Lenguaje específico para el modelado de flujos de trabajo aplicados a ciencia de datos

Rubén Salado-Cid y José Raúl Romero

Dpto. de Informática y Análisis Numérico
Universidad de Córdoba, Campus de Rabanales, 14071 Córdoba
{rsalado,jrromero}@uco.es

Resumen La ciencia de datos permite la extracción de conocimiento utilizando fuentes heterogéneas con grandes volúmenes de datos. Este campo es actualmente una de las áreas de mayor crecimiento debido al aumento exponencial de datos disponibles, con aplicación en un amplio rango de dominios. De hecho, es frecuente que los usuarios no posean conocimientos específicos en computación. Para ellos, los flujos de trabajo son un mecanismo adecuado de representación y modelización de su proceso de trabajo, ya que permiten definir a alto nivel la secuencia de acciones que permitirá capturar la información y transformarla en conocimiento. Los sistemas de gestión de flujos de trabajo son los encargados de ejecutarlos de forma transparente. En este trabajo se presenta, formalizando su sintaxis abstracta, un lenguaje para la definición de flujos de trabajo a distintos niveles de abstracción. El desarrollo de este lenguaje, por su independencia de la notación concreta, se hace necesario para alcanzar un mayor nivel de interoperabilidad entre las aplicaciones ya desarrolladas para este fin.

Keywords: lenguaje específico del dominio, flujos de trabajo, ciencia de datos

1. Introducción

El crecimiento constante de la cantidad de datos disponible ha aumentado el tamaño y la complejidad de la información que necesita ser almacenada y manipulada, provocando que los procesos y técnicas tradicionales de procesamiento de datos hayan dejado de ser eficaces bajo estas condiciones [3]. Esta circunstancia ha dado lugar al crecimiento de áreas de investigación, como la ciencia de datos [6], que tratan de dar solución a los problemas derivados de la ingente cantidad de datos a tratar. La ciencia de datos utiliza conocimientos procedentes de otros campos, como la minería de datos, estadística o aprendizaje automático, para el desarrollo de nuevos procedimientos que permitan la extracción de conocimiento a partir de fuentes de datos heterogéneas, independientemente de las restricciones de almacenamiento, volumen de los datos, capacidad de procesamiento o medios de comunicación. En este contexto, la mayor parte del tiempo de cómputo se emplea en operaciones de transporte y procesamiento de grandes cantidades de datos, lo que es denominado como *computación intensiva en datos*.

La computación intensiva en datos [5] utiliza los principios de la paralelización de los datos y de la computación distribuida para procesar enormes conjuntos de datos. Este tipo de computación es escalable con respecto a los recursos computacionales empleados, proporciona alta disponibilidad, tolerancia a fallos y alto rendimiento debido al uso de numerosos nodos de procesamiento con una alta tasa de acceso a disco, y es flexible a la hora de integrar otro tipo de tecnologías de procesamiento de datos. Actualmente, algunas de las técnicas y arquitecturas para la computación intensiva en datos [2,17] ya están siendo ampliamente explotadas para el desarrollo de aplicaciones para ciencia de datos.

Sin embargo, existe una gran dificultad en el uso de las técnicas de ciencia de datos por parte de muchos de los usuarios que las requieren en sus dominios de trabajo, como expertos en astronomía, medicina o biología, debido a la necesidad de poseer profundos conocimientos en áreas relacionadas con la informática como la minería de datos, computación paralela, computación distribuida o el desarrollo avanzado de software. Por este motivo se hace necesario proporcionar mecanismos que faciliten al experto, que no posee estos conocimientos avanzados en computación, la representación y definición de la secuencia de acciones necesaria para capturar la información de su dominio y convertirla en conocimiento sin tener que especificar los detalles de bajo nivel de ejecución.

El uso de flujos de trabajo [1] para la especificación de procesos de análisis de información y extracción de conocimiento proporciona un nivel de abstracción adecuado para este tipo de usuarios. Un flujo de trabajo permite capturar y modelar el conocimiento del experto como una secuencia de acciones o tareas que trabajan en coordinación para llevar a cabo un objetivo determinado, favoreciendo además la automatización de su ejecución por parte de los denominados *sistemas de gestión de flujos de trabajo* [18]. De esta forma, los aspectos de bajo nivel de computación son transparentes al usuario en este tipo de sistemas, incrementando su productividad y reduciendo el *time-to-value*.

Actualmente ya existen sistemas de gestión de flujos de trabajo en distintos campos de aplicación intensivos en datos, como bioinformática [11], minería de datos [8] o metaheurísticas [13]. Sin embargo, ninguno de estos sistemas formaliza la sintaxis abstracta del lenguaje con el que definen los flujos de trabajo, lo que conlleva la incompatibilidad e imposibilidad de reutilizar el conocimiento capturado de un sistema a otro. Existe la necesidad de estandarizar la definición de flujos de trabajo en dominios intensivos en datos para el desarrollo de herramientas interoperables, tal y como se indica en [14].

En este trabajo se presenta la formalización de la sintaxis abstracta de un lenguaje específico de dominio para el modelado de flujos de trabajo aplicados a ciencia de datos. Este lenguaje define un conjunto de elementos, relaciones y restricciones que permiten expresar a un alto nivel de abstracción la secuencia de tareas a ejecutar para realizar el proceso de análisis de la información y extracción de conocimiento en cualquier dominio de trabajo. De esta manera, también se favorece la posterior utilización de técnicas de transformación de modelos para alcanzar la interoperabilidad entre diferentes sistemas de gestión de flujos de trabajo. Además, para mostrar la aplicabilidad del lenguaje, se

introduce un ejemplo de uso donde se define un flujo de trabajo para el análisis de información en un dominio científico intensivo en datos, utilizando una sintaxis concreta definida a modo ilustrativo.

En el resto del artículo se presenta en profundidad el lenguaje específico de dominio propuesto. En la Sección 2 se introducen los antecedentes de este trabajo. En la Sección 3 se describe la formalización de la sintaxis abstracta del lenguaje. En la Sección 4 se muestra un ejemplo de uso donde se define un flujo de trabajo aplicado a ciencia de datos. Finalmente, en la Sección 5 se exponen las conclusiones obtenidas, destacando algunas líneas de trabajo futuro.

2. Antecedentes

Tradicionalmente, los flujos de trabajo han sido utilizados en otros ámbitos, como los procesos de negocio. La gestión de procesos de negocio [16] es un área relacionada con la especificación y mejora de los procesos existentes dentro de un entorno industrial, que utilizan distintos tipos de tecnologías para facilitar la coordinación y comunicación entre los diferentes agentes involucrados, pudiendo ser tanto agentes humanos como informatizados. WS-BPEL [9] es el lenguaje estandarizado por OASIS para la definición y ejecución de flujos de trabajo para la gestión de procesos de negocio como servicios web.

Sin embargo, la gestión de procesos de negocio tiene objetivos distintos a la computación intensiva en datos [7]. En la gestión de procesos de negocio, el foco recae sobre la orquestación de los recursos humanos y de los sistemas de información involucrados para alcanzar un objetivo determinado, mientras que la computación intensiva en datos se centra en la ejecución eficiente de una serie de actividades computacionales, optimizando todos los recursos disponibles e integrando distintos tipos de datos. Por tanto, estas diferencias provocan que no resulte adecuado reutilizar estos lenguajes en dominios diferentes para los que fueron desarrollados.

Para el caso de la definición de procesos basados en flujos de trabajo en áreas de computación intensiva de datos, como la ciencia de datos, no existe actualmente ningún lenguaje estandarizado para su especificación. Sin embargo, existen herramientas que internamente utilizan un lenguaje propio para la especificación de este tipo de procedimientos.

Taverna [11] es un sistema de gestión de flujos de trabajo que permite la definición, ejecución e intercambio de flujos de trabajo científicos. Proporciona acceso a numerosos servicios procedentes de distintas áreas, como bioinformática, astronomía o informática química, además de permitir la invocación de servicios genéricos que dispongan de un documento WSDL [15]. Por defecto, el orden de ejecución de los componentes que conforman los flujos de trabajo es derivado de las dependencias de datos existentes entre las distintas operaciones a ejecutar, en vez de ser definido por el usuario. No obstante, es posible establecer un orden de ejecución secuencial de forma explícita.

KNIME [8] es una plataforma de minería de datos que utiliza los flujos de trabajo como método para la modelar y ejecutar procedimientos de análisis de

datos. Proporciona numerosos tipos de algoritmos para la manipulación, visualización, análisis, y entrada y salida de datos, que principalmente son implementados utilizando Java. Sin embargo, es posible incorporar nuevos algoritmos desarrollados con lenguajes de programación como Java, R o Python. La ejecución de los flujos de trabajo es llevaba a cabo por un motor de ejecución que deriva el orden de los componentes del flujo de trabajo únicamente de las dependencias de datos existentes entre ellos.

También existen sistemas que aplican los flujos de trabajo sobre otro tipo de dominios, como el prototipo presentado en [13]. Esta herramienta proporciona un entorno gráfico para la definición y ejecución visual de flujos de trabajo para la computación evolutiva. Proporciona los elementos funcionales necesarios para la especificación de algoritmos evolutivos, como operadores de mutación, cruce y selección, codificación de individuos, etc. Además, facilita la creación de un marco de experimentación para validar los resultados obtenidos por los flujos de trabajo compuestos anteriormente, incluyendo distintos tests estadísticos. La funcionalidad de la herramienta puede ser extendida incluyendo nuevos algoritmos desarrollados en Java. El orden de ejecución de los componentes del flujo de trabajo es explícitamente establecido por el usuario, mediante líneas de conexión que establecen dependencias de control entre ellos.

Por tanto, es necesario formalizar la sintaxis abstracta de un lenguaje para la especificación de flujos de trabajo para ciencia de datos, de forma que se favorezca la extracción de conocimiento en todos aquellos dominios de trabajo que lo necesiten. Este lenguaje debe reunir las principales características de los sistemas que ya están utilizando flujos de trabajo para este propósito, con el fin de que pueda ser utilizado como base para alcanzar la interoperabilidad entre sistemas y para el desarrollo de nuevas herramientas en todo tipo de dominios intensivos en datos.

3. Sintaxis abstracta del lenguaje

Actualmente, ninguno de los lenguajes de flujos de trabajo existentes para computación intensiva en datos formaliza su sintaxis abstracta, lo que obliga a deducir un metamodelo a partir de una especificación informal. La sintaxis abstracta del lenguaje propuesto contiene las principales características y elementos comunes de estos lenguajes, lo que permite la interoperabilidad entre los distintos sistemas de gestión de flujos de trabajo. Además, se han incorporado nuevos elementos para satisfacer requisitos específicos de la ciencia de datos, como distintos proveedores de datos o metainformación sobre el contexto del flujo de trabajo.

La estructura del metamodelo propuesto se ha separado en capas para facilitar la comprensión, agrupando los elementos con funcionalidad similar. En la Figura 1 se ilustra dicha organización:

Capa estructural. Contiene todos los elementos necesarios para estructurar los flujos de trabajo en forma de grafo dirigido, donde los vértices representan los diferentes tipos de unidades funcionales a ejecutar y los arcos indican las distintas dependencias existentes.

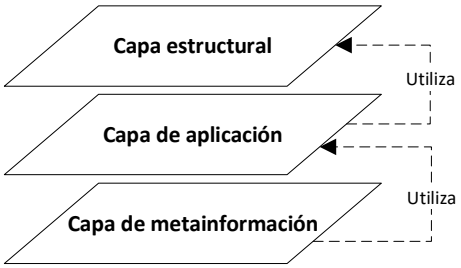

Figura 1. División por capas del lenguaje

Capa de aplicación. Agrupa los elementos que permiten capturar los requisitos específicos del dominio, como las tareas a ejecutar, los tipos de datos soportados, la definición de los recursos computacionales mínimos o la gestión de los errores de ejecución.

Capa de metainformación. Contiene los elementos que permiten asociar información acerca del contexto en el que ha sido desarrollado el flujo de trabajo o sobre los autores del mismo. Esta información está destinada a ser consumida por los propios usuarios, no a ser interpretada durante la ejecución del flujo de trabajo.

Obsérvese que no todos los elementos tienen que estar presentes en la definición de un flujo de trabajo con este lenguaje. Esto se debe a las particularidades de cada dominio y, también, a que algunos de los elementos podrían ser generados automáticamente por un sistema de gestión de flujos de trabajo o herramienta similar, por lo que podría omitirse su uso por el experto. El lenguaje puede ser extendido para adaptarlo a especificaciones concretas, ampliando la definición de determinadas metaclases, estereotipadas como «*extension point*».

3.1. Capa estructural

En la Figura 2 se muestran todos los elementos del lenguaje que, junto con sus relaciones, conforman esta capa. El lenguaje define un flujo de trabajo como un grafo dirigido (*ExecutionGraph*) que contiene un conjunto de nodos de ejecución (*Node*) y conexiones que expresan dependencias entre ellos (*DirectedEdge*). Esta estructura en forma de grafo se adecua a la especificación del proceso a modelar como una serie ordenada de acciones y favorece su comprensión por los usuarios.

Un nodo es un elemento con un nombre (*NamedElement*) que representa una unidad básica de ejecución y contiene un conjunto de puntos de conexión (*Endpoint*). Éstos le permiten interaccionar con el resto de nodos. Cada nodo admite distintos tipos de conexiones según los puntos de conexión que contenga, los cuales pueden ser para aceptar conexiones de control (*ControlEndpoint*), conexiones de datos (*DataEndpoint*) o conexiones de excepción (*ExceptionEndpoint*).

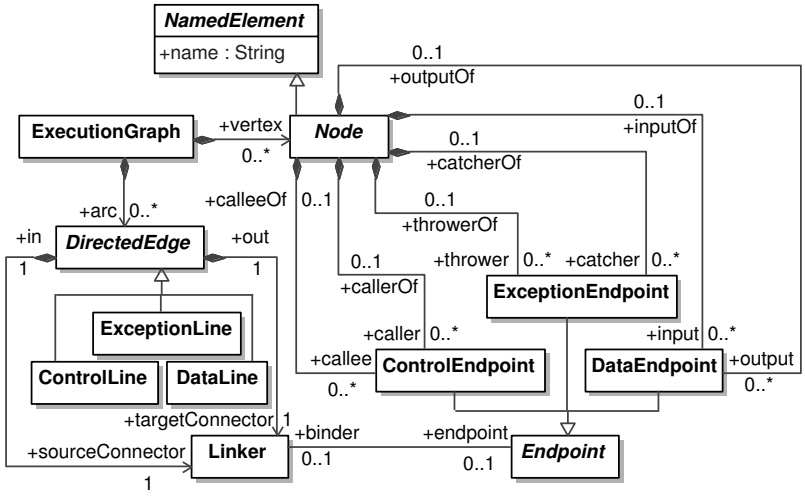

Figura 2. Metamodelo de la capa estructural

Una conexión permite establecer una relación de dependencia entre dos nodos distintos. Dependiendo del tipo de relación, una conexión puede expresar una dependencia de control (*ControlLine*) si la conexión determina el orden secuencial de ejecución entre ambos nodos, una dependencia de datos (*DataLine*) si especifica un intercambio de información entre los nodos, o una situación de excepción (*ExceptionLine*) si determina el nodo alternativo de ejecución en caso de que un error sea detectado. Un elemento vinculante (*Linker*) se encarga de relacionar cada nodo a través de los puntos de conexión, especificando tanto el nodo origen como el destino.

3.2. Capa de aplicación

En la Figura 3 se muestra la vista parcial de los principales elementos del metamodelo de esta capa de aplicación. Algunos elementos del metamodelo se omiten por motivos de espacio. Todo flujo de trabajo (*Workflow*) definido con este lenguaje está compuesto por una o varias unidades funcionales, denominadas *constructores* (*Constructor*), que permiten llevar a cabo un tipo de tarea específico. Un flujo de trabajo puede tener un orden de ejecución explícitamente definido por el usuario, o bien pueden ser las dependencias existentes entre los datos las que determinen el control de ejecución. También, un flujo de trabajo puede permitir que el orden de ejecución sea tanto explícitamente definido como derivado de las dependencias de los datos. Cada tipo de flujo de trabajo determina los tipos de constructores y conexiones admitidos.

Dependiendo de la funcionalidad proporcionada, un constructor puede ser definido como actividad (*Activity*), proveedor de datos (*DataProvider*) o estructura de control (*ControlStructure*). Una actividad representa la transformación

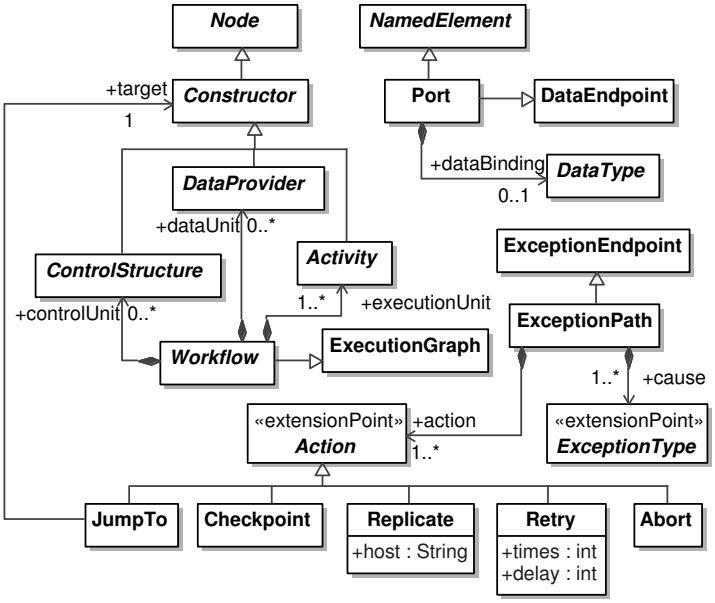

Figura 3. Metamodelo de la capa de aplicación

de un conjunto de datos de entrada para generar datos de salida. Las actividades pueden ser especializadas (Figura 4) como tareas computacionales (*ComputationalTask*) o de visualización (*DisplayTask*). Las tareas computacionales pueden ser ejecutadas sin necesidad de interacción por parte del usuario, tratándose tanto de procesos (*Process*) como de servicios (*Service*). Los tipo de procesos soportados son procesos Java (*JavaProcess*) para la ejecución de código escrito en Java, procesos CLI (*CLIProcess*) para ejecutar programas que utilicen una interfaz por línea de comandos, y flujos de trabajo previamente definidos con el propio lenguaje. No obstante, éstos pueden ser extendidos para referirse a otros lenguajes, plataformas y tecnologías.

También son soportados distintos tipo de servicios web (*WebServices*), como servicios SOAP (*SOAPClient*) y servicios REST (*RESTClient*). Al igual que los procesos, este tipo de servicios es extensible en el lenguaje. Además, las tareas computacionales permiten la definición de los recursos computacionales requeridos para su ejecución, como las características de la máquina específica donde ejecutar, el tipo de sistema operativo compatible o las características mínimas requeridas del procesador. Esta definición es compatible y puede ser extendida con el lenguaje estandarizado por OGF (Open Grid Forum) para este propósito [10]. Por otro lado, las tareas de visualización determinan cómo se deben mostrar gráficamente los datos gestionados en el flujo de trabajo. Este tipo de tarea puede establecer que la visualización de los datos sea delegada en una aplicación externa (*DelegatedDisplay*), o que sea integrada dentro de la propia aplicación que gestiona el flujo de trabajo (*EmbeddedDisplay*).

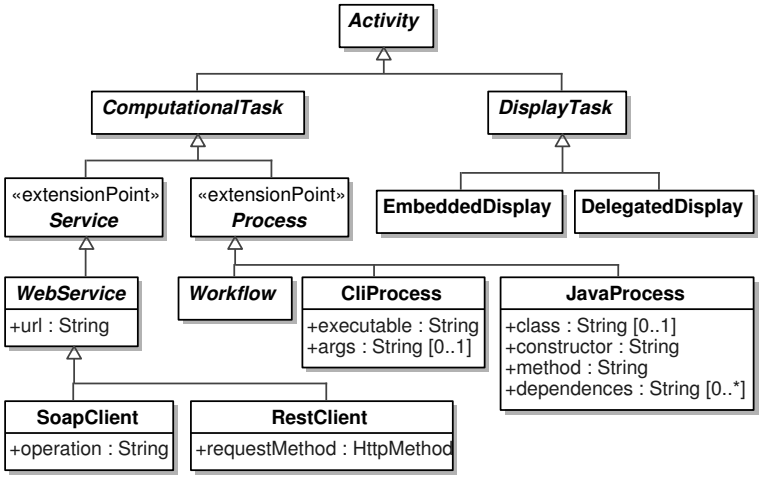

Figura 4. Vista parcial de las actividades de la capa de aplicación

Una estructura de control permite controlar explícitamente el orden de ejecución de los distintos elementos que componen el flujo de trabajo. Para ello, se proporcionan distintos tipos de estructuras de control que permiten indicar el punto de comienzo y de finalización del flujo de trabajo, establecer el siguiente constructor a ejecutar en base al cumplimiento, o no, de una expresión lógica, dividir el flujo de ejecución en dos o más flujos de ejecución paralelos que serán ejecutados de forma concurrente, unificar y sincronizar flujos de ejecución paralelos y, finalmente, contar el número de veces que el flujo de ejecución pasa a través de un elemento específico, lo que permite la realización explícita de bucles.

Por su parte, un proveedor de datos (Figura 5) define el lugar de origen y de destino de los datos que van a ser manipulados dentro del flujo de trabajo. Dependiendo del lugar donde los datos se encuentran almacenados, un proveedor de datos puede ser un registro (*Record*) que permite leer y escribir información en un bloque de memoria principal, un fichero (*File*) o una base de datos (*Database*).

Cada uno de estos constructores contiene una serie de puertos (*Port*), que permiten el envío y recepción de información entre ellos. Un puerto es identificado por un nombre y puede ser asociado con un tipo de dato (*DataType*) específico, limitando el tipo de información aceptada. El lenguaje proporciona una serie de tipos de datos básicos, que se corresponden con los tipos de datos primitivos de cualquier lenguaje de programación tradicional (valores enteros, booleanos o caracteres), y tipos de datos complejos, que se corresponden con estructuras de datos más complejas (vídeos, audios, imágenes o documentos XML).

Además, también son definidos elementos para la gestión de situaciones alternativas (*ExceptionPath*) cuando se producen errores inesperados en tiempo de ejecución (*ExceptionType*). Los tipos de errores soportados por el lenguaje permiten detectar si un tipo de dato es incompatible, si la memoria secundaria

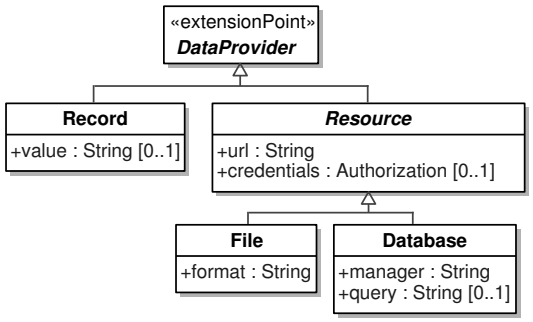

Figura 5. Vista parcial de los proveedores de datos de la capa de aplicación

está completa, si el permiso de acceso es denegado a un determinado recurso, si el recurso no es accesible temporalmente, si se produce un error genérico en tiempo de ejecución, si el tiempo de espera ha finalizado o si se produce un error de origen desconocido. Las acciones (*Action*) que se pueden establecer para gestionar dichas situaciones permiten continuar y ejecutar un constructor determinado (*JumpTo*), guardar el estado actual de la ejecución para continuar cuando el sistema se recupere (*Checkpoint*), intentar la ejecución en otro lugar indicando su dirección (*Replicate*), detener la ejecución y finalizar (*Abort*) o reintentar la ejecución un determinado número de veces máximo con un tiempo de espera específico entre cada intento (*Retry*). Tantos los tipos de excepción como las acciones de resolución de conflictos son extensibles en el lenguaje.

3.3. Capa de metainformación

En la Figura 6 se muestran todos los elementos que conforman esta capa del lenguaje. Cualquier flujo de trabajo tiene asociado una descripción (*WFSpecification*) que permite definir diversos aspectos relacionados con el contexto que ha propiciado su desarrollo. Este tipo de información no tiene que ser tenida en cuenta por los sistemas de gestión de flujos de trabajo, sino que está destinada a ser utilizada como documentación para el usuario.

Los elementos definidos permiten indicar información acerca del proyecto (*Project*) que ha impulsado la especificación del flujo de trabajo. Un experimento (*Experiment*) es un tipo de proyecto que se refiere a un procedimiento de estudio científico. Actualmente, se permite definir un experimento básico (*BasicExperiment*), que contiene la hipótesis científica (*Hypothesis*) a confirmar, y un conjunto de parámetros para definir la configuración y restricciones (*Configuration*) que deben ser consideradas durante su realización. La definición de un experimento científico es compatible y puede ser ampliada con el lenguaje SEDL presentado para este propósito en [12]. A su vez, un proyecto está compuesto por uno o varios participantes (*Stakeholder*), que puede ser tanto una organiza-

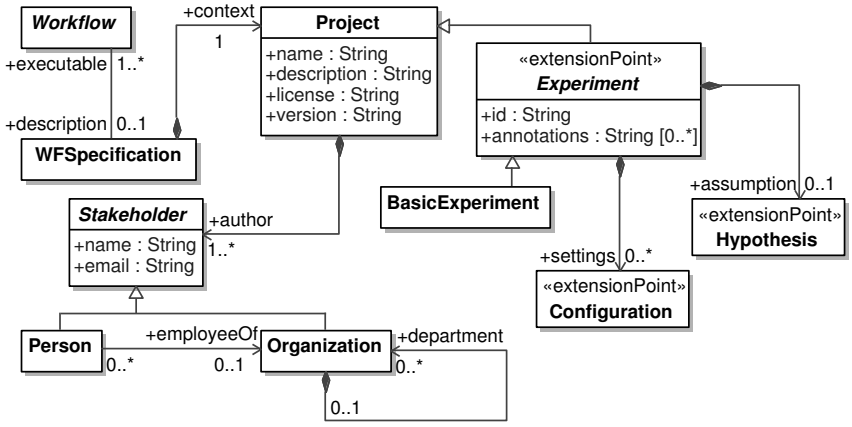

Figura 6. Metamodelo de la capa de metainformación

ción (*Organization*) compuesta por departamentos, como una persona (*Person*) asociada a una organización.

4. Ejemplo de uso

Para mostrar la aplicabilidad del lenguaje propuesto, se ha realizado un ejemplo de uso en el que se define un flujo de trabajo aplicado a ciencia de datos. Debido a que su principal objetivo es facilitar la especificación de procesos de extracción de conocimiento como una secuencia de tareas a ejecutar por parte de usuarios no expertos en computación, se ha propuesto una sintaxis concreta, a modo de ejemplo, que permite representar los constructores del lenguaje con una serie de elementos gráficos. No obstante, el lenguaje es independiente de cualquier sintaxis concreta y permite su representación tanto de forma gráfica como textual.

4.1. Especificación de la sintaxis concreta

La sintaxis concreta implementada a modo de ejemplo asocia una serie de figuras gráficas básicas a los principales constructores del lenguaje. Esta sintaxis está inspirada en la propuesta por UML para la representación de los diagramas de actividades, puesto que algunos de los constructores del lenguaje presentan una funcionalidad similar. En la Figura 7 se muestran los principales constructores con la representación gráfica asociada. Por simplicidad, el constructor *Activity* es representado utilizando la misma figura para cada una de sus distintas especializaciones. Los constructores *Begin*, *End*, *Conditional*, *Merge*, *Fork*, *Synchronizer* y *Counter* son especializaciones de la metaclase *ControlStructure*, omitidos del metamodelo por motivos de espacio. Esta sintaxis será utilizada

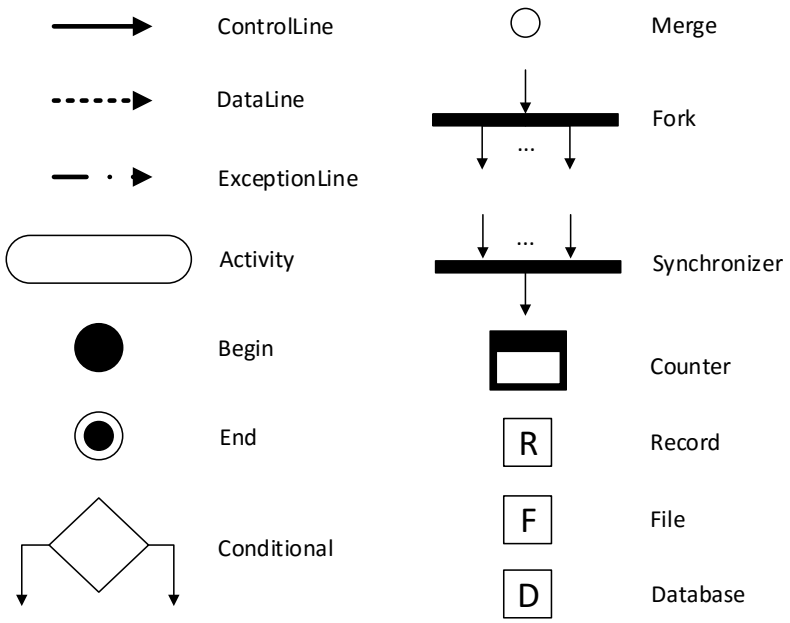

Figura 7. Representación gráfica parcial del lenguaje

para la definición de un flujo de trabajo para ciencia de datos explicado a continuación. Obsérvese que junto al elemento gráfico se ha indicado el nombre de la metaclase del lenguaje que representa.

4.2. Definición de un flujo de trabajo para ciencia de datos

La Figura 8 muestra un flujo para el análisis y extracción de conocimiento en el dominio intensivo en datos de la bioinformática, obtenido de un repositorio público de flujos de trabajo científicos [4]. Para su representación se ha utilizado la sintaxis concreta de la sección 4.1. Debido a que se trata de un flujo de trabajo dirigido por los datos, el lenguaje admite solamente la inclusión de constructores de tipo *Activity* y de tipo *DataProvider*, además de las líneas de tipo *DataLine* y *ExceptionLine*.

El objetivo del flujo de trabajo definido es encontrar posibles enfermedades en base a una serie de términos introducidos por el usuario. Para ello se realiza una consulta, a través de un servicio SOAP, a una base de datos de bibliografía médica a través de la actividad correspondiente (*Retrieve documents*). Esta actividad recibe como parámetros de entrada el número máximo de documentos que pueden obtenerse como respuesta a dicha consulta (*Max number of docs to retrieve*), una cadena con los términos que deben contener los documentos que se

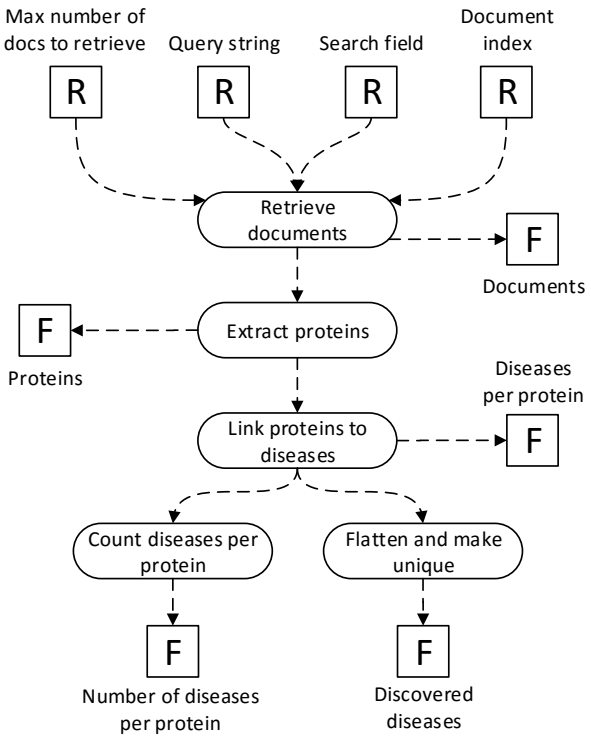

Figura 8. Ejemplo de flujo de trabajo para bioinformática

desean consultar (*Query string*), la parte del documento donde se van a buscar dichos términos (*Search field*) como, por ejemplo, en el título o en el contenido, y el nombre de la base de datos donde se va a realizar la búsqueda (*Document index*) como, por ejemplo, MedLine.

Una vez realizada esta consulta, se obtiene una lista con todos los documentos que cumplen los requisitos especificados. Dicha lista es almacenada en un fichero (*Documents*) y es utilizada por la actividad (*Extract proteins*) para el descubrimiento de proteínas. Esta actividad ejecutará un código escrito en Java para obtener una lista con todas las proteínas citadas en los documentos. Las proteínas descubiertas son almacenadas en un fichero (*Proteins*) y son manipuladas por una actividad (*Link proteins to diseases*) que permite asociar dichas proteínas a determinadas enfermedades. Esta asociación se hace mediante una invocación a un servicio SOAP, que accede a una base de datos de enfermedades. El resultado de esta invocación es una lista que relaciona las enfermedades posibles con cada proteína, la cual es almacenada en un fichero (*Diseases per protein*) y sirve como parámetro de entrada a las dos actividades siguientes.

Por un lado, una actividad (*Count diseases per protein*) utiliza un código escrito en Java para contar el número de enfermedades posibles asociadas a cada proteína. El resultado de esta actividad es almacenada en un fichero (*Number of diseases per protein*). Por otro lado, una actividad (*Flatten and make unique*) manipula la lista de entrada para generar un documento en un formato legible por los humanos con las enfermedades descubiertas. Este documento es también almacenado en otro fichero (*Discovered diseases*). Como se puede comprobar, el modelado del proceso de análisis de información permite la ocultación de los detalles de bajo nivel de ejecución. Este proceso es definido como un conjunto de tareas que realizan distintos tipos de procesamiento sobre grandes volúmenes de datos para extraer conocimiento útil para el experto.

5. Conclusiones y trabajo futuro

En este artículo se ha presentado la formalización de la sintaxis abstracta de un lenguaje para el modelado de flujos de trabajo que permiten la definición de procedimientos de análisis de información y extracción de conocimiento. Actualmente no existe ningún lenguaje de este tipo cuya sintaxis abstracta haya sido formalizada para este propósito. Los sistemas actuales que utilizan flujos de trabajo sobre este tipo de dominios intensivos en datos suelen definir su propio lenguaje de forma interna, con sus propias características, lo que provoca que los procedimientos desarrollados no puedan ser reutilizados en otros sistemas de similares características, ni que éstos puedan ser compatibles entre sí. Todo esto dificulta la labor del usuario, al estar limitada la interoperabilidad de sus flujos de trabajo por el lenguaje que implementa el sistema utilizado.

El lenguaje propuesto sirve como base para el desarrollo de nuevos sistemas basados en flujos de trabajo al proporcionar una formalización de los conceptos, relaciones y restricciones conforme a las reglas del dominio. Esta formalización permite lograr la interoperabilidad entre los distintos sistemas de gestión de flujos de trabajo mediante la transformación de modelos. No obstante, obsérvese que la sintaxis abstracta de estos lenguajes habrá de extraerse directamente de la representación concreta de los flujos de trabajo definidos ya que ninguno de los sistemas actualmente disponibles formaliza su sintaxis abstracta.

Además, para mostrar la aplicabilidad del lenguaje se ha presentado un ejemplo de uso. Primero se ha desarrollado una sintaxis concreta a modo ilustrativo. Luego, se ha definido un flujo de trabajo para bioinformática que permite descubrir enfermedades a partir de una serie de términos específicos. Para lograr este propósito, se especifican los distintos servicios y procesos que serán utilizados, estableciendo las dependencias de datos que existen entre ellos.

La línea a seguir en el trabajo futuro consiste en incluir nuevos tipos de constructores que amplíen la variedad de procesos y servicios soportados, como procesos desarrollados usando lenguajes de programación como R o Ruby, y el desarrollo de un catálogo de transformaciones de modelos que permitan lograr la interoperabilidad entre los distintos sistemas actuales de gestión de flujos de trabajo, como Taverna o KNIME.

Agradecimientos

Trabajo apoyado por el Ministerio de Economía y Competitividad, proyecto TIN2014-55252-P.

Referencias

1. Terminology & glossary. Tech. Rep. WFMC-TC-1011, Workflow Management Coalition (1999)
2. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
3. Fan, W., Bifet, A.: Mining Big Data: Current status, and forecast to the future. SIGKDD Explorations Newsletter 14(2), 1–5 (2013)
4. Goble, C.A., De Roure, D.C.: myExperiment: Social networking for workflow-using e-scientists (2007)
5. Gorton, I., Greenfield, P., Szalay, A., Williams, R.: Data-intensive computing in the 21st century. Computer 41(4), 30–32 (2008)
6. Loukides, M.: What is data science? O'Reilly radar (2010)
7. Ludäscher, B., Weske, M., McPhillips, T., Bowers, S.: Scientific workflows: Business as usual? Business Process Management pp. 31–47 (2009)
8. Mazanetz, M., Marmon, R., Reisser, C., Morao, I.: Drug discovery applications for KNIME: An open source data mining platform. Current Topics in Medicinal Chemistry 12(18), 1965–1978 (2012)
9. OASIS: Web services business process execution language version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf` (2007)
10. OGF: Job submission description language (JSDL) specification, version 1.0. `http://www.ogf.org/documents/GFD.136.pdf` (2008)
11. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M., Wipat, A., Li, P.: Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20(17), 3045–3054 (2004)
12. Parejo, J.A.: MOSES: A metaheuristic optimization software ecosystem. Ph.D. thesis, University of Sevilla (2013)
13. Salado-Cid, R., Luque, G., Romero, J.R.: Sistema de gestión de flujos de trabajo para la definición visual de aplicaciones basadas en algoritmos evolutivos. XVI Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA) (2015)
14. Salado-Cid, R., Romero, J.R., Ventura, S.: Metaherramienta para la generación de aplicaciones científicas basadas en workflows. X Jornadas de Ciencia e Ingeniería de Servicios (JCIS) pp. 96–105 (2014)
15. W3C: Web services description language version 1.1. `https://www.w3.org/TR/wsdl` (2001)
16. Weske, M.: Business process management: Concepts, languages, architectures. Springer Berlin Heidelberg (2012)
17. White, T.: Hadoop: The definitive guide. O'Reilly, first edn. (2009)
18. Yu, J., Buyya, R.: A taxonomy of workflow management systems for grid computing. Journal of Grid Computing 3(3), 171–200 (2006)