Datasheet Modeler: Una herramienta de soporte para el desarrollo de funcionalidades en LPS

Mailen Mancuso[1] *, Agustina Buccella[12], Alejandra Cechich[1], Maximiliano Arias[12], and Matias Pol'la[12]

[1] GIISCO Research Group
Departamento de Ingeniería de Sistemas - Facultad de Informática
Universidad Nacional del Comahue
Neuquén, Argentina
{mailen.mancuso,agustina.buccella,maximiliano.arias,matias.polla,
alejandra.cechich}@fi.uncoma.edu.ar
[2] Consejo Nacional de Investigaciones Científicas y Técnicas - CONICET

Abstract. Las líneas de productos de software (LPS) son un paradigma basado en el reuso definido mediante la especificación de aspectos comunes y variables para la creación flexible de productos dentro de un dominio en particular. En este trabajo, y dentro de esta misma línea, hemos definido un marco de desarrollo para dar soporte al diseño de funcionalidades, adaptadas a nuestra metodología LPS propuesta en trabajos previos. A su vez, hemos implementado una herramienta de soporte llamada *Datasheet Modeler*, la cual permite a los ingenieros de software diseñar funcionalidades dentro del marco definido. Por último, presentamos el diseño de una nueva funcionalidad en un contexto real.

Keywords: Líneas de Producto de Software, Gestión de Variabilidad, Herramientas de Software

1 Introducción

Las líneas de productos de software (LPS) [2, 8] han surgido como un paradigma de reuso muy útil cuando estamos analizando dominios particulares. El paradigma se sustenta en base a dos ingenierías: la de *dominio* y la de *aplicación*. La primera se basa en la definición de servicios o características comunes y variables que conformarán parte de una plataforma LPS. La segunda, en base a esta plataforma, permite personalizar los servicios variables y a su vez crear nuevos servicios, de manera de cumplir con las particularidades de cada organización. Dentro de ambas ingenierías, un aspecto importante es el denominado *gestión de variabilidad*, ya que engloba todas actividades necesarias para identificar, analizar, seleccionar, modelar, implementar e instanciar la misma dentro de la plataforma LPS.

* Este trabajo está parcialmente soportado por el proyecto UNCOMA F001 "Reuso Orientado a Dominios" como parte del programa "Desarrollo de Software Basado en Reuso".

En la literatura existen muchos trabajos relacionados con la variabilidad, los cuales proponen diferentes enfoques para dar soporte a las actividades involucradas en la misma. Por ejemplo, los trabajos presentados en [6, 7] analizan diferentes aspectos como el modelo utilizado, tipos, actividades del desarrollo LPS soportadas, etc. Dentro del modelado de la variabilidad, el cual es un aspecto en el que estamos interesados, las propuestas se enmarcan en el modelo de variabilidad soportado. Entre los más comunes, están: el modelo de características, el modelo de decisión, extensiones UML y el modelo ortogonal OVM (Orthogonal Variability Model) [7]. A su vez, varias de estas propuestas proveen herramientas de soporte para asistir al modelo y a las diferentes particularidades que se desprenden de los mismos. Por ejemplo, herramientas como XFeature[3] y fmp (Feature Modeling Plug-in)[4], proveen soporte para el diseño de variabilidades dentro del modelo de caractarísticas, y VarMod[5] y PLUM[6] implementan el modelo OVM.

En este trabajo estamos interesados en las tareas de modelado y diseño de las LPSs como parte de la ingeniería de dominio. Nuestra propuesta surge como una continuación de trabajos realizados previamente [3–5] para la definición de un proceso de desarrollo de LPSs basado en recursos semánticos, definidos como artefactos de software. En este trabajo presentamos el marco de desarrollo necesario para llevar a cabo el diseño de funcionalidades, junto con su herramienta de soporte que implementa el mismo. De esta manera, el diseño de funcionalidades se efectúa en una forma mas rápida y ágil, beneficiando así, al proceso de desarrollo en general.

El artículo se estructura de la siguiente manera. En la sección siguiente se describen los antecedentes de nuestra propuesta, enfocándonos en las bases de nuestra metodología y en los artefactos de software creados. Luego, en la Sección 3, se describe el marco de desarrollo en el cual se basa el diseño de funcionalidades en la plataforma LPS junto con la herramienta de soporte desarrollada. En la Sección 4 se ilustra dicho proceso mediante el desarrollo de una nueva funcionalidad dentro de una LPS. Por último se describen las conclusiones y trabajos futuros.

2 Antecendentes

En trabajos previos [3–5] hemos presentado una metodología para el desarrollo de LPS basada en niveles de dominios. En particular, la metodología define recursos semánticos mediante representaciones específicas enmarcadas dentro del enfoque de modelado OVM y en una implementación anotativa y composicional. Dichos recursos asisten a la comunicación entre los participantes y definen reglas para la construcción de artefactos de software en cada una de las etapas del desarrollo. En la Figura 1 podemos observar algunas de las actividades de la

[3] http://www.pnp-software.com/XFeature/
[4] http://gsd.uwaterloo.ca/fmp
[5] http://www.swpl.de/SEGOS-VM-Tool/index.html
[6] http://www.esi.es/plum/index.php

fase de ingeniería de dominio de dicha metodología, junto con los artefactos de software creados. La metodología posee más actividades que construyen otros artefactos de software necesarios para dar soporte a la ingeniería de dominio, los cuales pueden ser vistos en [5].

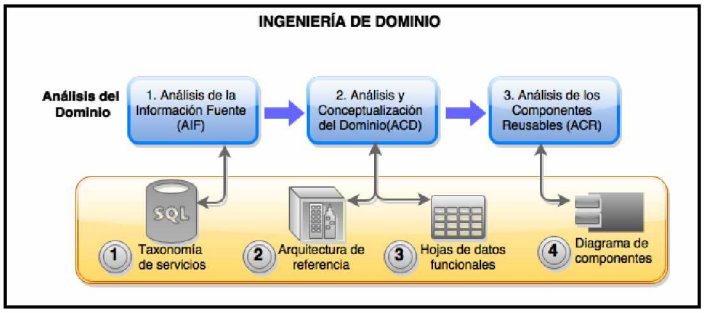


Fig. 1. Parte de las actividades involucradas en la ingeniería de dominio de las LPSs

En la parte inferior de la Figura 1 podemos observar los cuatro artefactos de software (numerados del 1 al 4) que deben ser desarrollados dentro de las tres actividades involucradas en el análisis del dominio. El primero de ellos es la taxonomía de servicios [4] creada en la actividad de análisis de la información fuente (AIF). La misma es una estructura jerárquica, la cual posee en los nodos superiores las categorías por las cuales clasificar los servicios. El segundo artefacto es la definición de una arquitectura de referencia como parte de la actividad de análisis y conceptualización del domino (ACD). Dicha arquitectura debe especificar una estructura preliminar para la interacción de los servicios definidos en la taxonomía. A su vez, en esta actividad se debe crear el tercer artefacto, las hojas de datos funcionales, las cuales poseen la especificación de cada funcionalidad. Por último, el cuarto artefacto es una estructura preliminar de componentes de software basados en la especificación previa de las hojas de datos funcionales. Dicha estructura debe crearse considerando la clase de servicios definidos (comunes, variantes y puntos de variación) y sus interacciones. La misma se realiza en la actividad de análisis de los componentes reusables (ACR).

Para dar soporte a la construcción de los artefactos mostrados en la Figura 1, hemos definido las pautas fundamentales para dar la construcción de los artefactos de software involucrados. Así, en [1] hemos desarrollado la herramienta llamada *Service Mapper*, la cual asiste a los ingenieros de software en la búsqueda de los servicios de la taxonomía más adecuados para cumplir con los requerimientos del domino (formulados por los usuarios expertos). Luego, en este trabajo describimos el marco de trabajo y los recursos definidos para la creación de las hojas de datos funcionales por cada funcionalidad de la LPS. A su vez, aquí también se crean los archivos XML que sirven para describir, en un formato

legible por una computadora, las relaciones y servicios definidos en cada una de las funcionalidades. Por último, estos archivos son leídos y analizados para crear la estructura de componentes preliminar que deberá ser analizada por los ingenieros de software y desarrolladores, para luego conformar la arquitectura de la plataforma.

3 Diseño de Funcionalidades en LPSs

Como hemos descripto en la sección anterior, el diseño de funcionalidades de la plataforma LPS se basa en la creación de las hojas de datos funcionales, las cuales poseen una serie de ítems para la especificación de las dependencias y variabilidades de los servicios de la taxonomía. Los ítems a completar dentro de cada hoja de datos son: la identificación o código de la funcionalidad con su descripción textual, el dominio donde la funcionalidad está incluida, la lista de servicios utilizados, un modelo gráfico que muestre la interacción de los servicios para lograr dicha funcionalidad y, finalmente, un conjunto de archivos XML que describen dicha interacción [5].

Para el modelo gráfico dentro de la hoja de datos definimos, a su vez, una serie de dependencias que nos permiten representar las interacciones posibles entre los servicios, junto con su variabilidad, las cuales se basan en una extensión del enfoque OVM. Las mismas son:

- *Uso* (<*MandatoryVP*>): especifica una dependencia entre servicios comunes que no necesariamente están asociados a puntos de variación.
- *Punto de Variación Obligatorio* (<*MandatoryVP*>): indica la obligatoriedad de seleccionar los servicios variantes.
- *Punto de Variación Opcional* (<*OptionalVP*>): indica que pueden seleccionarse cero o más servicios variantes.
- *Punto de Variación Alternativo* (<*AlternativeVP*>): indica que debe seleccionarse sólo un servicio variante (relación XOR).
- *Punto de Variación Variante* (<*VariantVP*>): indica que debe seleccionarse al menos un servicio variante (relación OR).
- *Requiere* (*dependency:Requires = "serviceName"*): especifica que si un servicio variante es seleccionado, requiere que otro servicio también lo sea, independientemente del punto de variación en donde estén asociados.
- *Excluye* (*dependency:Excludes = "serviceName"*): es el opuesto al anterior, en donde se excluye a un servicio variante si otro es elegido.
- *Punto de Variación Global* (<*GV*>): especifica que si se instancia el punto de variación de una manera específica, la misma será aplicada a todas las funcionalidades que contengan dicho punto.
- *Punto de Variación Específico* (<*SV*>): especifica que la instanciación de un punto de variación es particular a cada funcionalidad.

En la descripción previa, podemos observar la etiqueta XML utilizada para la creación de los archivos XML que describen la interacción de los servicios en el modelo gráfico. Estos archivos, que también son parte de la hoja de datos, son creados para garantizar un formato legible por la computadora y así permitir el

uso de herramientas de software en etapas posteriores. Los tres tipos de archivos XML creados a partir de cada una de las hojas de datos funcionales son: *información del servicio*, el cual especifica información por cada servicio involucrado como su identificación y nombre, entre otros; *interacción de servicios*, especificando todas las dependencias entre servicios del diseño en la notación gráfica de la hoja de datos; y *restricción de servicios* que posee las restricciones de *excluye* y *requiere*, cuando sea necesario.

3.1 Herramienta de soporte para el Diseño de Funcionalidades

Para diseñar funcionalidades de la plataforma desarrollamos una herramienta de soporte, llamada *Datasheet Modeler*, que permite a los ingenieros de software o diseñadores interactuar con los servicios de la taxonomía y con la arquitectura de referencia (artefactos 1 y 2 de la Figura 1) para definir las hojas de datos funcionales con todos sus ítems. La herramienta permite entonces, crear cada funcionalidad y a su vez generar los archivos XML, según las etiquetas definidas para las dependencias. En la Figura 2 podemos observar los componentes que conforman la herramienta.

El componente de *Definición de Hojas de Datos* es el responsable de solicitar al usuario los ítems generales por lo cuales está compuesta cada hoja de datos, siendo ellos: la identificación de la funcionalidad, el nombre y el dominio en donde estará incluida la funcionalidad. Como podemos observar en la Figura 3, la interfaz gráfica le permite al diseñador completar esos ítems mediante un formulario y además, la interfaz posee dos botones que le permiten al usuario realizar el diseño gráfico de la funcionalidad y luego, realizar la transformación XML para generar los archivos correspondientes al diseño.

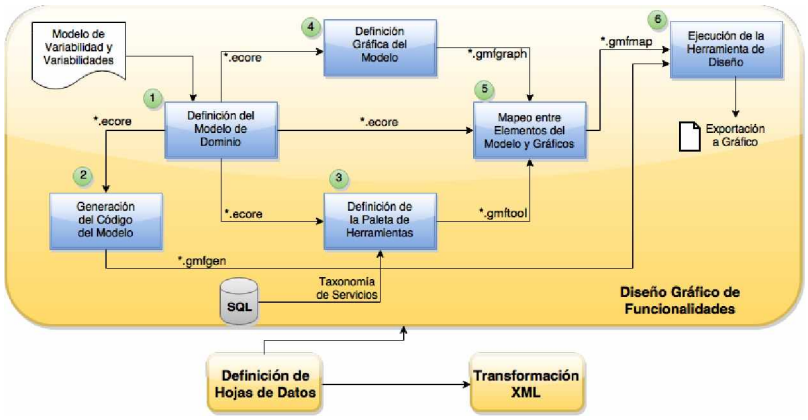


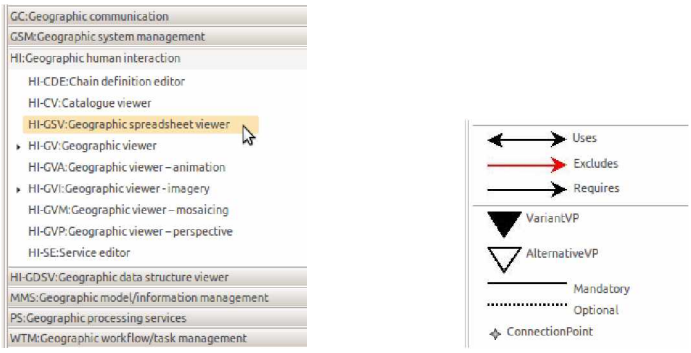
Fig. 2. Proceso para la creación de la herramienta de soporte *Datasheet Modeler*

Fig. 3. Interfaz gráfica para la definición de hojas de datos

Al presionar el botón *Diseño Gráfico* se invoca a los componentes involucrados en la herramienta de diseño gráfico en sí misma, la cual fue desarrollada utilizando el framework *GMF (Graphical Modeling Framework)*[7] provisto por la plataforma de desarrollo Eclipse[8], implementado en JAVA. En la parte superior de la Figura 2, podemos observar un conjunto de seis componentes que representan cada una de las etapas necesarias para cumplir con todos los requerimientos que implica el desarrollo del editor gráfico. Estas etapas son:

1. *Definición del Modelo*: Aquí se genera el *modelo de dominio base* para el editor gráfico. Todo lo que el modelo base permita y restringa, será exactamente lo que el asistente de GMF permita realizar para diseñar las hojas de datos. Así, considerando la naturaleza de los servicios de la taxonomía, las dependencias y variabilidades permitidas, definimos un modelo conceptual que incluye las abstracciones necesarias para interactuar con todos ellos. Como podemos observar en la Figura 3, la salida del componente es un archivo **.ecore* del cual se derivarán el resto de los subprocesos, generando así una serie de transformaciones de modelos para la implementación final de la herramienta.
2. *Generación del Código del Modelo*: Este componente genera un archivo con extensión **.gmfgen*, a partir del archivo **.ecore* anterior. Permite transformar automáticamente el modelo de dominio base, a su correspondiente código fuente. Este proceso se lleva a cabo mediante la aplicación de patrones de transformación, y el resultado es un conjunto de clases Java.
3. *Definición de la Paleta de Herramientas*: El tercer componente, que utiliza el archivo **.ecore*, se encarga de especificar los elementos que componen la paleta gráfica para diseñar las hojas de datos. En nuestro caso, definimos una serie de nodos para especificar los servicios y enlaces para sus dependencias, ya que todos ellos representan el comportamiento de las variabilidades entre los diferentes servicios. A su vez, aquí creamos las imágenes correspondientes a los elementos (íconos) de la paleta. De esta manera, la paleta contiene todos los recursos que se pueden utilizar para crear el modelo gráfico de las hojas de datos. La parte (b) de la Figura 4 muestra dicha paleta, según está implementada en la herramienta, con sus respectivas imágenes y descripciones. Este componente genera como resultado un archivo **.gmftool*.

[7] https://www.eclipse.org/gmf-tooling/
[8] https://www.eclipse.org



(a) Servicios de la Taxonomía (b) Variabilidades

Fig. 4. Paleta

A su vez, es importante resaltar que durante la ejecución de la herramienta (paso posterior) se podrá contemplar en la paleta, la taxonomía de servicios proveniente de una base de datos (Parte (a) de la Figura 4). Así, la paleta incorpora los servicios reales de forma dinámica, con los cuales el diseñador puede generar nuevas funcionalidades, siendo servicios actualizados y existentes en una base de datos previamente definida.

4. *Definición Gráfica del Modelo*: El cuarto componente permite especificar la definición gráfica del modelo de dominio base, a partir del archivo **.ecore*. Esto significa que, para cada uno de los elementos del dominio, habrá una manera particular de graficarlos. En concreto, aquí diseñamos las diferentes figuras que cada uno de nuestros elementos debe poseer. Así, por cada una de las primitivas definidas, realizamos su correspondiente figura. Este componente genera como resultado un archivo **.gmfgraph*.
5. *Correspondencia entre Elementos del Modelo y Gráficos*: En este componente se realiza el mapeo entre los resultados obtenidos por los componentes 1, 3 y 4. El mismo consiste en establecer, para cada elemento del modelo del dominio base, su respectiva representación gráfica en la paleta (según el archivo **.gmftool*) y su correspondiente definición gráfica (según el archivo **.gmfgraph*). Este componente genera como resultado un archivo **.gmfmap*. A su vez, en esta etapa se establecen las restricciones (OCLs) necesarias, limitadas a la etapa de diseño únicamente. Algunas de estas restricciones fueron agregadas para evitar valores nulos en varios atributos de las clases, especificar obligatoriamente servicios origen y destino para las dependencias, controlar que los puntos de variación estén correctamente definidos y no permitir ciclos en estas relaciones. Todas estas restricciones evitan que la transformación XML posterior produzca errores e inconsistencias en los modelos generados.
6. *Ejecución de la Herramienta de Diseño*: La ejecución del modelo **.gmfgen*, genera una plataforma Eclipse que permite poner en ejecución el editor construido, para realizar los diseños gráficos de las funcionalidades. Siempre y

cuando se respeten las reglas explicitadas en los pasos anteriores, el asistente de GMF se encargará de graficar cada uno de los elementos selectos de la paleta, según se hayan definido. Así, la paleta presentada en la Figura 4 es utilizada por el diseñador para generar las hojas de datos. Aquí, el diseñador podrá ver los servicios de la taxonomía mostrados en un árbol jerárquico y podrá seleccionar los que desee para diseñar sus funcionalidades, respetando las restricciones del modelo. Al finalizar con el diseño de cada funcionalidad, la herramienta permite exportar el modelo gráfico a un formato de imagen (png o gif).

Por último, una vez ingresados los ítems de una hoja de datos funcional y habiendo creado el modelo gráfico de la misma, la herramienta permite generar los archivos XML que describen la funcionalidad. Presionando el botón de *Transformación XML* (de la Figura 3) se invoca al componente del mismo nombre (Figure 2), el cual crea cada uno de los archivos. De esta manera, por cada funcionalidad el componente crea el archivo XML de *interacción de servicios*, el cual se genera a partir de la obtención de los datos relevantes del archivo generado automáticamente por GMF, como los servicios involucrados, sus dependencias, restricciones y variabilidades. También crea un archivo XML de *información del servicio* por cada servicio involucrado en la funcionalidad. Por último, el componente crea un archivo de *restricción de servicios* en caso de haber colocado restricciones de *excluye* o *requiere* entre servicios.

4 Diseñando una Nueva Funcionalidad

La herramienta de soporte para el diseño de funcionalidades fue aplicada para el desarrollo de una nueva funcionalidad, a fin de mostrar su funcionamiento. En particular, describimos el diseño de la funcionalidad *Análisis de Datos Biológicos* incluida dentro del dominio de Ecología Marina. Esta funcionalidad permite a los usuarios del producto final, seleccionar un conjunto de datos almacenados y elegir formas gráficas de visualización de los mismos, como por ejemplo, mediante histogramas, diagramas de tortas, etc.

En la Figura 5 se muestra una parte del modelo gráfico para la funcionalidad analizada. El ingeniero de software puede definir componentes arquitectónicos (en este caso capas de una arquitectura) y arrastrar servicios y sus dependencias desde la paleta hacia el editor de la herramienta. En la figura podemos ver que la paleta contempla los servicios (en la parte superior derecha), estructurados jerárquicamente según la taxonomía, y todas las posibles dependencias, restricciones y relaciones de variabilidad (en la parte inferior derecha) como vimos en la Figura 4. En este caso, el ingeniero ha ido seleccionando y relacionando una serie de servicios comunes y variantes, colocando también componentes arquitectónicos. Se han adicionado dos capas, "Human interaction" y "User Processing", y agregado el servicio PS-S5.1 (análisis de datos biológicos) que posee una dependencia de uso con el servicio HI-GDV (visor gráfico de datos). Éste último se define a su vez, como punto de variación de dos servicios variantes HI-GSV

(visor geográfico de hoja de cálculo) y HI-GCV (visor geográfico), el cual es también un punto de variación de tres servicios variantes HI-GCV1 (mostrar información como histograma), HI-GCV2 (mostrar información como gráfico de área) y HI-GCV3 (mostrar información como diagrama de torta).

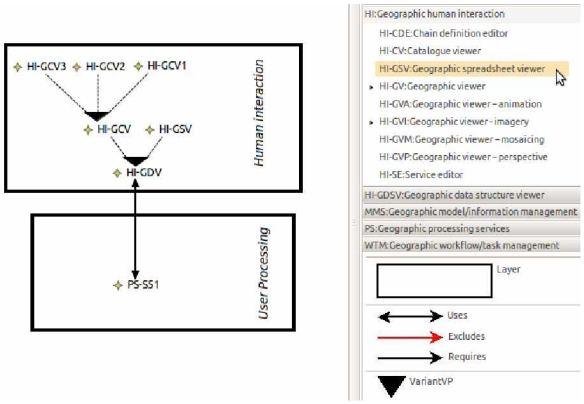


Fig. 5. Parte del diseño de la funcionalidad Análisis de Datos Biológicos utilizando la herramienta

Una vez finalizado el diseño de la funcionalidad y completado el resto de los datos de la hoja de datos funcional, generamos los archivos correspondientes mediante la transformación XML. Como producto de dicha transformación obtenemos al menos siete archivos que corresponden a cada uno de los servicios incluidos en la funcionalidad (Figura 5). A su vez, se genera también el archivo de interacción de servicios de la funcionalidad con todos los servicios que la integran y sus respectivas dependencias, restricciones y variabilidades. Este archivo puede observarse en la Figura 6. En este ejemplo, el proceso de transformación XML no creó el archivo de restricciones, ya que no hay ninguna restricción de requiere y/o excluye en la funcionalidad diseñada.

Como hemos descripto en la Sección 2 y específicamente en la Figura 1, estos archivos XML son los que se utilizan para crear la estructura de componentes preliminar (cuarto artefacto). La misma debe ser luego analizada por los ingenieros de software para conformar la arquitectura de la plataforma.

5 Conclusión y Trabajo Futuro

En este trabajo hemos presentado el marco de desarrollo necesario para el diseño de funcionalidades dentro de una plataforma LPS. El diseño de funcionalidades se basa principalmente en la creación de hojas de datos funcionales, las cuales se construyen en base a otros recursos semánticos ya definidos en trabajos previos de nuestra metodología de desarrollo de LPS [5]. A su vez, para dar soporte

Fig. 6. Archivo XML de interacción de servicios para la funcionalidad análisis de datos biológicos

al diseño de las hojas de datos funcionales, hemos presentado el desarrollo de una herramienta, como un plug-in de Eclipse, llamada *Datasheet Modeler*, la cual provee todos los recursos necesarios (taxonomía de servicios y dependencias de variabilidad) para crear cada una de las funcionalidades incluidas en una plataforma LPS. Como trabajo futuro continuamos extendiendo las reglas y procedimientos necesarios para completar las actividades de la ingeniería de domino definidas en nuestra metodología, para luego extendernos a la fase de ingeniería de la aplicación basada en los artefactos creados.

References

1. Arias, M., Renzis, A.D., Buccella, A., Cechich, A., Flores, A.: Búsqueda de servicios para asistir en el desarrollo de una línea de productos de software. In: Proceedings of the ASSE'15. Rosario, Argentina (2015)
2. Bosch, J.: Design and use of software architectures: adopting and evolving a product-line approach. ACM Press/Addison-Wesley Publishing Co. (2000)
3. Buccella, A., Cechich, A., Arias, M., Pol'la, M., Doldan, S., Morsan, E.: Towards systematic software reuse of gis: Insights from a case study. Computers & Geosciences 54(0), 9 – 20 (2013)
4. Buccella, A., Cechich, A., Pol'la, M., Arias, M., Doldan, S., Morsan, E.: Marine ecology service reuse through taxonomy-oriented SPL development. Computers & Geosciences 73(0), 108 – 121 (2014)
5. Buccella, A., Pol'la, M., Cechich, A., Arias, M.: A variability representation approach based on domain service taxonomies and their dependencies. In: International Conference of the SCCC'14. Chile (2014)
6. Chen, L., Muhammad, B.A., Nour, A.: Variability management in software product lines: A systematic review. In: Proceedings of the 13th International SPL Conference. pp. 81–90. SPLC '09, Pittsburgh, PA, USA (2009)
7. Galster, M., Weyns, D., Tofan, D., Michalik, B., Avgeriou, P.: Variability in software systems 2014 - a systematic literature review. Software Engineering, IEEE Transactions on 40(3), 282–306 (March 2014)
8. Klaus, P., Bŏckle, G., van del Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc. (2005)