A Reusable Software Architecture for Geographic Information Systems based on Software Product Line Engineering *

Nieves R. Brisaboa[1], Alejandro Cortiñas[1], Miguel R. Luaces[1], and Matias Pol'la[2,3]

[1] Databases Lab, University of A Coruña, Spain
[2] Giisco, Facultad de Informática, Universidad Nacional del Comahue, Neuquén, Argentina
[3] Concejo Nacional de Investigaciones Científicas y Técnicas
{brisaboa, alejandro.cortinas, luaces}@udc.es,
matias.polla@fi.uncoma.edu.ar

Abstract. In the last years there has been a continuous growth in functionality of geographic information systems (GIS) resulting in many different software artifacts. Even though each GIS is used in different areas with different objectives, they all share many features and requirements and therefore it is possible to apply techniques based on intensive software reuse, such as software product line engineering (SPLE). Although there has been much research on software product line engineering in the last years, the definition of a software product line for the domain of geographic information systems has not been undertaken.
In this work we identify the requirements and functionalities of a generic product for a web-based geographic information system, grouping them into commonalities that allow us to reuse many software artifacts, and variabilities that allow use to configure different products. Then, we define the functional and technological architecture of a software product line that uses current technologies for web-based application development. Finally, we design a tool to configure and assemble the components to generate the possible products. The resulting platform is flexible enough to adapt each product to the specific needs of each customer.

Keywords: geographic information systems, software product line engineering, general-purpose software architecture, variability management

1 Introduction

The field of Geographic Information Systems (GIS) has received much attention in the last years. Many disciplines such as cartography, biology, ecology, transportation and warehouse logistics use GIS to store, query and visualize

* Partially funded by MINECO ref. TIN2013-46801-C4-3-R (PGE & FEDER) and Xunta de Galicia ref. GRC2013/053 (FEDER) for authors in UDC.

geographic information. The improvements in communication technologies, together with the increased penetration of Internet access, have enabled the use of GIS technology with mobile equipment to visualize and manage data stored on remote computers accessible over the Internet. Clear examples of this trend are the success of applications like Google Maps, the inclusion of location-based functionality on a large number of web applications (e.g., Flickr, Facebook, Twitter), and the emergence of different types of GIS applications for municipal management, urban planning, tourism or property management.

Even though GIS applications have always had many common features and requirements such as storing and indexing geo-referenced data, displaying information as a set of layers, or grouping layers into different maps, the software artifacts used to implement the applications used different and incompatible conceptual, logical and physical data models (e.g., different definitions for the data type *polygon*, or different semantics for the predicate *overlaps*). Hence, it was very difficult to build interoperable applications because even the simplest task (e.g., data migration) was an arduous one. To solve this problem, a collaborative effort to define standards for GIS has been carried out by two organizations: ISO (through ISO/TC 211 and the 19100 set of standards) and the Open Geospatial Consortium (OGC). Nowadays, the application of techniques based on intensive software reuse such as software product line engineering (SPLE) is possible and relevant in the GIS domain. A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [12]. Software product lines enable systematic reuse in cases where there are families of products, i.e. similar products differentiated by certain characteristics. This new paradigm enables companies to improve the quality of software produced as well as reduce costs and launch times, thereby promoting the industrialization of software development.

Starting from the experience of our previous work in geographic subdomains such as marine biology [4, 5] or the development of GIS applications [11, 2] and geographic information retrieval algorithms [3], we have designed a software product line for GIS. The products of this SPL are web-based GIS that can be used to browse, query, analyze and manage geographic information. In Section 2 we present background concepts on SPLE. Then, in Section 3 we define the features of the product and we group them into components. After that, in Section 4 we define the platform architecture in the functional and technological levels. Then, in Section 5 we present the architecture of the tool that allows us to configure and assemble the software assets to generate the final products. Finally, future work and conclusions are discussed in Section 6.

2 Related Work

The traditional approach to software development performs the elicitation of requirements, design, implementation, testing and maintenance for each individual according to the specific needs of each customer. The disadvantage of this ap-

proach is that it requires high development and maintenance costs in order to produce high quality products. For this reason, new methodologies have emerged to apply mass-production and reuse strategies to the software development process. One such methodology is Software Product Lines Engineering (SPLE) [1, 12, 15], which focuses on separating the development of core, reusable software assets (i.e., the *platform*), and the development of the actual applications (i.e., the *products*). The platform is modeled as a set of *features* that represent a characteristic of a system relevant for some stakeholder (e.g., a requirement, a technical function or function group or a non-functional characteristic) and a set of *variation points* that represent the commonalities and variabilities of the different products. Individual products are modeled as a concrete selection of features and alternatives for the variation points in the platform. Finally, the product is built by adapting and assembling software artifacts from the platform.

The scope and range of products that a product line can deliver is determined by the flexibility of the platform, which in turn is determined by the the variability of the platform features. Therefore, variability management, which involves the tasks of identifying and defining the platform features, defining the functional and technological architectures of the product, and defining the product line configuration and derivation processes, is one of the main tasks in SPL development. There are many modeling techniques to identify and define the platform features, such as FODA (Feature Oriented Domain analysis) [8], FORM (Feature Oriented Reuse Method) [9], FM (Feature Modeling) [7], DM (Decision Model) [14], or OVM (Orthogonal Variability Model) [12]. There are also some approaches centered on the product architecture and the configuration and derivation process, which can be classified into annotative [10] (i.e., adding annotations in the source code to indicate the variant point and the different variants) or compositional (i.e., implementing the variants with different software assets). A metadata model that follows a mixed approach, taking into account the main advantages of both of them such as traceability (compositional) and fine-grained adjustments (annotative), has been defined in [13, 6].

Furthermore, we have presented in [4, 5] a methodology that extends the framework presented in [12] to create a product line in the marine ecology subdomain. This product line has been used in a case study that includes the work with biologists from two institutes and that resulted in the instantiation of two products within this subdomain. However, although there has been much research on software product line engineering in the last years, the definition of a software product line for the domain of geographic information systems has not been undertaken.

3 Identifying Features and Components for GIS Products

In this section, we identify and define all the functional and non-functional features for web-based geographic information systems analyzing several existing GIS applications with different scopes and features. Then, we determine which software assets are required by the platform to provide all the features identified.

3.1 Feature Variability

We use the variability management model presented in [13, 6]. It defines four types of variability for each feature (Mandatory (M), Optional (O), Alternative (A) and Variant(V)), and three types of dependencies between features (Uses, Requires and Excludes). The graphical notation is depicted on Figure 1.

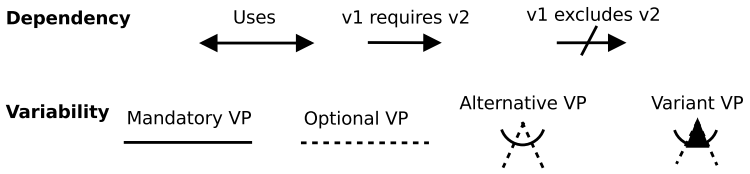

Fig. 1. Variability Management - Graphical Notation. Extracted from [6]

In addition to the variability types and dependencies described above, it is necessary to add a new element to the variability management model, called *variability scope*, which represents whether the scope of variability for a feature is *global* (notated as *VG*) and the variant is chosen once and applied every time the functionality is selected, or the the scope is *specific* (notated as *VS*) and a concrete variant must be chosen every time the functionality is selected. For example, the architecture selection is *global* feature because it only has to be selected once and it is applied in all variation points. However, the visualization type of map viewers is a *specific* feature because an application can have multiple map viewers with some of them being *embedded* within a web page, some others being displayed in a *full page*, and some others being *detachable* (i.e., they start embedded but they can also be displayed in a full page).

Id	Feature	M	O	A	V	Scope
1	Architecture selection	•				VG
2	Data model definition	•				VG
3	Maps definition	•				VG
3.a	Layers definition	•				VG
3.b	Styles definition		•			VG
4	Map viewers	•				-
4.a	Visualization type	•				VS
4.b	Geographic scope	•				VS
4.c	Map viewer tools		•			VS
5	Geodata edition		•			VG
6	User management		•			VG
...						

Table 1. Table Feature Variability - Subset

In Table 1 we show a subset of the features identified (the full table includes more than 90 features). We have grouped them into different variant points according to their function and we have established the variability type and scope of each one. From the various GIS applications studied, we have not only defined functional features like *user authentication*, but we have also defined many non-functional features such as *map viewer library selection*. While the former will turn out in new functionalities added to the product, the latter only represents a design decision that changes the technology used to view the maps in the web application. Some of the main features identified in our platform are the following:

- **Architecture selection.** The technology used for some functions of the products may vary (e.g., the DBMS or the map visualization library). We give more detail regarding this feature on Section 4.
- **Data model definition.** Each product may have an specific data model that must be defined as the first step in the configuration tool, as we can see on Section 5.
- **Managing maps, layers and styles.** Each product may define its own collections of map layers, visualization styles, and maps (i.e., a named collection of ordered map layers with default styles). Furthermore, another feature enables users of the product to manage them on runtime. The definition of layers and maps is a mandatory feature because it has no sense to build a GIS application without them.
- **Configuring the map viewers.** Every GIS application has one or more map viewers, each one with its own configuration. Therefore, the scope of these features is specific. Some of the features in this variation point are the selection of the visualization type (i.e., the map viewer may be embedded in some other content or shown in full page mode) or the selection of the different tools that can be enabled (zooming and panning the map, opening modal views with info of the selected elements, getting the permanent link of the current view, etc.).
- **Editing the geographic data.** We can optionally enable the edition of the geographic data on our GIS product. This allows the user to import data from shapefiles or a WMS service, or even update the geographic data directly on a map viewer.
- **Enabling user and authentication management.** As in any web-based system, we can control the people who has access to the application and to each feature of it. We can see this feature is optional, since we may want the application to be totally open.

In Table 2 we present a subset of the dependencies that we have found between each feature. For example, if we choose not to use a *Map Cache Server* in our application, we force the layers to be generated each time by our *Map Server*. For example, feature *Maps definition* (3) has a *requires* dependency with the feature *Layers definition* (3.a), and this one has a *uses* dependency with the feature *Styles definition* (3.b). Also, the *Map viewers* feature has a *requires*

Id	Name	Use	Requires	Excludes
...				
3	Maps definition		3.a	
3.a	Layers definition	3.b		
3.b	Styles definition			
4	Map viewers	4.c	3, 4.a, 4.b	
4.a	Visualization type			
4.a.1	Embedded map			
4.a.2	Full page map			
4.a.3	Switching type allowed		4.a.1, 4.a.2	
4.b	Geographic scope			
4.c	Map viewer tools			
...				

Table 2. Table Feature Dependencies - Subset

dependency with *Maps definition*, *Visualization type* and *Geographic scope*, and a *uses* dependency with *Map viewer tools*. Furthermore, the subfeature of *Visualization type*, *Switching type allowed*, has a *requires* dependency with both subfeature *Embedded map* and *Full page map*.

After defining all dependencies and variabilities, the relationships and constraints between features can be shown graphically in a variability model. Figure 2 shows the features described in Table 2. In the model we can see all variation points (places where the variability occurs) represented by light gray squares with the annotation VS or VG, according to their scope.

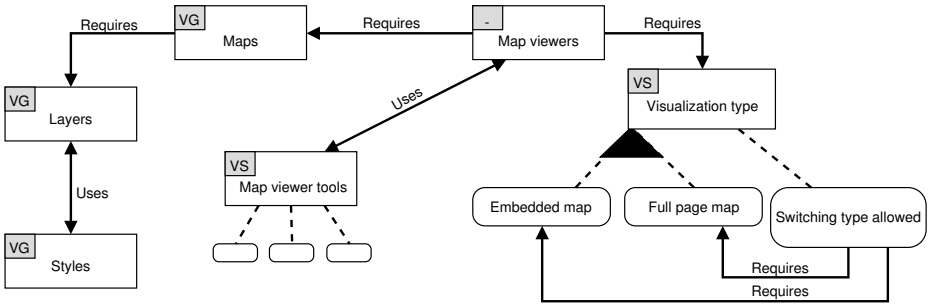

Fig. 2. Portion of the variability model

3.2 Functional Components

Starting from the collection of features identified and defined in the variability model described above, we have identified and defined a set of software components that will implement all the possible features of the resulting products. The

components implementing any mandatory variant are called *core components* and they will be included in all the generated products.

- **Data model.** The software engineer that uses the SPL to create a product must provide a complete definition of the GIS application data model. The software engineer may use alphanumeric and geographic data types, relationships between entities with concrete cardinality, and simple restrictions like the non-nullity of any element. This component is in charge of implementing the data model in a specific architecture.
- **Data access.** This component is responsible for all the functionality that involves creating, reading, updating and deleting elements from the data model.
- **Map viewer.** This component implements the functionality related to viewing and interacting with geographic information using maps.
- **Data viewer.** The functionality related to browsing and interacting with text-based data (e.g., lists, editing forms, etc.) is implemented by this component.
- **Menu.** Nearly all web applications have a menu component to access all sections and features thereof. Thus, our generated GIS application should have also this component.
- **Database management system.** The DBMS chosen will affect many other components, especially in the low level layers (Data model and Data access components). The configuration of this component will include the definition of the database, as well as the credentials used to connect to it. We provide various options to choose between them.

The rest of the components are called *optional components* and may be or not in the final products. Some of these components are totally isolated from the rest, but there are many functionalities that require certain relationship and interoperability between them.

- **User manager.** Some GIS applications will require authentication, different user roles and the common functionality provided for any user-related web product.
- **Map manager.** Maps, layers and styles are defined during the configuration of the product. Furthermore, this component allows the product administrator to manage these elements on runtime. Otherwise, the initial configuration will be static.
- **Map data importer.** If the GIS produced should be able to import new geographic data (i.e., from shapefiles), this component will allow it. The new data can be added from different sources, besides the mentioned one.
- **Map data exporter.** Component used to export any to map to several type files, like PDF or PNG. It also allows the user to print the visualized maps.
- **Map server.** Most GIS applications use a map server to produce the cartography images. We have decided to make this component optional because

a map server is particularly costly in terms of hardware and configuration. Therefore, some GIS applications may use an internal map server to draw cartography whereas other applications may rely on external map servers (e.g., OpenStreetMap) and display all geographic elements only at the client side.

- **Map cache.** Some GIS applications that require an internal map server may not require the cartography to be generated in real-time. Therefore, this component implements a map cache so that static maps are displayed instead of being generated each time they are required.
- **Static pages manager.** Not all the content displayed on the GIS application can be generated automatically. Some content may depend on specific and product-related information. To provide this flexibility on the user interface, we have defined a component that handles the creation of customized content using static web pages. These pages will be able to use other components of the SPL (such as embedded maps) to enrich the interface.

4 Platform Architecture for GIS Products

The software architecture plays an important role in any information system. Particularly, in SPL the definition of the platform architecture is a major task because it must support the entire product range and scope previously defined. For this reason, the platform architecture must be general and flexible enough to capture both commonalities and variabilities. On the other hand, the platform architecture must be concrete and efficient in order to support the creation of products that can be used in real-life problems.

In this sense, we have defined a three-layer architecture composed of a user interface, responsible for the interaction with the user; a processing layer which contains all the functionality defined for the GIS; and a model management layer, responsible for physical data storage and data management. In Figure 3 we have classified the major components described in Section 3.2 according to the layer where their activity takes place. We can see that there are three types of components: *user interface-layer* components, *data-layer* components and *transverse* components, which belong mostly to the processing layer but that also affect the other two.

Regarding the technological architecture, web-based GIS applications usually have simpler or less functionalities than desktop-based ones. However, thanks to the late improvements in communications, server-side and client-side processing, and web-based frameworks, we believe that a web-based GIS product can be built. Figure 4 presents the platform architecture in terms of technology. It also shows a feature model of the architectural variation points, which are annotated with the legend *VG* and all the variants associated to each one are represented with light gray boxes. We have decided to define only five variation points to avoid an unmanageable number of alternatives that add unnecessary complexity.

In the user interface layer we have decided to use an open-source web application framework called AngularJS because it presents a modular design which

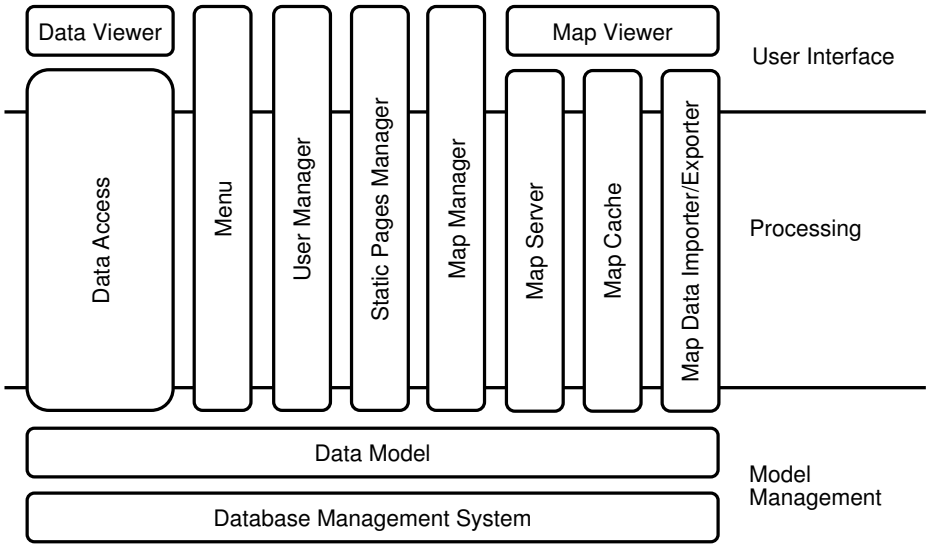

Fig. 3. Platform architecture with functional components

enables us to define a flexible configuration and to implement the different variants with different software artifacts. In this layer, we have also decided to define a variant point with two alternatives to visualize the geo-referenced data in the user interface: OpenLayers and Leaflet. These libraries have different scopes and therefore provide different advantages to the final product.

The processing layer is based on Spring MVC. The REST services used to communicate the user interface with the processing layer are implemented with Spring controllers, and the services that provide communication with the model management layer are built using the dependency injection pattern of Spring. We also use Spring Security to support user authentication and access-control. Finally, as described in Section 3.2, the functional architecture defines a variation point to include an internal map server (GeoServer) and a map cache (TileCache).

Finally, the model management layer a variant point with two alternatives for the data access technology: Java Database Connectivity (JDBC) or Hibernate. This layer also has a variant point with three alternatives for the Database Management System: PostgreSQL and PostGIS, MySQL or Oracle Spatial.

5 Configuration and Derivation Tool for a SPL in GIS

The process for the generation of products in a software product line consists of two main activities: *product configuration*, where the selection of the desired features occurs; and *product derivation*, where the final product is assembled. Figure 5 presents the tool that we have designed that enables both tasks. The

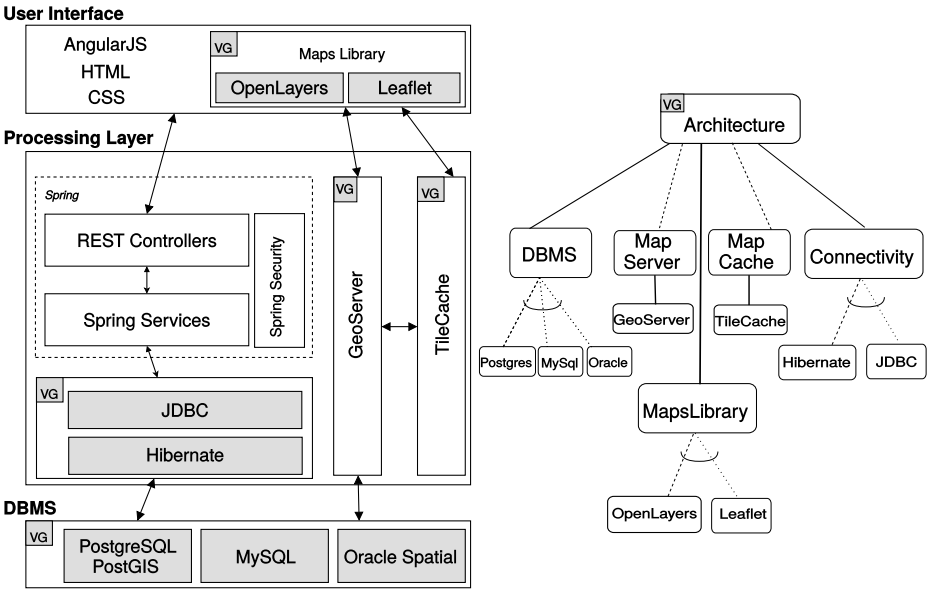

Fig. 4. Technological platform architecture and variability model

left side of the figure shows all the components of the product configuration tool. On the right side of the figure we show the product derivation tool that receives the product configuration and assembles the final product using the software assets from the asset repository. The configuration of each product is stored in the configuration repository and can be loaded in the tool in any moment, so we can derive the exact same product again or update a product starting from a previous configuration. We also maintain version control over the products and their evolution.

The configuration tool is composed of several modules that allow us to tailor the product to the user's needs. In this task, all the variability present in the shared platform should be instantiated according to specific needs within the defined range. For this, first the data model of the application must be provided. Once the tables are described, the set of maps that provide the geographical context as well as the layers and the styles must be described. Then, the product features must be selected in the variability configuration and feature selection module. The module receives as input the variability model in XML format as defined in [6]. This model is composed of different tags that represent the variabilities, constraints and relationships defined in Section 3. Then, the module generates an instance of the model with the functionality and actions selected by the software engineer. This module also controls the validity of the selection of features, according to the constraints and dependencies in the variability model. In addition to the selection of features, the software engineer can optionally provide static content such as text and pages in the applications, as well as the

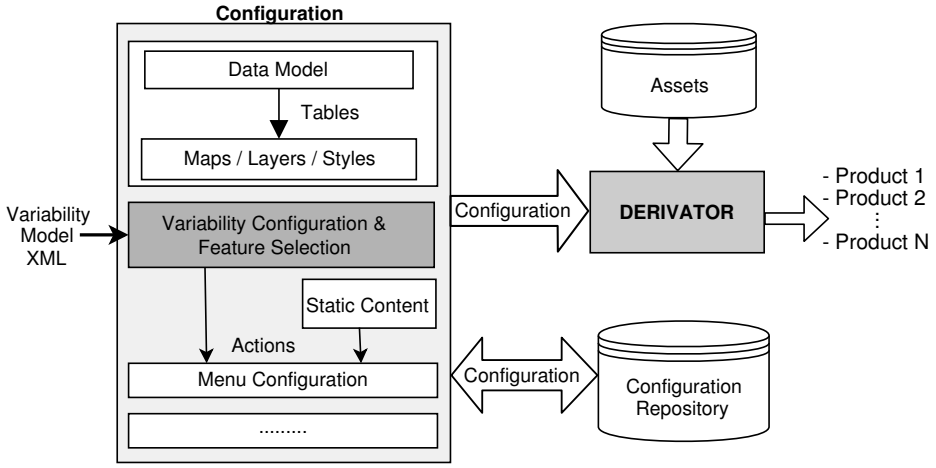

Fig. 5. Configuration and Derivation Tools

details of the web aspect. The software engineer can also configure the application menu using the menu configuration module.

The input of derivation tool is the configuration generated with the configuration tool in the previous step. The task of the derivation tool is performed without user interaction. First, the configuration is parsed and after that the product is built using the software assets extracted from the asset repository.

6 Conclusion and Future Work

In this paper we have presented a software product line for web-based geographic information systems. We have first identified and defined functional and non-functional requirements for such systems based on the analysis of the characteristics of a set of GIS applications. Then, we have defined a platform for these products as a set of features, their variation points, and their restrictions following the model introduced in [6]. After that, we have grouped the features identified into two set of components: core and optional. This allowed us to define the functional and technological architecture of the platform taking into account the current trends in web-based technology for GIS applications. Both are highly-modular, which facilitates the assembly of products and their maintainability. Finally, we have designed the configuration and derivation tools that are used to build the final products.

Regarding future work, we are currently implementing all the components, working with three different products to validate and evaluate the benefits of the software product line. We are also testing and analyzing other technologies such as Yeoman, in addition to those described in [13, 6], in order to be able to deal with the high level of variability proposed for the software product line.

References

1. J. Bosch. *Design and use of software architectures: adopting and evolving a product-line approach*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.
2. N. R. Brisaboa, J. A. C. Lema, A. Fariña, M. R. Luaces, J. R. Parama, and J. R. R. Viqueira. Collecting and publishing large multiscale geographic datasets. *Softw., Pract. Exper.*, 37(12):1319–1348, 2007.
3. N. R. Brisaboa, M. R. Luaces, A. S. Places, and D. Seco. Exploiting geographic references of documents in a geographical information retrieval system using an ontology-based index. *GeoInformatica*, 14(3):307–331, 2010.
4. A. Buccella, A. Cechich, M. Arias, M. Pol'la, S. Doldan, and E. Morsan. Towards systematic software reuse of gis: Insights from a case study. *Computers & Geosciences*, 54(0):9 – 20, 2013.
5. A. Buccella, A. Cechich, M. Pol'la, M. Arias, S. Doldan, and E. Morsan. Marine ecology service reuse through taxonomy-oriented SPL development. *Computers & Geosciences*, 73(0):108 – 121, 2014.
6. A. Buccella, M. Pol'la, A. Cechich, and M. Arias. A variability representation approach based on domain service taxonomies and their dependencies. *XXXIII International Conference of the Chilean Society of Computer Science (SCCC'14). Talca, Chile. IEEE Computer Society Press*, November, 08-14.
7. K. Czarnecki, P. Grunbacher, R. Rabiser, K. Schmid, and A. Wäsowski. Cool features and tough decisions: a comparison of variability modeling approaches. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 173–182, New York, NY, USA, 2012. ACM.
8. K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University Pittsburgh, PA., 1990.
9. Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5:143–168, January 1998.
10. Christian Kästner, Sven Apel, and Martin Kuhlemann. Granularity in software product lines. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 311–320, New York, NY, USA, 2008. ACM.
11. A. S. Places, N. R. Brisaboa, A. Fariña, M. R. Luaces, J. R. Paramá, and M. R. Penabad. The galician virtual library. *Online Information Review*, 31(3):333–352, 2007.
12. Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
13. M. Pol'la, A. Buccella, A. Cechich, and M. Arias. Un modelo de metadatos para la gestión de la variabilidad en líneas de productos de software. In *Proceedings of the ASSE'14: 15th Simposio Argentino de Ingeniería de Software*, Buenos Aires, Argentina, 2014.
14. K. Schmid, R. Rabiser, and P. Grünbacher. A comparison of decision modeling approaches in product lines. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 119–126, New York, NY, USA, 2011. ACM.
15. Frank van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.