>Domain Ontology: %s </h1>" % ontology.name)

    @visitor(domain_class)
    def visit(self, domain_class):
        print("<p>Domain Class: %s </p>" % domain_class.name)
```

En este caso tan sencillo, la clase “html_visitor” se ha utilizado simplemente para generar un código HTML a modo de documentación. Este HTML incluirá los nombres de las ontologías y de las clases que se han importado en cada una.

Los métodos “visit” creados han sido decorados mediante la anotación “@visitor” indicando el tipo de objeto que trata cada uno. Esta clase extiende la clase “dsl_visitor” que es la base de la implementación incluida en el marco de trabajo. Mediante este patrón de diseño, cada uno de los objetos del modelo acepta la visita de un objeto de este tipo, permitiendo que las implementaciones concretas realicen las operaciones o apliquen los algoritmos concretos.

```
class domain_ontology(parent, parent_visitor):
    def accept(self, visitor):
        parent_visitor.accept(self, visitor)
        for c in self.domain_classes:
            c.accept(c, visitor)
```

Por ejemplo, en el fragmento de código anterior se puede ver la implementación del método “accept” de la clase “domain_ontology”. Esta clase acepta la visita, utilizando los mecanismos genéricos incluidos en la clase base “parent_visitor” y además, por cada una de las clases de dominio incluidas en la ontología, se invoca al método para continuar con el recorrido jerárquico del modelo de objetos. Este mismo mecanismo se puede utilizar para otras tareas más complejas, como por ejemplo, la generación automática de código de los componentes.

4. Conclusiones

4.1. Resultados

El trabajo realizado es la consecución del esfuerzo por iniciar una línea de investigación sobre la posible evolución en el desarrollo de un tipo de productos software, basados en procesos analíticos avanzados mediante soluciones Big Data, a través de la aplicación de modelos específicos del dominio.

Esta línea de trabajo parte de la detección de una serie de necesidades en este ámbito y la falta de soluciones concretas tras una amplia experiencia en el uso de diferentes soluciones comerciales y metodologías más comunes en la industria. Desde este punto, se han estudiado las diferentes aproximaciones y trabajos realizados por otros investigadores, así como las posibilidades de reutilización de los paradigmas vigentes en otros procesos de ingeniería y ecosistemas de desarrollo de software.

Esto ha dado lugar a la realización de un planteamiento teórico de la solución, que se basa fundamentalmente en el diseño de un marco de trabajo específico, que hace uso de modelos y lenguajes específicos del dominio para permitir que sean los propios conceptos del dominio los que guíen la especificación de características de los procesos analíticos que componen el producto software a construir.

Además del planteamiento teórico, el trabajo se ha completado con la implementación de un prototipo del marco de trabajo propuesto, con la inclusión de un conjunto básico y restringido de funcionalidades, pero que forman un sistema completamente funcional. Esta solución se puede adoptar de forma directa en la resolución de problemas de minería de datos basados en la generación de modelos estadísticos predictivos, tal y como se ha comprobado en los ejemplos y pruebas de aplicación realizados.

Por todo esto, el principal resultado extraído de la elaboración del trabajo es la capacidad efectiva de la solución planteada para aportar una serie de ventajas diferenciales en el desarrollo de software analítico avanzado. Estas ventajas pueden marcar una evolución necesaria en los procesos de desarrollo actuales.

A continuación se incluye un resumen de las principales ventajas aportadas por la solución realizada:

- Permite incluir una descripción precisa y estructurada de las características de los procesos analíticos construidos. Desde la compresión del problema y la información disponible, hasta las conclusiones extraídas o decisiones tomadas.
- Todo ello expresado en términos del propio dominio, muy cercanos y comprensibles desde un punto de vista de negocio. Por lo que se puede utilizar de forma sencilla como documentación exhaustiva del trabajo realizado.

- Agiliza el traspaso y compartición directa del conocimiento adquirido durante el estudio del dominio y el problema planteado, a través del análisis y transformación de los datos, hasta llegar a los resultados intermedios y conclusiones finales.
- Facilita el desarrollo, mantenimiento, ampliación y mejora del software construido durante el proceso de minería de datos, al igual que en su posterior puesta en producción.
- Minimiza el esfuerzo de duplicar funcionalidades ya disponibles y de realizar intentos previamente fallidos o no concluyentes, al estructurar y especificar las características concretas de todos los procesos analíticos.
- Fomenta la reutilización del código fuente de los programas a través de componentes independientes, así como, las técnicas (procesos, algoritmos, etc.) utilizados y la información (conjuntos de datos) agregada o derivada durante el análisis.
- Posibilita la creación de nuevas metodologías y herramientas que mejoren la productividad, reducción de coste e incremento en la velocidad de desarrollo de productos software con características similares.
- Proporciona una serie de interfaces y mecanismos básicos que sirven de soporte e integración, de una forma sencilla, dentro de otras aplicaciones o entornos de trabajo para la creación de soluciones más avanzadas.
- Abre nuevas vías de investigación al sentar las bases iniciales y proporcionar una serie de conceptos, técnicas y herramientas totalmente extensibles.

Mediante la realización del trabajo y durante la realización de esta memoria se han podido extraer y poner de manifiesto una serie de conclusiones importantes sobre el ámbito de aplicación de la solución, que confirman los objetivos planteados inicialmente.

Se ha podido comprobar que efectivamente existe una necesidad real de evolución de las metodologías y procesos de desarrollo de este tipo de productos software basados en procesos analíticos de minería de datos sobre tecnologías Big Data. Para alcanzar esta evolución, se antoja necesario elevar el nivel de abstracción y agilizar los desarrollos, incorporando el conocimiento específico del dominio, independizando la definición de los procesos de las plataformas utilizadas y fomentando la modularización y reutilización de componentes.

Se necesita pasar de una definición clásica de características de los procesos centrada en los datos y acoplada a la tecnología utilizada, a una nueva descripción abstracta del proceso, guiada por los conceptos del dominio, que pueda ser consultada desde los componentes específicos construidos. Esto implica replantear y mejorar las metodologías de trabajo, basando los nuevos esfuerzos en la utilización de marcos de trabajo que aporten una serie de ventajas prácticas y elementos diferenciales para mejorar su desarrollo.

En este sentido, se ha podido comprobar que los paradigmas cercanos al diseño y desarrollo de software dirigidos por modelos del dominio pueden ejercer una papel determinante para alcanzar una nueva generación de soluciones, que aporten mayor flexibilidad y agilidad. Y en definitiva, mejorar el ciclo de vida completo del software.

Por último, las características de la solución alcanzada y los resultados obtenidos hacen indicar que el marco de trabajo planteado se puede utilizar como base para la evolución del desarrollo de los productos software analíticos. Por ello, la principal conclusión derivada de la realización del trabajo es que se han alcanzado totalmente los objetivos planteados inicialmente.

4.2. Trabajos futuros

El marco de trabajo desarrollado como prototipo incluye una serie de funcionalidades básicas para demostrar las posibilidades de la solución planteada. Sin embargo, en su diseño se ha procurado siempre sentar las bases para que pueda tener un recorrido más amplio, posibilitando diferentes líneas de investigación y muchas posibles funcionalidades nuevas que se podrían incorporar.

Tanto por la forma de planteamiento realizada como por las características de este tipo de marcos de trabajo, es un candidato perfecto para formar por ejemplo, el inicio de un proyecto colaborativo de código abierto. Esto podría aportar muchos beneficios y cubrir dos necesidades básicas que determinen el éxito de la solución. Por una parte, se incorporarían nuevos desarrolladores que podrían incrementar y mejorar las capacidades del sistema, y por otra parte, se podría potenciar y popularizar el uso del marco de trabajo entre la comunidad de desarrolladores, para estudiar sus experiencias y recopilar sus necesidades concretas.

En las tablas 7 y 8 se han intentado recoger algunos de los posibles trabajos futuros partiendo del trabajo y estado actual de la solución. En lo que respecta tanto a la incorporación de nuevas características y mejoras, como también en otras líneas de trabajo más avanzadas.

1.1	Mejorar comportamiento del módulo de parseo de ontologías
	Para continuar utilizando la librería OWLready tal y como está actualmente, se deberían mejorar una serie de funcionalidades que permitan mayor flexibilidad en la definición de las ontologías. Otra posibilidad es sustituir esta librería por un módulo específico a medida.
1.2	Añadir nuevos formatos para leer y utilizar las ontologías
	Además de OWL, existen actualmente otros formatos y especificaciones diferentes para el desarrollo de ontologías (por ejemplo: RDF). Aunque existen programas que permiten la conversión, se podrían permitir la lectura y parseo directo desde el sistema.
1.3	Mejorar las relaciones entre los conceptos de dominio
	Se podría incorporar el soporte para relaciones semánticamente más complejas entre los conceptos del dominio. Además de las relaciones más habituales (estilo 1 a N y N a N) del tipo “X está relacionado con Y”, aportaría gran valor diferenciar relaciones del tipo “posee”, “es un”, “pertenece a”, “es la unión de”, “es lo contrario a”, etc.
1.4	Mejorar el mapeo entre conceptos de dominio y conjuntos de datos

	Dentro de las operaciones del lenguaje se podría aumentar las opciones de mapeo entre los conceptos de dominio y los datos a analizar, por ejemplo a través del uso de expresiones naturales más complejas. Esto permitiría soportar más formatos diferentes en los conjuntos de datos, que incluyan o excluyan ciertos atributos y relaciones.
1.5	Mejorar las operaciones y capacidades del DSL
	El lenguaje específico de dominio posee varios campos distintos de mejora, por ejemplo en lo que respecta a su configuración, integración con otros sistemas, capacidades de desarrollo iterativo, versionado, etc. También podría ser interesante crear funcionalidades avanzadas para usuarios expertos, por ejemplo, sobrecargando operadores o definiendo atajos para ciertas operaciones.
1.6	Soportar múltiples procesos analíticos enlazados
	El marco de trabajo actual soporta la definición de procesos analíticos independientes, aunque podría ser de utilidad tener un componente global que pueda gestionar los diferentes procesos definidos a través del DSL, pudiendo incluso definir dependencias o enlazarlos para su ejecución.
1.7	Incluir funcionalidades y capacidades para nuevos casos de uso
	En la implementación actual se ha limitado el alcance a una serie casos de uso concretos de minería de datos y generación de modelos predictivos. Aunque estos casos son muy frecuentes sería interesante incorporar nuevas capacidades para otros problemas analíticos: optimización, simulación, etc.
1.8	Incluir nuevas características analíticas avanzadas
	Continuando con el punto anterior, otra posibilidad es incluir nuevas características analíticas más avanzadas, que requerirían el diseño de nuevas funcionalidades específicas en el marco de trabajo, como por ejemplo: Deep Learning, Probabilistic Programming, Text Analytics, etc.
1.9	Incrementar los interfaces para la consulta de los metadatos
	Mejorar los interfaces de programación de aplicaciones es casi un punto obligado a medida que se incrementan las características del marco de trabajo. En este sentido, las mejoras podrían ir en la dirección de incluir nuevas facilidades para la consulta y modificación de metadatos o la transformación de los modelos.

Tabla 7: Lista de posibles mejoras y trabajos futuros

2.1	Nuevos interfaces para incluir conceptos y conocimiento extraído
	Esta posible línea de trabajo otorgaría al marco de trabajo la capacidad de incorporar nuevos conceptos de dominio y conocimiento adquirido a través del propio análisis de los datos. Se podrían realimentar las ontologías para que reflejen las conclusiones y los resultados obtenidos.
2.2	Soporte para el razonamiento sobre las ontologías
	Otra posibilidad es la de incorporar capacidades de razonamiento sobre el conocimiento incluido en las ontologías. Esto podría permitir automatizar ciertas fases de los procesos de análisis y sería un primer paso hacia la integración con asistentes inteligentes.
2.3	Añadir más soporte para la generación automática de código
	Mejorar las capacidades de transformación de los modelos de características de los procesos dentro del propio marco de trabajo, podría permitir también una nueva línea de trabajo más cercana a la ingeniería de desarrollo dirigida por modelos y a los paradigmas de generación automática de código.
2.4	Aumentar las opciones de ejecución directa de procesos
	A pesar de que el DSL se puede utilizar para realizar la ejecución directa de los procesos, a través del sistema de inversión de control, se antoja necesario un incremento de estas capacidades para que las ejecuciones se puedan configurar dentro del marco de trabajo y que este pueda ser la pieza central con la que construir el sistema software completo.
2.5	Construir herramientas basadas en el marco de trabajo
	De cara a popularizar el uso de la solución, otra línea de trabajo futura recomendable sería la construcción de herramientas que faciliten y mejoran su utilización de cara a los usuarios. Basándose en el DSL se podrían construir, por ejemplo: un analizador avanzado que tenga en cuenta la información de dominio e incluso algunas herramientas visuales, como un entorno de desarrollo integrado (IDE).

Tabla 8: Lista de características avanzadas y líneas de investigación futuras

5. Referencias

5.1. Bibliografía

[1] A. W. Brown y K. C. Wallnau, «Engineering of Component-Based Systems,» de *Second IEEE International Conference on Engineering of Complex Computer Systems*, 1996.

[2] A. W. Brown, «Preface: Foundations for component-based software engineering,» de *Selected papers from the Software Engineering Institute*, IEEE Computer Society Press, 1996, pp. 7-10.

[3] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley, 2004.

[4] E. Evans, Domain-Driven Design: Definitions and Pattern Summaries, Domain Language Inc., 2006.

[5] U. M. Fayyad, G. Piatetsky-Shapiro y P. Smyth, «From data mining to knowledge discovery: an overview,» *Advances in Knowledge Discovery and Data Mining,* pp. 1-34, 1996.

[6] A. Azevedo y M. F. Santos, «KDD, SEMMA y CRISP-DM: A Parallel Overview,» de *IADIS European Conference Data Mining*, Amsterdam, 2008.

[7] «SEMMA - SAS Enterprise Miner - SAS Institute,» SAS, [En línea]. Available: https://web.archive.org/web/20120308165638/http://www.sas.com/offices/europe/uk/technologies/analytics/datamining/miner/semma.html/.

[8] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer y R. Wirth, «CRISP-DM 1.0 Step-by-step data mining guide,» CRISP-DM Consortium, 1999.

[9] C. Shearer, «The CRISP-DM Model: The New Blueprint for DataMining,» *JOURNAL of Data Warehousing,* vol. 5, nº 4, pp. 13-22, 2000.

[10] G. Piatetsky, «CRISP-DM, still the top methodology for analytics, data mining, or data science projects,» KDnuggets, Octubre 2014. [En línea]. Available: http://www.kdnuggets.com/2014/10/crisp-dm-top-methodology-analytics-data-mining-data-science-projects.html.

[11] J. Haffar, «Have you seen ASUM-DM? - IBM SPSS Predictive Analytics,» 16 Octubre 2015. [En línea]. Available: https://developer.ibm.com/predictiveanalytics/2015/10/16/have-

you-seen-asum-dm/.

[12] J. Rollins, «Why we need a methodology for data science - IBM Big Data & Analytics,» 24 Agosto 2015. [En línea]. Available: http://www.ibmbigdatahub.com/blog/why-we-need-methodology-data-science.

[13] Microsoft, «Team Data Science Process (TDSP) - Microsoft Azure,» 2016. [En línea]. Available: https://azure.microsoft.com/en-us/documentation/learning-paths/data-science-process/.

[14] J. J. Zubcoff y J. J. Trujillo, «Extending the UML for designing association rule mining models for data warehouses,» de *DaWaK'05: Proceedings of the 7th international conference on Data Warehousing and Knowledge Discovery*, 2005.

[15] J. J. Zubcoff y J. J. Trujillo, «Conceptual Modeling for Classification Mining in Data,» de *DaWaK'06: Proceedings of the 8th international conference on Data Warehousing and Knowledge Discovery*, 2006.

[16] J. J. Zubcoff y J. J. Trujillo, «A UML 2.0 profile to design Association Rule mining models in the multidimensional conceptual modeling of data warehouses,» *Data & Knowledge Engineering,* vol. 63, nº 1, 2007.

[17] J. N. Mazón y J. Trujillo, «An MDA approach for the development of data warehouses,» *Decision Support Systems,* vol. 45, nº 1, pp. 41-58, 2008.

[18] J. Pardillo, J. N. Mazón, J. Zubcoff y J. Trujillo, «Towards a model-driven engineering approach of data mining,» de *IADIS European Conference on Data Mining*, 2008.

[19] J. Zubcoff, J. Pardillo, J. N. Mazón y J. Trujillo, «Integrating the Development of Data Mining and Data Warehouses via Model-driven Engineering,» de *Jornadas de Ingeniería del Software y Bases de Datos*, 2008.

[20] D. Breuker, «Towards Model-Driven Engineering for Big Data Analytics - An Exploratory Analysis of Domain-Specific Languages for Machine Learning,» de *47th International Conference on System Sciences (HICSS)*, 2014.

[21] J. Winn, C. Bishop y T. Diethe, Model-Based Machine Learning, Microsoft Research, 2015.

[22] M. Guerriero, S. Tajfar, D. A. Tamburri y E. Di Nitto, «Towards A Model-Driven Design Tool for Big Data Architectures,» de *BIGDSE '16 Proceedings of the 2nd International Workshop on BIG Data Software Engineering*, 2016.

[23] B. Deshpande, «The value of domain knowledge in data science,» Simafore, 6 Marzo 2015. [En línea]. Available: http://www.simafore.com/blog/the-value-of-domain-

knowledge-in-data-science.

[24] B. Raskutti, «The Role of Domain Experts in Data Science,» Teradata, 23 Marzo 2015. [En línea]. Available: http://blogs.teradata.com/international/role-domain-experts-data-science/.

[25] D. Roy, «The missing D in Data Science,» Julio 2015. [En línea]. Available: http://www.kdnuggets.com/2015/07/data-science-domain-knowledge.html.

[26] T. Ruths, «The New Culture: Domain Data Science,» Tibco, 6 Agosto 2015. [En línea]. Available: http://www.tibco.com/blog/2015/08/06/the-new-culture-domain-data-science/.

[27] K. Leetaru, «Why We Need More Domain Experts In The Data Sciences,» Forbes, 12 Junio 2016. [En línea]. Available: http://www.forbes.com/sites/kalevleetaru/2016/06/12/why-we-need-more-domain-experts-in-the-data-sciences/.

[28] L. Cao y C. Zhang, «Domain-driven data mining: a practical methodology,» *International Journal of Data Warehousing and Mining (IJDWM),* vol. 2, nº 4, pp. 49-65, 2006.

[29] L. Cao y C. Zhang, «The evolution of KDD: Towards domain-driven data mining,» *International Journal of Pattern Recognition and Artificial Intelligence,* vol. 21, nº 04, pp. 677-692, 2007.

[30] L. Cao, «Domain-driven, actionable knowledge discovery,» 2007.

[31] L. Cao, «Domain driven data mining (d3m),» de *IEEE International Conference on Data Mining Workshops*, 2008.

[32] L. Cao, «Domain-driven data mining: Challenges and prospects,» *IEEE Transactions on Knowledge and Data Engineering,* vol. 22, nº 6, pp. 755-769, 2010.

[33] C. Zhang, P. S. Yu y D. Bell, «Introduction to the Domain-Driven Data Mining,» *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING,* vol. 22, nº 6, pp. 753-754, 2010.

[34] L. Cao y C. Zhang, «Domain-Driven Actionable Knowledge Discovery in the Real World,» de *10th Pacific-Asia Conference, PAKDD, Advances in Knowledge Discovery and Data Mining*, 2006.

[35] L. Brisson y M. Collard, «An ontology driven data mining process,» de *International Conference on Enterprise Information Systems*, 2008.

[36] F. Mota Pinto y M. F. Santos, «Considering Application Domain Ontologies for Data Mining,» *WSEAS Transactions on Information Science and Applications,* vol. 6, nº 9, pp.

1478-1492, 2009.

[37] P. Panov, S. Dzeroski y L. Soldatova, «OntoDM: An ontology of data mining,» de *8th IEEE International Conference on Data Mining (ICDM)*, Pisa, 2008.

[38] J. Vanschoren y L. Soldatova, «Exposé: An ontology for data mining experiments,» de *Third Generation Data Mining Workshop at ECML PKDD 2010*, 2010.

[39] H. Blockeel y J. Vanschoren, «Experiment Databases: Towards an Improved Experimental Methodology in Machine Learning,» de *Proceedings of the 11th European conference on Principles and Practice of Knowledge Discovery in Databases*, 2007.

[40] J. Vanschoren, H. Blockeel, B. Pfahringer y G. Holmes, «Experiment databases,» *Journal Machine Learning,* vol. 87, nº 2, pp. 127-158, 2012.

[41] J.-U. Kietz, F. Serban, A. Bernstein y S. Fischer, «Towards Cooperative Planning of Data Mining Workflows,» de *Workshop on Third Generation Data Mining at the European Conference on Machine Learning*, 2009.

[42] A. Bernstein , S. Hill y P. Foster, «Intelligent Assistance for the Data Mining Process: an Ontology-Based Approach,» *Information Systems Working Papers Series,* 2002.

[43] A. Bernstein, F. Provost y S. Hill, «Toward Intelligent Assistance for a Data Mining Process: An Ontology-Based Approach for Cost-Sensitive Classification,» *IEEE Transactions on Knowledge and Data Engineering,* vol. 17, nº 4, pp. 503-518, 2005.

[44] T. Euler y M. Scholz, «Using Ontologies in a KDD Workbench,» de *Workshop on Knowledge Discovery and Ontologies at ECML/PKDD*, 2004.

[45] K. Morik y M. Scholz, «The MiningMart Approach to Knowledge Discovery in Databases,» *Intelligent Technologies for Information Analysis,* pp. 47-5, 2004.

[46] «Protégé Ontology Library - Protégé Wiki - Stanford University,» [En línea]. Available: http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library.

[47] «MLlib Guide: Decision Trees - RDD-based API,» Apache, [En línea]. Available: http://spark.apache.org/docs/latest/mllib-decision-tree.html.

[48] «Python MLlib Examples - Github Apache Spark,» [En línea]. Available: https://github.com/apache/spark/tree/master/examples/src/main/python/mllib.

5.2. Siglas

A	
AI	Artificial Intelligence
	Inteligencia Artificial
AKD	Actionable Knowledge Discovery
	Descubrimiento de Conocimiento Accionable
API	Application Programming Interface
	Interface de Programación de Aplicaciones
B	
BC	Bounded Context
	Contexto Delimitado
C	
CIM	Computation Independent Model
	Modelo Independiente de la Computación
CRISP-DM	Cross Industry Standard Process for Data Mining
CWM	Common Warehouse Metamodel
D	
D3M	Domain Driven Data Mining
	Minería de Datos Dirigida por el Dominio
DAX	Data Analysis Expressions
DDD	Domain Driven Design
	Diseño Dirigido por el Dominio
DMG	Data Mining Group
DMO	Data Mining Ontology
DMX	Data Mining Extensions
DSL	Domain Specific Language
	Lenguaje Específico de Dominio
DSM	Domain Specific Modeling
	Modelado Específico del Dominio
DSML	Domain Specific Modeling Language
	Lenguaje de Modelado Específico del Dominio
I	
IDA	Intelligent Discovery Assistant

	Asistente Inteligente para el Descubrimiento
IDE	Integrated Development Environment
	Entorno de Desarrollo Integrado
IoC	Inversion of Control
	Inversión de Control
IoT	Internet of Things
	Internet de las cosas

K

KDD	Knowledge Discovery in Databases
	Descubrimiento de Conocimiento en Bases de Datos
KR	Knowledge Representation and Reasoning
	Representación y Razonamiento del Conocimiento

M

MDA	Model Driven Architecture
	Arquitectura Dirigida por Modelos
MDD	Model Driven Development
	Desarrollo Dirigido por Modelos
MDE	Model Driven Engineering
	Ingeniería Dirigida por Modelos
MDX	Multidimensional Expressions
ML	Machine Learning
	Aprendizaje Automático
MODB	Mining Oriented DataBase
	Base de Datos Orientada a la Minería

O

ODIS	Ontology Driven Information System
	Sistema de Información Dirigido por Ontologías
OE	Ontology Engineering
	Ingeniería Ontológica
OMG	Object Management Group
OWL	Ontology Web Language
	Lenguaje de Ontologías Web

P

PFA	Portable Format for Analytics

	Formato Portable para Analítica
PIM	Platform Independent Model
	Modelo Independiente de la Plataforma
PM	Predictive Modeling
	Modelado Predictivo
PMML	Predictive Model Markup Language
	Lenguaje de Marcado para el Modelado Predictivo
PSM	Platform Specific Model
	Modelo Específico de la Plataforma
R	
RDD	Resilient Distributed Dataset
RDF	Resource Description Framework
	Marco de Descripción de Recursos
S	
SEMMA	Sample Explore Modify Model Asses
SML	Supervised Machine Learning
	Aprendizaje Automático Supervisado
SQL	Structured Query Language
	Lenguaje Estructurado de Consulta
T	
TDSP	Team Data Science Process
U	
UL	Ubiquitous Language
	Lenguaje Ubicuo
UML	Unified Modeling Language
	Lenguaje Unificado de Modelado
X	
XML	Extensible Markup Language
	Lenguaje de Marcado Extensible