Software Product Line for web-based Geographic Information Systems

Autor: Alejandro Cortiñas Álvarez

Tesis doctoral UDC / 2017

Directores:
Miguel Rodríguez Luaces
Óscar Pedreira Fernández

Programa Oficial de Doctorado en Computación

PhD thesis supervised by
Tesis doctoral dirigida por

Miguel Rodríguez Luaces
Departamento de Computación
Facultad de Informática
Universidade da Coruña
15071 A Coruña (España)
Tel: +34 981 167000 ext. 1254
Fax: +34 981 167160
luaces@udc.es

Óscar Pedreira Fernández
Departamento de Computación
Facultad de Informática
Universidade da Coruña
15071 A Coruña (España)
Tel: +34 981 167000 ext. 6028
Fax: +34 981 167160
opedreira@udc.es

Miguel Rodríguez Luaces y Óscar Pedreira Fernández, como directores, acreditamos que esta tesis cumple los requisitos para optar al título de doctor internacional y autorizamos su depósito y defensa por parte de Alejandro Cortiñas Álvarez cuya firma también se incluye.

A mi familia.

Acknowledgements

First of all, thanks to Miguel, the person who has introduced me to researching and who has invested more time and effort in helping me for many years. Thanks also to Oscar for his help since I arrived at the laboratory and throughout my thesis. I am also grateful for the support and the welcoming of all my colleagues at the Databases Laboratory of the University of A Coruña, especially to Nieves, who has always advised me and has taken care of me, without her help I would not be where I am. My fellow doctoral students, Fernando, Cristina, Adrián, Tirso and even Daniil deserve a special mention, thanks for making this whole path much more pleasant. I also want to thank my colleagues from Enxenio, in special to Carlos, Santi and Alejandro, for their collaboration throughout this work. And to all of you who gather to have lunch every day, sometimes I feel like working just to chat with you.

Besides I would like to thank my friends from Como, Mehdi, Theo, Shruti, Melisa and all the others for having made those three months flown by. Thanks to all those people of CUAC that have been standing me for many years and my colleagues in Spoiler (*talking about series... seriously*). Thanks to Vanisa for those remote talks that help me to disconnect. Thanks to the *hillbillies* I have for friends, especially Pata, Seo and Shorkan, who are always there bothering (was it the other way around?). Thanks to Antonio, who finally finished his project.

Lastly, I am very grateful to my family, who are not to blame for anything and who have always trusted and supported me.

Agradecimientos

Ante todo gracias a Miguel, la persona que me ha introducido en la investigación y que más tiempo y esfuerzo ha depositado en mí desde hace ya muchos años. Gracias también a Óscar por su ayuda desde que llegué al laboratorio y a lo largo de mi tesis. Me gustaría agradecer también el apoyo y acogida de todos mis compañeros y compañeras del LBD, especialmente a Nieves, que siempre me ha aconsejado y ha tirado de mí, sin su ayuda no estaría donde estoy. Mención aparte merecen mis colegas doctorandos, Fernando, Cristina, Adrián, Tirso y hasta Daniil, gracias por hacer todo este camino mucho más agradable. También quiero agradecer a mis compañeros de Enxenio, especialmente a Carlos, Santi y Alejandro, por su colaboración a lo largo de este trabajo. Y a todos los que nos reunimos para comer cada día, que a veces apetece trabajar sólo para charlar con vosotros.

Por otro lado me gustaría dar las gracias a mis amigos y amigas de Como, a Mehdi, Theo, Shruti, Melisa y a todos los demás por haber hecho que esos tres meses hayan pasado volando. Gracias a toda esa gente de CUAC que me lleva aguantando ya muchos años y a mis colegas de Spoiler (*hablamos de series... en serio*). Gracias a Vanisa por esas charlas en remoto que me ayudan a desconectar. Gracias a los garrulos que tengo por amigos, sobre todo a Pata, Seo y a Shorkan, que siempre están ahí dando la tabarra (¿o era al revés?). Gracias a Antonio, que por fin se sacó el proyecto.

Por último, muchas gracias a mi familia, que los pobres no tienen culpa de nada y siempre han depositado toda su confianza y su continuo apoyo en mí.

Abstract

Software Product Line Engineering (SPLE) is a research field that seeks to industrialize software development using techniques such as mass-production and mass-customization or reusing software components. A geographic information system (GIS) is an information system that works, in some way, with geographic information. Although each GIS is used in a particular area, there are many features common to all of them. In addition, strong standardization has been carried out so that most GIS software components are interoperable. Consequently, the application of SPLE in this domain is a feasible and interesting problem.

Applying SPLE to a new domain is a complex process and, in order to guarantee the validity of the final design of the SPL and its evolution, it is important to strictly follow a methodology appropriate to the specific domain. Considering that it does not exist a suitable methodology for the context of our work (i.e., web-based GIS applications developed in a software company with several products in the market), we have decided to combine several existing methodologies and extend their scope with additional tasks that will are very useful in our context.

After defining our SPL following this methodology, we found that the traditional techniques to implement SPL are not suitable for our domain, due to the peculiarities and requirements in the development of web-based GIS applications. Therefore, we have defined and implemented a new derivation engine for automatic software generation that maintains the formalities behind SPLE but at the same time provides a new degree of flexibility thanks to the use of a well-known industrial technique: scaffolding.

Resumen

La ingeniería de líneas de producto software (LPS) es un campo de investigación que pretende industrializar el desarrollo de software usando técnicas como la producción y customización en masa, o la reutilización de componentes software. Un sistema de información geográfica (SIG) es un sistema de información que trabaja, de alguna manera, con información de carácter geográfico. A pesar de que cada SIG se utiliza en un área en particular, hay muchas características comunes a todos ellos. Además, se ha llevado a cabo una fuerte estandarización de forma que la mayor parte de componentes software SIG son interoperables. En consecuencia, la aplicación de la ingenería de LPS en este dominio es un problema factible e interesante.

Aplicar ingenería de LPS a un nuevo dominio es un proceso complejo y, para garantizar la validez del diseño final de la LPS y su evolución, es importante seguir de manera estricta una metodología adecuada al dominio concreto. Considerando que no existe una metodología adecuada para el contexto de nuestro trabajo (es decir, aplicaciones SIG basadas en web desarrolladas en una compañía de desarrollo de software con varios productos en el mercado), hemos decidido combinar varias metodologías existentes y extender su alcance con determinadas tareas que servirán para sacar el máximo provecho a nuestro contexto.

Tras la definición de nuestra LPS siguiendo esta metodología, hemos encontrado que las técnicas tradicionales para implementar LPS no son adecuadas para nuestro dominio, debido a las particularidades y requerimientos en el desarrollo de aplicaciones GIS basadas en la web. Por lo tanto hemos definido e implementado un nuevo motor de derivación para la generación automática de software que mantiene las formalidades detrás de las LPS pero, al mismo tiempo, proporciona un nuevo grado de flexibilidad gracias al uso de una conocida técnica industrial: scaffolding.

Resumo

A enxeñería de liñas de produto software (LPS) é un campo de investigación que pretende industrializar o desenvolvemento de software usando técnicas como a producción e customización en masa, ou a reutilización de componentes software. Un sistema de información xeográfica (SIX) é un sistema de información que traballa, de algún modo, con información de carácter xeográfico. Aínda que cada SIX utilízase nun área en particular, existen moitas características comúns a todos eles. Ademáis, levouse a cabo unha forte estandarización de xeito que a maior parte dos componentes software SIX son interoperables. Polo tanto, a aplicación da inxeñería de LPS neste dominio é un problema factible e interesante.

Aplicar enxeñería de LPS a un novo dominio é un proceso complexo e, para garantizar a validez do deseño final da LPS e a súa evolución, é importante seguir de maneira estricta unha metodoloxía adecuada ao dominio concreto. Tendo en conta que non existe ningunha metodoloxía adecuada para o contexto do noso traballo (é dicir, aplicacións SIX baseadas na web desenvoltas nunha compañía de desarrollo de software con varios productos no mercado), decidimos combinar varias metodoloxías existentes e extender o seu alcance con determinadas tarefas que servirán para sacar o máximo aproveitamento ao noso contexto.

Tras a definición da nosa LPS seguindo esta metodoloxía, encontramos que as técnicas tradicionais para implementar LPS non son axeitadas para o noso dominio, debido ás particularidades e requerimentos no desenvolvemento de aplicacións SIX basadas na web. Polo tanto deseñamos e implementamos un novo motor de derivación para a xeración automática de software que mantén as formalidades das LPS pero, ó mesmo tempo, proporciona un novo grado de flexibilidade grazas ó uso dunha coñecida técnica industrial: scaffolding.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivation

The traditional approach to software development consists of a series of steps that must be repeated for each new product. Requirements analysis, solution design, implementation, testing and maintenance are performed over and over even when similar products are developed. It is still common that software is developed in an artisan way following this process. The problem with this approach is that both software development itself and product maintenance are slow and highly costly processes in order to produce high quality software. For this reason, many efforts have been made to industrialize software development. Software product line engineering and model driven development are two of the main research fields in this direction.

Software product line engineering (SPLE) uses strategies such as mass production, mass customization or reusable software artefacts to automate the development of software product families. That is, this discipline is used to create similar software systems that are different only in certain characteristics [ABKS13]. A software product line is defined as "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" [CN02]. In short, a software product line allows us to generate a family of similar software products built by assembling and automatically integrating a set of common reusable components. The set of features provided by a SPL is usually organized in what we call a *feature model*, and each of the products to be generated is defined as the set of features that it provides.

Model driven development (MDD) is a paradigm for applying the modelling advantages to software engineering activities [BCW12]. The two main concepts in MDD are models, which are simplified representations of reality centered on a particular domain, and transformations, which are manipulative operations on these models that allow them to be transformed into new, more refined models, or even source code of the final system. The actual implementation of MDD in the industry is very low [BCW12]. Nevertheless, there is a technique used in industry that is an informal application of some principles of MDD: *scaffolding*. This technique was popularized in 2005 by Ruby on Rails[1], and it allows accelerating the development of software by generating source code. It is usually used by programmers in the early stages of software development, because scaffolding tools allow to generate repetitive and easily abstracted code from a specification. Other current frameworks using this technique are Grails[2], Spring Roo[3] or Yeoman[4].

The objective of SPL and MDD is not only to improve the efficiency in the production of software, but also the quality of the software systems produced. Both seek to change the paradigm of software development from artisanal to industrial production. The difference is that SPL focuses on building product families that share identical and reusable components, which differ only in some features, while MDD generates platform-specific code from a more abstract and flexible model. As a consequence, the range of different products that can be created with MDD is broader than those that can be created by SPL, but products generated by an SPL are easier to specify and are normally ready for production when they are generated [PPP09, CAK+05]. Hence, both approaches seek to automate and industrialize the development of complex software systems, but the tools and features provided by each approach are very different in nature. Previous investigations have resolved that some application domains would benefit from a combination of SPL and MDD [TBD07, CFP08]. Geographic information systems (GIS) are one such domain.

A GIS is an information system with geospatial features and capabilities [WD04]. Within a geographic information system, geographic data such as the shape of a

[1] http://rubyonrails.org/
[2] https://grails.org/
[3] http://projects.spring.io/spring-roo/
[4] http://yeoman.io/

river or the area of a building can be handled and represented in a map viewer. GIS are widely used in various general-purpose applications (e.g. web search engines, social networks, etc.) and in various fields of research and production (e.g. engineering, resource management, biology, ecology, logistics, etc.). The impulse that has occurred in communication technologies and the improved Internet access allows the current use of GIS in many mobile devices to visualize and manage geographic data stored on computers over the Internet. Furthermore, the advance in the geolocation capabilities allows any common device such as mobile phones can get our position. This has benefited the appearance of new GIS features in existing applications and the improvement of working workflows. Examples of this are all the positioning functionalities of applications such as Facebook[5] or Twitter[6]. Another change in the GIS software provoked by these advances is the increase of the functionalities and use of the geographic information systems based on web. Traditionally, these web-based GISs have provided a small set of features but, with today's technology, they can be as powerful as desktop GIS applications retaining all the benefits of being a web application.

Geographic information systems have always shared an enormous amount of functionality and features between them, regardless of the GIS application context. Certain requirements, such as storing and indexing georeferenced data, performing location-based queries, displaying information as a set of layers, or grouping layers on different maps are shared by a good number of existing GISs. However, the early components and GIS software used to be implemented following different and incompatible conceptual, logical and physical models. For example, depending on the software used the *polygon* data type had a different definition, or the *overlaps* predicate had a different semantic meaning in each case. Because of this, it was very complex to develop interoperable applications, software or components because one could not even migrate data from one application to another without implementing an *ad-hoc* process. To solve this problem, a joint and collaborative effort was undertaken by two organizations, the International Organization for Standardization[7] (ISO) by means of the the ISO/TC 211 [fSn] that is defining the ISO 19100 set of standards, and the Open Geospatial Consortium[8] (OGC). Most GIS software now complies with these standards, interoperability between the different components is simple, and equivalent components can be replaced without problem.

Since the functionalities of GIS are very similar between the different applica-

[5] https://facebook.com/
[6] https://twitter.com/
[7] https://www.iso.org/home.html
[8] http://www.opengeospatial.org/

tions, they usually share a common set of components, such as the map viewer or the geographic data import library. The biggest difference between the different applications is the specific domain to which the GIS is being focused, which directly affects the dataset that is handled by the application. That is to say, it is not the same to make a GIS application that improves the workflow of a transport company, where we can see the position of the trucks, than an application for the identification of agricultural parcels. But even those modules that depend on the data to be handled are very similar to each other and their code can be abstracted and generalized based on specifications such as the data model. Therefore, although each application may have a different purpose, they are all very similar from the point of view of architecture, components and technology.

As a result, we find that web-based GIS applications have many modules that can be produced by assembling components of the more classic SPL style (e.g., a map viewer library with its sub-features, a mapping importer, etc.), but there are other modules that need to be generated specifically for each product depending on some specification (e.g.., all modules related to the data model, such as georeferenced entities, properties, relations, layers, maps, or the structure of the menu, etc.). Current SPL implementation techniques, most of them shown in [ABKS13, MTS+14], are not suitable for generating dynamic code from product specifications. We find that the *scaffolding* technique, wich can be considered a subset of the MDD paradigm, can be applied to extend the classical functionalities of SPL and automate the development of those parts of the system. Therefore, our hypothesis is that we can combine SPL and MDD techniques to produce web-based GIS applications more efficiently and with greater quality.

Although the idea of combining SPL and MDD has already appeared in previous research papers [CAK+05, VG07], its practical combination, which is precisely our focus on this work, is far from easy. First, most existing tools for SPL or MDD have emerged from research projects and they are strongly academy-oriented instead of industry-oriented. Therefore, they have some difficulties supporting the wide variety of technologies applied in web applications, geographic information systems and in both types combined. Furthermore, the tools developed to support SPL and the tools developed to support MDD are not the same, and there is no platform capable of combining these two approaches. Therefore, the first objective of this thesis is to design and implement a derivation engine for a SPL that can assemble components from a set of features selected in a feature model, but that it is also able to follow a MDD approach to generate source code by transforming high-level models of the system. Furthermore, the derivation engine must be based on modern software development technologies in order to be used directly in the industry.

In order to validate the derivation engine and the SPL for web-based GIS

applications, it is necessary to apply them in a real industry environment. We can achieve this by joining efforts with Enxenio, which is a Spanish SME (small and medium-sized company) with experience in geographic information systems. To take full advantage of this collaboration, the knowledge of experts in GIS architectures, requirements and technology must be considered in the process that defines the SPL. We also have access to existing product code, so it must also be considered when defining the SPL. In addition, geographic information systems are a field with strong standardization and a large collection of service definitions and architectures that must be taken into account. Finally, if the software product line is to be used in the real industry, both the SPL and the products generated will evolve in time. As we have not found a methodology that covers all these points, the second objective of this thesis is to define a methodology to create SPLs that takes into account all these aspects.

Given that the software product line is designed to be used within a SME, we cannot assume that the analysts that will define and generate products are expert domain engineers with background in SPLE. Therefore, the process for the definition and derivation of products must be simple and it must not require deep knowledge on SPL or any specific technology, apart from the ones that are already present in the common industry practice. For this reason, the third objective of this thesis is to develop a tool for the definition and the derivation of the products in a way that its usage is non-intrusive with the current processes within the company. To comply with this, the tool must be developed using web technologies so the tool can be used from any device of the company without requiring the installation of any additional software.

To summarize, the main goal of this thesis is to prove that SPL and MDD techniques can be combined to produce web-based GIS applications more efficiently and with greater quality. In order to do that, we divide this goal into three specific sub-goals:

1. To design and implement a derivation engine that can assemble components using SPL techniques but that it is also able to generate source code using MDD approaches.

2. To define a methodology to create SPLs that is tailored to SMEs in terms of knowledge elicitation and the evolution of the SPL and its products.

3. To develop a tool for the specification and derivation of the products within the SPL in a way that its usage is non-intrusive with the current workflow of a software development company.

1.2 Contributions

Our first contribution is a methodology for the definition of software product lines based on other two methodologies and enhanced by ourselves taking into account the input of experts from industry. Due to the context of our work, we decided to develop a new methodology that can exploit the advantages in our context, such as having a supporting company (Enxenio) with GIS experts and existing products. At the same time, we also looked for existing methodologies that fit our problem, and we have found two different methodologies that combined work in synergy to complement each others: Magro et al. [MGP08] defines a process in six steps but without defining the process of construction of the architecture of the product line; Nakagawa et al. [NBM13] focuses precisely on the definition of the software product line architecture and therefore complements the above. In addition, we have taken into account traceability considerations proposed by Iida et al. [IMY+16], and we have extended all phases of the methodology to introduce our own steps.

Our second contribution is the design of a software product line for web-based geographic information systems. This design is the result of the application of our methodology. Two derived contributions are:

- An exhaustive list of features for generic web-based geographic information systems, taking into account different existing products.
- An architecture for web-based geographic information systems based on an architecture of reference, specified by standards, and enhanced with features from architectures of existing products.

Our last contribution is a tool for the definition and generation of web-based geographic information systems. This tool follows the specifications determined in the design of our SPL, and it includes several contributions to the state of the art technology for the implementation of software product lines:

- A derivation engine based on scaffolding providing a grade of flexibility not found in the state of the art alternatives.
- A previewing component that enables our tool to show the analyst a preview version of the products in run-time, thus allowing him or her to validate and refine the definition of the products before their real generation and deployment.

1.3 Thesis Outline

This thesis is composed by three parts. The first part is dedicated to the methodology and it consists of three chapters. Chapter 2 introduces the concepts behind software product lines and it describes the state of the art of the field, including the advantages of SPL and their unsolved problems. Chapter 3 shows the methodology we have defined for the application of software product line engineering in any domain.

The second part covers the application of the previously defined methodology within our context, web-based geographic information systems. In Chapter 4, we make a brief summary of the basic concepts for geographic information systems, and we describe some software related to GIS. In the rest of the chapters of this part, each step of our methodology is addressed: requirement analysis in Chapter 5, extracting requirements and features from existing products; architecture design in Chapter 6, where we study the reference architectures for GIS and we select one as our own; evaluation and derivation of the products in Chapter 7, showing the traceability between the features and the architecture and the details regarding the derivation of specific products.

The last part is about GISBuilder, a tool implementing the specifications extracted from our process. In Chapter 8, the state of the art in software product lines technologies and in industrial generation of software techniques is shown. In Chapter 9, we describe our tool and each part that compose it. Finally, in Chapter 10 we validate our tool and propose some use cases. The last chapter is conclusions and future work.

Part I

Software Product Line Engineering: methodology

2

Software Product Line Engineering: state of the art

2.1 Basic Concepts

Traditionally, the development of every software product goes through a series of steps: elicitation of requirements, design, implementation, testing and maintenance. When a software development company has to build a family of products, all the stages mentioned must be done for each one of them, even when the products share functionalities or are focused in the same specific market. The downside of this approach is that it requires high development and maintenance costs in order to produce high quality products, while the time to market for each product is long since development starts from scratch.

In other classic manufacturing industries, such as the automotive or the textile, the way the products are built went from a manual manufacturing process to industrial processes that use proper machinery [KS90]. This change allowed industries not only to produce their products massively, but also to confront the rising demand for individualised products [PBL05], i.e., mass-customization, and large-scale production of goods tailored to the individual customers' needs [Sta87].

Software product lines engineering (SPLE) [Bos00, PBL05, vdLSR+07, ABKS13]

is a discipline that aims at applying the same kind of evolution to the way software is built, i.e., applying mass-production, mass-customization and reuse strategies to software development. In [CN02], Clements provides the most well-known definition of a software product line (SPL):

> 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.

This is, a software product line is a family of software products sharing features developed from a common set of reusable core assets that can be combined and configured in different ways for different products. A SPL separates the development of these core, reusable assets (i.e., the platform), and the development of the actual applications (i.e., the products). This definition includes five fundamental concepts:

- "a set of software-intensive systems...": a SPL does not pretend the development of a single product, but of many similar products (i.e., a family of products). Therefore, a domain engineer has to decide the scope of the products to build within a SPL.
- "...common, managed set of features...": each product built within a SPL shares a set of features with every other product of the family.
- "...satisfy the specific needs of a particular market segment or mission...": a SPL makes only sense to produce specialized products, this is, a set of products solving similar problems.
- "...a common set of core assets...": the products within a SPL are built using the same components. This is, the code of the different generated applications is shared and not unique for each application. In order to generate products that support variable requirements, a domain engineer has to specify variability within the set of components of the SPL. Therefore, some of these core assets are optional or variants.
- "...in a prescribed way.": the products built from a SPL are assembled or generated in a predefined way, i.e., following a pre-established architecture.

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 variability of the features defined for the family of products. A *feature* is an end-user visible aspect or characteristic of a software system [KCH+90]. Features are

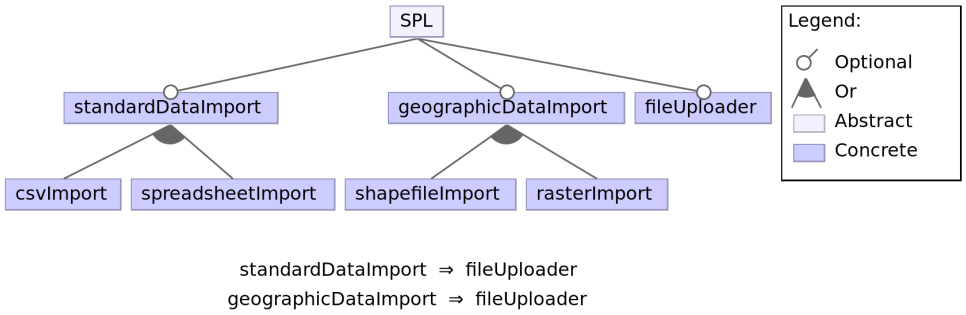


Figure 2.1: Example of a simple feature model

used in product line engineering to specify and communicate commonalities and differences of the products between stakeholders, and to guide structure, reuse, and variation across all phases of the software life cycle [ABKS13]. The platform of a SPL is modelled as a set of features, and variability management 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. Hence, variability management is one of the main tasks in SPL development. There are many modelling techniques to identify and define the platform features, such as FODA (Feature Oriented Domain analysis) [KCH+90], FORM (Feature Oriented Reuse Method) [KKL+98], FM (Feature Modeling) [CGR+12], DM (Decision Model) [SRG11], or OVM (Orthogonal Variability Model) [PBL05].

Features are usually classified within a **feature model**, which represents the features of a family of products, the relationship among them and whether if a feature is common (mandatory), alternative or optional [KCH+90, CHE05]. A simple feature model can be seen in Figure 2.1.

A *product* is specified by a feature selection, i.e., a subset of the features of the product line. However, not every subset of features is valid since there are relationships among the features and *cross-tree constraints* [Bat05] that need to be fulfilled. In Figure 2.1 we can see that the feature "standardDataImport" has two *sub-features*, "csvImport" and "spreadsheetImport" in an *OR* relationship. This means that if a product includes the feature "standardDataImport", then it must also include at least one of its sub-features in order to be a valid product of the SPL. In Figure 2.1 we can also see the cross-tree constraint "standardDataImport *implies* fileUploader". Therefore, beyond includying at least one of its sub-features, the product must also include the feature "fileUploader" to be a valid product. Precisely, there is an operation called *valid product* that checks wheter a subset of features of

a feature model complies with the restrictions and relationships specified on it and therefore a product can be generated from this selection of features. Besides this operation, there are many more useful ones associated to feature models [BSRC10], such as *number of products* or *valid feature model*.

2.2 Advantages of software product lines

The main target of SPLE is to be able to build customized products at a fraction of the cost, solving customers needs with full products specifically crafted from their requirements.

The evolution of costs between conventional development and SPL is very different. Following a traditional approach, this cost grows linearly as the number of products to be developed increases. On the contrary, the cost of developing a SPL is very high initially, but it is being amortized with each new product of the family that is built. According to [PBL05], the *break-even point for the cost of developing a SPL is when the number of products of the family developed is three.* From that number, *the SPL offers very remarkable development cost savings* especially if we also consider the maintenance of the products, as we can see in Figure 2.2.

In a conventional environment, a very common approach is "copy & own". A developer starts by copying a previous application similar to the one he or she needs to built, and from there the copy evolves independently. This is applied not only for full applications but also for every core asset. The similarity between the two applications serves to speed development, but in no case for their maintenance. In case a bug is discovered, the development team in charge of the maintenance must debug the code in both applications. In case new features are required, they will have to develop them in both applications. This is, the similarity between the applications is not exploited to reduce the maintenance costs. Reuse is opportunistic, and the management of similarity (common parts and variable parts) is secondary. Therefore, the traditional environment tends to focus on the product: each conventional product is maintained and human teams also tend to fragment in this way. The company does not take advantage of the potential synergies that could be derived from the similarity between products, and the number of different products that can be managed effectively is very limited.

On the contrary, a SPL environment is specifically designed to manage both the commonalities and the variabilities among the products. *Reuse is not opportunistic, but planned, implemented, and the incorporation of new variants is done in a systematic and controlled manner.* This streamlines not only the product development and its time-to-market, but also maintenance. Maintenance efforts

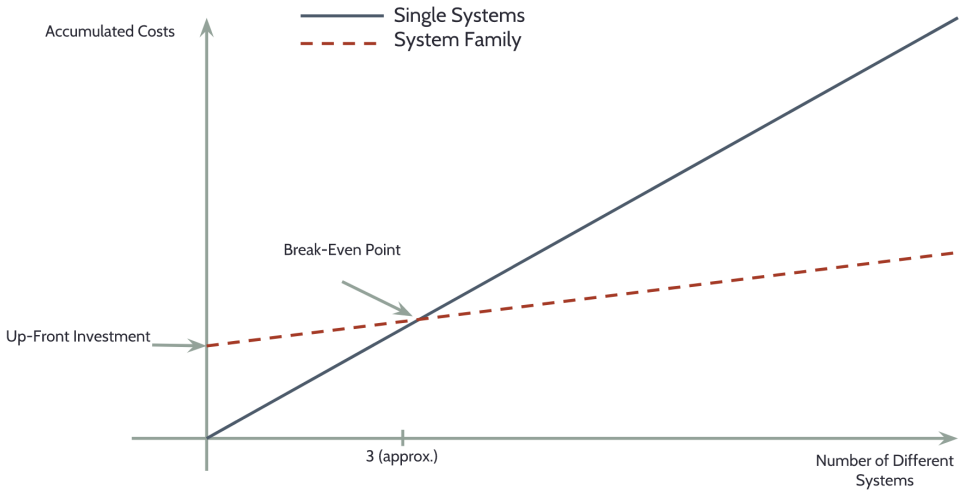


Figure 2.2: Break-even point for a SPL

are capitalized by all products. An error detected in a product can be relatively easy to correct in all products of the line thanks precisely to the existence of the common framework offered by the product line. As a straight consequence from this point, the *quality of the products is enhanced*. The products built within a SPL are much less prone to have bugs because their code has already been tested for the rest of the products already built.

The last main advantage of using a SPL is the *reduction of time to market* [PBL05], often a very critical success factor. Although the time to market for the first products of a SPL is higher, because the common artefacts have to be built first, as soon as the SPL is developed and running the time to market of new products is shortened since most of the code of the new products is already written. As opposite, in traditional development, this time is always linear to the number of products (see Figure 2.3). Even in the case that a customer requires a feature not provided by a SPL, a similar product an be generated with all the rest of the features and then this new feature can be added, shortening anyway the time to market of this new application.

2.3 Unsolved problems

There are many well-known success stories in the use of software product line [WCK06]. However, despite the years that have passed since they were initially

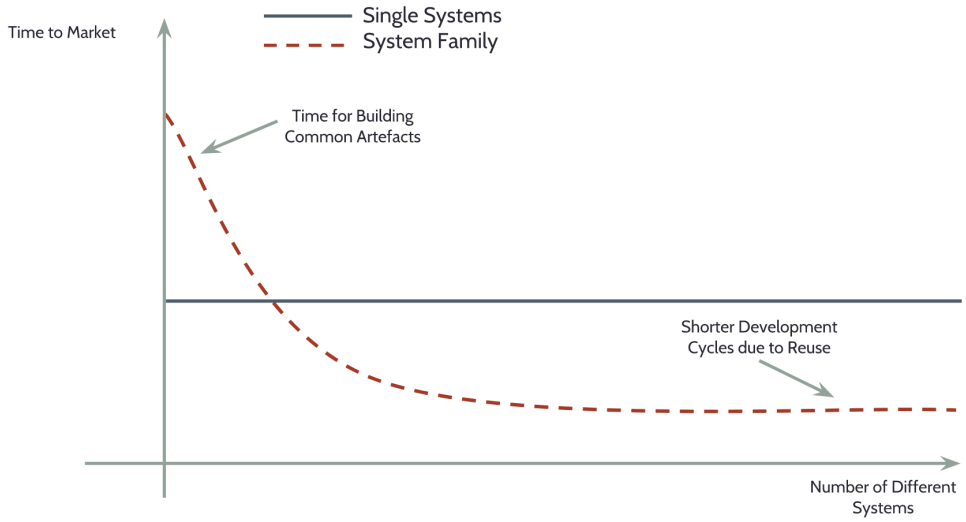


Figure 2.3: Time to market for a SPL

proposed, SPL still pose many challenges from the research point of view [MP14]. Some of them as discussed below.

- **Optimizing the scope of a SPL**. Instead of developing very specialized product families, it would be interesting to optimize the scope of a product taking into account not only market aspects but technical aspects such as the life-cycle management of features, the core assets which implement them and the product architecture. Finally, the economic viability of the whole product line has to be considered as well.

- **Describing the interrelation between scope, requirements, features and other development activities**. This is, traceability has to be explicitly stated between all the stages of the development of a product within a SPL. This way, the impact of requirement changes and the evolution of both the platform of the SPL and the product can be properly handled. Maintaining a clear traceability also facilitates changing a specific core asset for a new version, for example.

- **Eliciting and handling application-specific deviations**. State of the art SPL usually do not cope with customer-specific requirements. However, even when we are dealing with products belonging to a family, there are domains in which some application-specific requirements, features or components needs

have to be managed by th SPL in all the stages of the development process, from the design of the products to their maintenance, going through the derivation of the application-specific code. We deal with one of such domains in this work, web-based geographic information systems (see Chapter 5).

In this work we address each one of this issues. The methodology we use to approach the definition of our product line, described in Chapter 3, specifically provides tasks for solving the first and second issue: most of the decisions regarding the product line are solved by taking into account technical aspects of existing products, literature and technical knowledge from experts; at the same time, traceability is maintained all over our process. Furthermore, handling application-specific deviations is totally achieved in our product line from the moment we deal with specific data model specifications for each product.

To conclude, the application of software product lines in the development companies of software has been limited to very specific contexts such as the development of *embedded systems* [BRN+13,WCK06,Van02]. In other more complex areas, such as the development of web applications or other fields with cutting-edge and changing technologies, the application of SPLE is becoming more frequent. But the proposals in the literature usually focus in the domain variability model at a higher abstraction level rather than in the management of variability at the implementation level [MSC+14]. Therefore, there is a need for new SPL methodologies, technologies, and tools for this type of cases [UBFC14].

3

Industrial expertise in the definition of a Software Product Line: a new methodology

3.1 Introduction and motivation

One of the objectives of this work is to use the resulting tool within a real software development company, Enxenio. In order to achieve this goal, we have had the collaboration of GIS experts from the company all over our process, taking us to a privileged and unusual context for carrying out this work. Enxenio is a Spanish small and medium enterprise with expertise in geographic information systems. In fact, this company is a leading provider of web-based geographic information systems at the Galicia region, with many previous projects for the public administration and private clients. Enxenio has collaborated with the Database Laboratory at the University of A Coruña for a long time, and several works, such as [LPFCP09, BCLF+07, PBF+07], are some results from this relationship. For our present work, Enxenio has been helping with their expertise and their existing web-based GIS applications. This collaboration is not altruistic since Enxenio would benefit greatly from the outcome of this thesis, since the application of SPLE for the development of future GIS would give the company a strategic advantage.

In order to make the best from this collaboration, we have looked for the

most adequate methodology for the definition of our SPL. We have found that the methodology to follow must comply with the next requirements:

- The methodology has to make the role of experts with knowledge about architectures, requirements and technology explicit.
- Considering that direct access to the souce code of the products may be provided, the methodology must take the source code into account.
- The software product line domain (in our case, geographic information systems) may have undergone a strong standardization effort that the methodology must not ignore.
- Finally, if the software product line is to be used in the real industry, the platform and the generated products will evolve and therefore, the methodology must facilitate handling this evolution.

We have found two different methodologies that comply with some of this requirements: Magro et al. [MGP09] and Nakagawa et al. [NBM13].

Magro et al. [MGP09] defines six steps based on the works of Bosch [Bos00], DSouza & Wills [DW99], and Mili et al. [MMYA02], and shows an example of its application for the construction of a SPL for validation systems. The six steps defined in this work are:

1. Definition and analysis of the domain, not only to use this information in furthers steps of the process but also to check if the application of SPLE in the selected domain is feasible and advantageous.
2. Product planning. All kind of requirements associated to the different products of the domain have to be accounted for.
3. Design of the architecture of reference by identifying the architectural components, their design and their interactions.
4. Development of the reference architecture components. This fourth step consisted in specifying and analysing the components of the architecture.
5. Architecture evaluation in order to detect problems and drawbacks.
6. Derivation of a specific product.

Nakagawa et al. in [NBM13] describe *ProSA-RA2PLA*, a methodology focused on the *product line architecture* (PLA) design. They define an iterative process of five steps (see Figure 3.1):

1. Reference architecture identification/selection, in which the reference architecture is selected taking into account the needs and the scope of the product line.
2. Elements selection/prioritization. They sort the elements from the reference architecture in order of importance, and even not all of them need to be considered for the product line.
3. PLA structure building, the step in which the product line architecture is designed.
4. Variability model building. The PLA variability is identify and designed in this step.
5. PLA evaluation to check if the elements from the reference architecture that were considered important for the product line were indeed added to the final PLA.

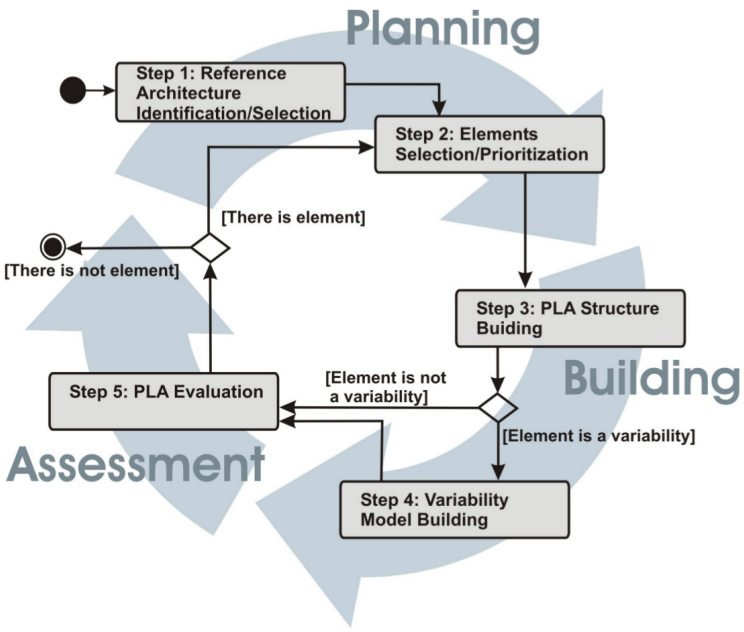


Figure 3.1: Structure of ProSA-RA2PLA by [NBM13]

We adapted the methodology of Magro et al. by decomposing the steps related with the PLA design (three and four) into those suggested by Nakagawa et al. In

addition, we also considered the work of Diaz et al. [DPG14], since the traceability among the requirements and the PLA is a key issue for guaranteeing the SPL maintenance [AK04]. This issue was also considered by Iida et al. [IMY+16] for the construction of an automotive braking system SPL.

3.2 Definition of a new methodology

We have defined a new methodology based on the two previous ones, mentioned above, that we consider complementary one to each other, and some extensions related to other issues that we consider critical to take advantage of our situation. We can see our methodology in Figure 3.2. The methodology is framed into the two levels of SPLE: domain engineering and application engineering [PBL05]. The application engineering stage is in charge of the derivation of products, whereas the domain engineering stage is divided into three phases (see Figure 3.2). Among these two levels we define a process which consists on four activities or stages of high level that are executed iteratively. The first stage is the *requirements analysis*, resulting on the definition of the feature model of the SPL. The second stage is the *architecture design* that leads to the design of the product line architecture. Next, the third stage is the *architecture evaluation*, whose result is the identification of issues in the feature model or in the product line architecture. This may lead to a new iteration of the process, going again to the first stage. The last stage is the *derivation of a specific product* in the the application engineering level. Each one of this stages is divided into steps or tasks which are described next. The steps that are based on another methodology are coloured in blue in the figure. The steps proposed by us are coloured in white.

3.2.1 Requirement Analysis

The first stage of our process is the requirements analysis, composed in turn by in two steps: domain analysis and product planning.

The domain analysis step (see the step 1.1, Figure 3.2) extends the work of Magro et al. [MGP09], which was only focused on analysing the domain to determine the feasibility of constructing a SPL and extracting the requirements of the products to identify the commonalities and variabilities of the domain. This takes place on the first task, *requirements* (see the step 1.1.1, Figure 3.2). In our case, we add a second task, *related work analysis*, in which we also consider the related work to determine requirements of the domain that may have not appeared in the previous analysis (see the step 1.1.2, Figure 3.2). For example, if the experts in a company extract the requirements for the family of products, then this step would help by considering

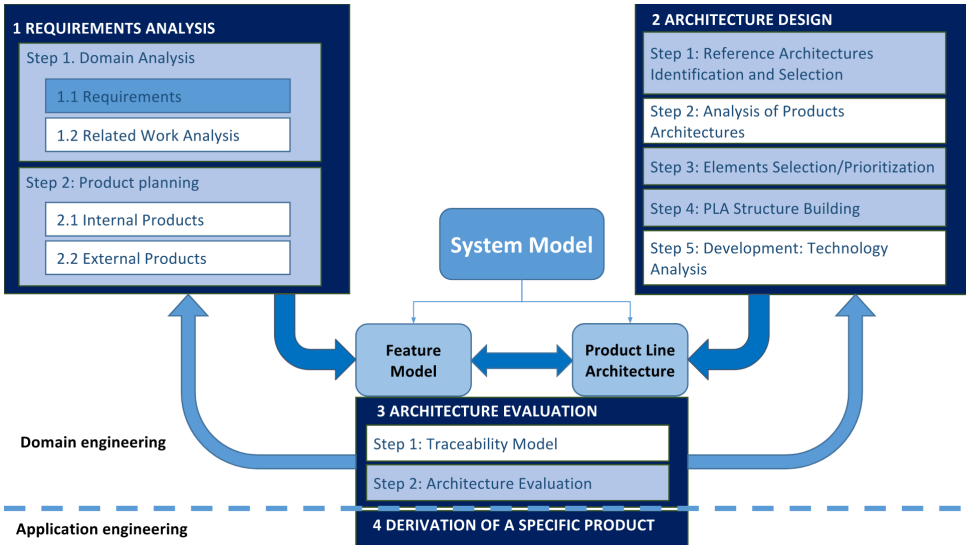


Figure 3.2: Methodology

requirements that have not appeared previously in product of the company but still are interesting. This is a good point specially to help designing an SPL even more prepared for its evolution. In addition, analysing the related work may reveal other SPLs of the domain that could help and not starting from scratch.

The product planning step considers all kind of requirements associated to the different products that can result from the SPL deployment (see the step 1.2, Figure 3.2). But it considers not only those that belong to the company (see the step 1.2.1, Figure 3.2), but also it is important to consider other projects that are known or relevant in the domain (see the step 1.2.2, Figure 3.2). Once the phase is complete, from these complete analyses of requirements, a feature model should be constructed.

3.2.2 Architecture Design

The second phase consists in the architecture design, which is divided into five steps (see Figure 3.2) heavily based on the methodology by Nakagawa et al. [NBM13]. The first step, *reference architectures identification and selection*, tries to identify existing reference architectures or standards in the field in order to not start from scratch (see the step 2.1, Figure 3.2). This is very adequate in our domain due to the strong standardization we have mentioned already (see Chapter 4 and

Section 6.2). Next step, *analysis of products architectures*, is added in top of the existing methodology to contemplate the architectures used by previously developed products in the company (see the step 2.2, Figure 3.2). Analysing and taking into account these architectures is interesting since it may enrich the architecture of the SPL. The steps 3 and 4 are proposed by Nakagawa et al. In the *elements selection/prioritization* the elements of the architecture are identified and selected (see the step 2.3, Figure 3.2), whereas in the *PLA structure building* the architecture is designed (see the step 2.4, Figure 3.2). Finally, we added a new step, *development: technology analysis*, to analyse the technology requirements in order to determine needs and interoperability problems, once again benefiting from our relationship with Enxenio (see the step 2.5, Figure 3.2). This step is also important since it helps to plan the technology evolution. As a result of this second phase, a PLA is obtained.

3.2.3 Evaluation

The third stage consists in mapping the features and architectural elements, in order to guarantee that all features of the SPL are supported by the PLA (see the stage 3, Figure 3.2). This is important not only to observe and guarantee the coherence between the feature set and the PLA, but also to prepare the SPL for its evolution. In these steps, it is important to check that there are no inconsistencies or drawbacks between previous stages and the results obtained. If something is missing, this should be solved in previous stages and the process starts again from stages 1 and 2, checking again all the steps (see the feedback arrows, Figure 3.2).

3.2.4 Derivation of a product

The last stage is the actual derivation and deployment of the products. This activity takes no part in the domain engineering processes but it is more a application engineering one. Our methodology only focus on the requirements, so this task is the analysis of the requirements for the derivation process so afterwards we can implement the tool or platform for the SPL. However, after finishing our process we have already developed a specific tool that follows the guidelines extracted from the process. We describe our tool in Part III.

Part II

Definition of a Software Product Line for web-based Geographic Information Systems

4

Geographic Information Systems: state of the art

4.1 Introduction

In Part I we have described a new methodology that benefits of the particular context of this work whereas it is still based on proven methodologies from the literature. Choosing a methodology is the obvious first step in order to define our software product line with the formalism that is required if we want to guarantee the quality of the design and the right and managed evolution of the product line.

Now that the methodology to apply is explained, in Part II we can proceed with the design of a software product line for web-based geographic information systems. Some steps of the methodology require to study the domain for the SPL, web-based geographic information systems, from a more technical point of view. For example, we need to define the list of requirements of this domain and also we need to study the existing reference architectures. Therefore, before starting this process, in this chapter we describe GIS from a more functional point of view, giving the reader an idea of what a GIS is and providing some examples of GIS applications and features. We also enumerate a list of software used by GIS to provide part of the features.

4.2 Basic Concepts

Geographic information, displayed as paper maps, has been one of the driving forces behind the progress of our society for many centuries. Until a few decades ago the representation, manipulation and synthesis of geographic information was limited to the use of paper maps, and these tasks were limited to manual, non-interactive processes. The exponential improvement in the performance of IT-based technologies and the increasing demand for manipulation and interactive analysis of geographic information have triggered the need for geographic information systems (GIS).

GIS are used in many fields, each one of them with its specific point of view. Due to that, there are many definitions of what is a GIS:

- A geographic information system is an application to assist in making decisions related to geography [HA03, LGM15].
- A geographic information system is a set of computer tools that allow analyzing and performing geographic type simulations [LT92, BMML15, RSV01].
- A geographic information system is a set of data structures and algorithms to represent, query, manipulate and visualize geographic information [WD04, RSV01].

We can make a more complete definition based on all the mentioned: a **geographic information system** is a set of software tools to model, represent, store, manipulate, query, analyze and display information that includes a geographic component. We consider a geographic information system as a set of computer tools without having into account the organizational and business aspects of its use, which are mentioned, for example, in [HA03]. In addition, the system must be able to model, represent and store geographic information by providing tools for conceptual modeling of applications, representation of information in data structures, and efficient storage [MPV05, SC03]. Finally, the system must allow the analysis of information by providing tools for the manipulation of the stored information, its querying and its visualization [YG05].

GIS is a field that has been receiving a lot of attention lately. We can note its presence in applications used everyday by millions of people, like Google Maps[1], or social networks with location-based features like Facebook or Twitter. Google Maps and Google Earth[2] represent an inflection point regarding web-based GIS.

[1] https://maps.google.com
[2] https://www.google.com/earth/

Before these two applications appeared, web map viewers were primitive and very focused on specific fields. Since they appeared, they have laid the foundations of how a web-based GIS should be. Besides these well-known examples, there are many disciplines using GIS to improve and facilitate its work, such as cartography, biology, ecology, transportation and warehouse logistics. GIS has also reached the mobile systems thanks to the huge evolution in communication technologies and the increased penetration of Internet access.

Even though GIS are used in many disciplines and with a variety of purposes, there are many features shared between almost every one of them. Examples of these common features are storing and indexing geo-referenced data, displaying information as a set of layers, or displaying information in map viewers with zooming and panning capabilities. At first, even with similar features, the software artefacts used to implement each GIS followed different and incompatible conceptual, logical and physical data models. For example, even a simple concept like the data type polygon had inconsistent definitions between GIS technologies. In the last years, the Open Geospatial Consortium (OGC) and the International Organization for Standardization (ISO) have defined a set of standards. With these standards being followed by most software artefacts, these artefacts become interoperable and because of this, a current GIS application can switch its components easily. The standardization processes also affected the architectures and services provided by the geographic information systems, establishing a series of standard services that include the most common ones. We detail this part of the standards in Section 6.2 as part of our methodology.

4.3 Geographic Information Systems features

Current geographic information systems provide lots of features. The most important is the visualization of maps, with the mentioned Google Maps and Google Earth as the two most representative examples. Furthermore, there are GIS features integrated in general purpose applications that do not require any map viewing. Anyone with a mobile device could be using this features without even noticing. For example, when a person publishes a tweet in Twitter, he or she has the option to add the geographical location of the device to the tweet so all his followers can know where was the message originated.

Beyond the map visualization, geographic information systems have been traditionally used as tools to manage and inventory resources. An example is the geographical information system EIEL (*Encuesta sobre Infraesctructura y Equipamientos Locales*), of the Provincial Council of A Coruña (Figure 4.1),

Figure 4.1: EIEL user interface

developed by the Databases Laboratory of the University of A Coruña since 2000 [BCLF+07]. This geographic information system consists of a cartographic inventory of the infrastructures and the equipment of the province that allows to manage of the territory, to verify the correct provision of services and to coordinate the services of different municipalities in common actions. This geographic information system has two different tools: a web-based map viewer that allows the user to visualize the information entered from the management tool (WebEIEL[3]), and a desktop tool available for the technical staff of the Provincial Council and of the municipalities of the province (gisEIEL[4]).

Another area in which geographic information systems are currently being used daily by lots of people are navigation devices. Such devices have emerged thanks to the development of the global positioning system (GPS), initially only for military use, in the 1960s, but extended to its civil use since the 1990s. This system allows the localization of an object through a series of satellites. A GPS receiver is incorporated into a mobile device which is capable of accurately determining its location. In addition, the road network is represented by a graph in which the nodes correspond to intersections of the roads and are labelled with the permitted turns, and the edges correspond to segments of the roads and are labelled with the allowed direction, the length, and the type. With this information the system can compute the optimum route between two points taking into account factors such as distance or time required. In addition, the system can show the user a map of the area with the location, direction and speed of the vehicle. This type of devices include also functionalities for calculation and communication of routes, which can

[3] http://webeiel.dicoruna.es/
[4] http://webeiel.dicoruna.es/giseiel

Figure 4.2: TomTom Go 1600[5] user interface

include in these calculations variables in real time such as current traffic, road events such as accidents or atmospheric difficulties.

Initially, GPS devices were designed and marketed independently only to provide this feature, and they are the only devices that can manage to do that. Some brands were very popular, such as TomTom[6] (see Figure 4.2). However, with the emergence of the smartphones which include GPS receivers themselves, nowadays most of these features can be used by them (see Figure 4.3), with different and numerous applications providing this kind of features and even more than the previous GPS devices.

The current high availability of devices with GPS receivers and mobile communication networks allows the use of geographic application systems to build fleet control tools. In this case, besides using a vehicle navigation device, this device communicates to a central server its location using mobile technology (GSM, GPRS, UMTS, or HSDPA). The server is responsible for storing the location of the vehicles and it can use this information to perform the tasks that are needed, for example, locating stolen vehicles, picking and delivery planning in parcel companies,

[5] https://www.tomtom.com/es_es/drive/car/products/go-6100-europe
[6] https://www.tomtom.com/es_es/
[7] https://www.waze.com/es/

Figure 4.3: Waze[7] application user interface

calculating the nearest vehicles for a taxi stand or managing traffic. Currently, there are different commercial systems following this philosophy, both for vehicles, ships and aircraft (Figure 4.4).

A more specific scope for geographic information systems more focused on simulation instead of management is the analysis of flooded areas. In this case, a digital terrain model is used to represent the height of each point (usually visualized on a 2D map using color), with a model of the hydrographic network representing the river channel and the river flow rate that can withstand naturally. From this information, together with data about precipitations or thawing in the area, the risk of flooding at each point of the terrain can be calculated [EIA+11] (Figure 4.5).

All the engineering disciplines that work with the terrain are frequent users of geographic information systems (road engineering, forest engineering, agronomic engineering). One example is the process of land consolidation (Figure 4.6), in which a new distribution of the properties in an area is carried out to increase the average size of the plots, reducing the smallholding and facilitating its exploitation. To do this, the geographic information system should include information about the

[11] https://www.simplytrak.com/
[11] https://www.flightradar24.com/
[11] https://www.quartix.net/
[11] https://www.marinetraffic.com/

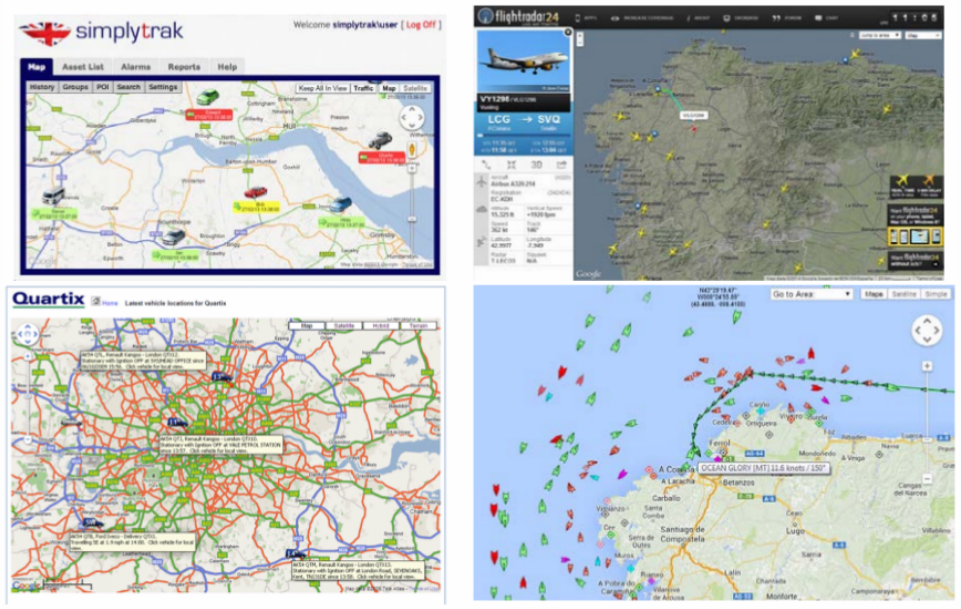


Figure 4.4: User interfaces for many different fleet management apps: Simplytrack[8], Flightradar24[9], Quartix[10]and MarineTraffic[11]

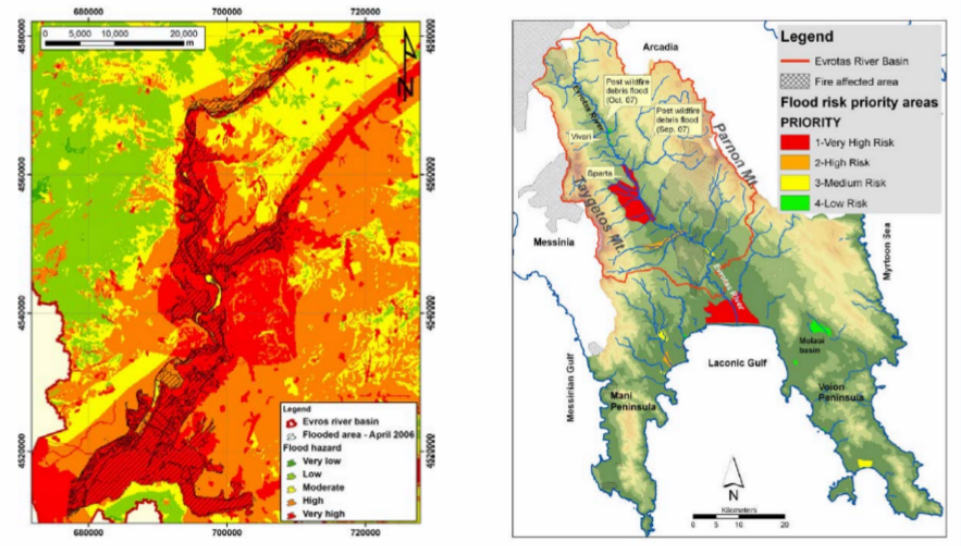


Figure 4.5: Result of a GIS for the calculation of flooded areas

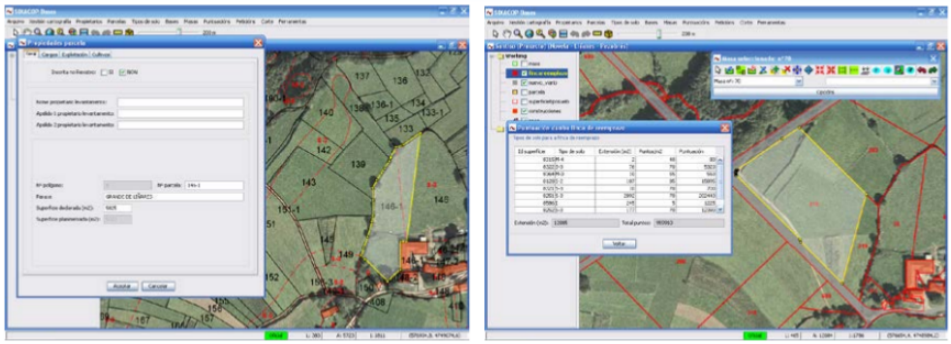


Figure 4.6: User interface of a GIS for managing the process of land consolidation

types of soil in the area, existing plots and their current owners, and should allow the definition of new replacement farms by automatically performing calculations and quality controls, allowing the technicians to work quickly and efficiently.

4.4 Geographic Information Systems software

To provide the reader with an idea of GIS tools, in this section we describe different types of resources and real tools that allow developers and users to implement and access GIS features.

There are currently many accessible projects whose purpose is the collection of information in the field of GIS or space DBMS. In this sense, these projects do not provide specific tools or resources, but act as repositories to find other existing projects.

The Open Geospatial Consortium[12] is dedicated to the definition of standards for geographic information. However, it is possible to find in this portal different proposals, articles and discussions on the different standards and technologies used. The organization also provides access to validation tools to check compliance with its standards, as well as a repository of tools conforming to those standards[13].

The Open Source Geospatial Foundation[14] works mainly as a project incubator of open source software projects related to spatial information.

[12] http://opengeospatial.org
[13] http://www.opengeospatial.org/resource/products/compliant
[14] http://www.osgeo.org/

4.4.1 Commercial GIS tools

In this section some commercial tools supporting GIS features are briefly introduced.

ArcGIS, by ESRI[15], is currently the market leader and has a very wide range of integrated functionalities as well as the possibility to integrate with a multitude of external tools.

ArcGIS has connectors for databases with SFS (Simple Feature Specification [fSi]) support such as Oracle Spatial or PostGIS. In addition, it also provides support for many other DBMS, such as Access, Oracle, DB2, SQL Server or Informix. Two data servers are provided, ArcSDE and ArcIMS, which allow the integration of geographic information from different environments by sending it to ArcGIS desktop systems or to other specific applications.

The platform has many functionalities for managing geographic information, both in the vector model and in the raster model. It also provides a free viewer, called ArcExplorer[16].

ArcGIS stores and exports information in Shapefile (SHP) format. This format is so widely used that nowadays it can be considered the *de facto* standard in geographic information systems. The technical description of this format has been published by ESRI [ESR], and it is used as an exchange format of geographic information thanks to many libraries that implement its functionalities. Shapefile allows the general inclusion of alphanumeric and geographic data. However, and despite being widely used, it has some limitations, such as not allowing geometries of different types in the same file.

Hexagon GeoMedia[17] can be considered the great rival of ESRI today. It has most of ArcGIS functionality and some similar tools. Like ArcGIS, Geomedia provides connectors for SFS databases such as Oracle Spatial and PostGIS. It also provides support for other databases, among which we can mention Access, SQL Server or DB2. The GeoMedia family of tools also provides a data server called **Geomedia WebMap Professional**[18], a web server-based solution that enables the availability of information through the web and mobile devices with ease. In terms of functionalities, GeoMedia provides many possibilities for managing vector and raster information, and a free viewer (GeoMedia Viewer[19]) for displaying the geographic information generated.

[15] http://www.esri.com/
[16] http://www.esri.com/software/arcgis/explorer
[17] http://www.hexagongeospatial.com/
[18] http://www.hexagongeospatial.com/products/power-portfolio/geomedia-webmap
[19] http://www.hexagongeospatial.com/products/power-portfolio/geomedia-add-ons/geomedia-viewer

MapInfo[20] provides support for the most common databases and spatial information formats, conforming to standards such as WMS [fSj] and WFS [fSl], and advanced functionalities for vector and raster information analysis, publication of thematic maps, etc.

AutoCAD Map 3D[21], which is part of the family of tools AutoCAD by Autodesk, provides more specific functionality that allows access to both spatial information and CAD information within the tool. **Autodesk Infrastructure Map Server**[22] is also a map server that allows the publishing GIS and CAD information.

Bentley Map[23] is the evolution of the Microstation Geographics tool. It allows the management of vector and raster information, it supports numerous GIS information formats and specific functionalities such as visualization and 3D processing.

All these tools have the usual advantages and problems in commercial products. On one side, the products are available immediately, have a large set of features and the support provided by a large company. In addition to the main applications, the product packages for these tools usually include different utilities to extend the functionalities and facilitate the integration with other systems of the same family. However, the features provided by these products, integrated into large specific packages, may not fit the specific needs of many companies since the product is normally sold in closed packages. On the other hand, since these products are not open source, the software has to be used as a black box, which prevents any access, inspection or modification beyond the extension points defined by the tool itself.

4.4.2 Spatial DBMS

This section describes the main database management systems with support for spatial information.

Oracle Spatial[24], currently known as Oracle Spatial and Graph, is a plugin that adds spatial functionalities to Oracle 11g DBMS.

In the last versions, Oracle has included in their DBMS the component Oracle Locator, providing the most basic tools to allow working with spatial information. Available functionalities include specific data types and operators, following OGC

[20] http://www.pitneybowes.com/us/location-intelligence/geographic-information-systems/mapinfo-pro.html

[21] https://www.autodesk.es/products/autocad-map-3d/overview

[22] https://www.autodesk.com/products/infrastructure-map-server/overview

[23] https://www.bentley.com/en/products/product-line/asset-performance/bentley-map

[24] http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/index.html

and SQL/MM standards [fSm]. Coordinate systems are also included, and the DBMS is improved with spatial indexes.

Oracle Spatial is an extra option that adds additional functionalities to the previous ones. Some of these new features are: support for raster information, models to represent topologies and networks, and support for 3D information. It also includes a route calculation engine. Finally, Oracle Spatial provides web services for publishing information conforming to OGC standards. The MapViewer tool allows the visualization of this information in a web application, and it is compatible with other applications in the Oracle suite.

PostGIS[25], developed by Refractions Research, is an extension to the DBMS PostgreSQL[26]. This extension provides data types and operations for geographic objects based on the vector model and, since version 2.0, also based on the raster model. PostGIS is an open source tool published under the GPL license. PostGIS provides management functions for the creation and deletion of geometric tables and columns, as well as the management of spatial reference systems. It also provides access and editing operators for the geometry types. Supported operations include modification of coordinates, transformation of spatial reference system, realization of different related transformations (rotation, displacement, scaling) and simplification of geometries. It also provides numerous functionalities for transforming and converting formats, supporting standard formats and shapefiles.

Raster related features are much more recent in PostGIS, and they include elements similar to those already defined for vector data: new data types, import utilities, etc.

The DBMS **MySQL**[27] includes support for geographic information natively, without requiring any extensions, integrating geographic data types transparently. However, unlike PostGIS, MySQL does not support spatial reference systems. Spatial predicates are implemented using bounding boxes instead of the actual geometries, and it does not provide spatial operators and editing operations.

MySQL supports conversion only among Well-Known Text (WKT) and Well-Known Binary (WKB) formats. It also does not directly support importation and exportation of data, for which external tools are needed.

SQLite[28] is a C library that implements a self-contained DBMS. A SQLite database is stored in a single file in a portable way. *SpatiaLite* and *VirtualShape*[29] provide SQLite extensions with geographic functionality. SpatiaLite provides

[25] http://www.postgis.org/
[26] https://www.postgresql.org/
[27] https://www.mysql.com/
[28] http://www.sqlite.org/
[29] http://www.gaia-gis.it/gaia-sins/

functionalities for working with WKT and WKB formats, as well as support for spatial reference systems. In the same way, it allows the construction of geometries and provides some access and measurement operations. It also includes spatial predicates calculated with the bounding box, and allows the realization of transformations like rotation, scaling and displacement. VirtualShape also provides shapefiles importation and exportation, as well as the use of shapefiles directly from SQL.

4.4.3 Map Servers

This section describes some of the most widespread geographic information web services currently available.

UMN **MapServer**[30] is the most successful web service today. It is implemented with C++ and it can be used as a web service, using CGI, and as a library (supporting PHP, Python, Perl, Ruby, Java and C#). Although it has been developed for Linux, it can also be used within Windows following a much more complex installation and configuration process. MapServer web service allows the developer to use all types of data sources: ESRI datasources such as shapefiles or ArcSDE, different DBMS such as Oracle, PostGIS, or MySQL, and many other data formats using the GDAL/OGR library[31]. This library allows reading, and sometimes writing, information in a large number of formats. MapServer supports the main standards of geographic information: WMS (client and server), WMC[32], SLD[33], WFS (non-transactional), Filter Encoding[34], WCS[35] and GML[36].

Geoserver[37] has a big advantage versus the previous alternative: it is the easiest geographic information web service to install and configure. Geoserver is implemented with Java and it can be used as a web service through its J2EE application. It can be configured through its own user interface in a simple way. However, one drawback in this web service is its limited performance, as it is slower than other competitors.

GeoServer uses OpenLayers as integrated viewer for visualization of the geographic data within the tool itself, and additionally it allows to generate maps in many formats: Google Earth KML, PDF or SVG.

[30] http://mapserver.org/
[31] http://www.gdal.org/
[32] http://www.opengeospatial.org/standards/wmc
[33] http://www.opengeospatial.org/standards/sld
[34] http://www.opengeospatial.org/standards/filter
[35] http://www.opengeospatial.org/standards/wcs
[36] http://www.opengeospatial.org/standards/gml
[37] http://geoserver.org/

Geoserver can use different data sources with different levels of integration. Among the sources of data that we can consider mature are the DBMS PostGIS, Oracle, DB2, ArcSDE or Shapefile formats. It also supports other WFS, MapInfo, or MySQL data sources. This service supports, like the previous one, most of the standards for the interoperability and communication of geographic information, including WMS, SLD, WFS, Filter Encoding, WCS and GML.

Deegree[38] is probably the most complete web service discussed here. It is developed with Java and it can be used as a web service through its J2EE application. The main problem of Deegree is the complex configuration, in addition to having worse documentation than other tools. However, this web service provides good performance and it is the one tool supporting more OGC standards. Deegree includes its own GeoPortal. Data sources supported by Deegree include all types of DBMS (PostGIS, Oracle, and others such as SQL Server or DB2 that can be supported as generic SQL) as well as ESRI data sources (ArcSDE and Shapefile).

As we have already indicated, Deegree supports a multitude of standards. Like the previous ones, it supports WMS and SLD, WFS (transactional) and Filter Encoding, WCS or GML. It also provides functionalities in accordance with other standards: catalogues according to the CSW standard[39], interactions according to the Web Processing Service (WPS[40]) protocol.

MapGuide[41] is a tool released by Autodesk as open source software. This tool is part of the commercial software of Autodesk, and is implemented with C++. It can be used as a web service using CGI or as a library in PHP, Java or .NET. MapGuide provides ready-to-use web clients. MapGuide supports access to data using the FDO library[42]. This allows you to use ESRI formats such as Shapefile or ArcSDE, MySQL, ODBC or GDAL / OGR formats as data sources. You can also use it as a WMS and WFS client. This web service supports the basic standards of OGC WMS and WFS.

TileCache[43] is an implementation of a WMS-C performed by MetaCarta. It is implemented in Python and it can be used as a CGI web service. Instead of generating the image with each request to the WMS, TileCache stores a tile cache. In this way, for each zoom level, the cartography is rendered in advance as a set of cells. TileCache acts as a WMS-C service that can be used on clients that support this standard.

[38] http://www.deegree.org/
[39] http://www.opengeospatial.org/standards/cat
[40] http://www.opengeospatial.org/standards/wps
[41] https://mapguide.osgeo.org/
[42] http://fdo.osgeo.org/
[43] http://tilecache.org/

FeatureServer[44] is a feature server developed by MetaCarta, just like TileCache. It is also implemented in Python and can be used as a CGI web service. FeatureServer supports DBM, BerkeleyDB and PostGIS DBMS data sources. It can also function as a WFS service client, and you can use other OGR data sources (Shapefile, GML, etc.) or even Flickr images. This web service provides services based on different formats. It allows the use of JSON, GeoRSS or KML in different services for both input and output data. It also provides services for sending data in HTML, WFS (GML) or OpenStreetMap format.

GeoNetwork[45] is a metadata catalogue implemented in Java, and available as a web service in a J2EE application. It allows to manage and publish metadata of the spatial data infrastructure. GeoNetwork can use data sources based on ISO 19115 and 19139 metadata standards [fSe, fSk]. It also supports FGDC standards [Com], used in the United States, or the Dublin Core standard [KB07]. This server provides catalogues compatible with the OGC CSW standard, specific for the availability of metadata catalogues. It also supports the OpenSearch standards (for the provision of search results) and the Open Archives Initiative standard for content interoperability on the Web [LdS01].

4.4.4 Map Visualization Clients

After our brief introduction to map servers, in this section we describe some of the web map viewers more used nowadays. Some of these clients are targeted to use specifically with UMN MapServer (Chameleon, CartoWeb and Ka-Map), whereas others are server-independent and can be used with any server that follows the standards (Leaflet and OpenLayers).

The three web clients oriented to work with UMN MapServer have many things in common: they are implemented in PHP and JavaScript, oriented to AJAX and all of them are currently reducing their activity in favour of technology-independent projects. **Chameleon**[46] is designed to be extended through widgets. **CartoWeb**[47] provides a scalable architecture based on a highly modular design, and its main difference is that it implements a complete geoportal, providing the possibility of acting as a data server without requiring any other software artefact. **Ka-Map**[48] bases its operation on the use of tiling and PreCache. It is a project carried out in collaboration with OpenLayers, which we will see below. Ka-Map is the least

[44] http://featureserver.org
[45] http://geonetwork-opensource.org/
[46] http://chameleon.maptools.org/index.phtml
[47] http://cartoweb.org/
[48] http://ka-map.maptools.org/index.phtml

dependent on MapServer among the clients introduced so far, but it still needs some adaptation to make it truly independent.

Leaflet[49] is a server-independent web client. This is, it does not require any specific server side infrastructure (the three alternatives defined above require a PHP server, at least). It is implemented as a JavaScript library providing functionality to easily include maps in HTML pages. The only step needed to create a Leaflet map is to include Leaflet's JavaScript library in the HTML of the page. Its usage is very simple to web developers since it provides APIs based on web technologies, and extending its behaviour through plugins is also very easy. It is a very active project, with new versions every few months.

OpenLayers[50] is a standalone web client, similar to Leaflet. It is developed in JavaScript, and makes extensive use of AJAX, tiles and cache. OpenLayers is a project with a lot of activity and synergy with many other projects in the area of geographic information. For this reason it is a more complete tool than Leaflet and provides a greater number of controls and options, although Leaflet can provide greater performance in simple maps. OpenLayers is also more complex to use and extend than Leaflet. OpenLayers provides functionalities as a JavaScript library for creating and manipulating maps and layers from different data sources. It can act as a client for WMS, WFS, WMC services, and use GeoRSS, KML, or GeoJSON data sources, among others.

4.5 Summary

In this section we have described what is a geographic information system, providing many examples about their functionalities and features, as well as several examples of GIS applications that have been in the market for some time. We could see that GIS applications can be used within many different fields without relationship among them but the functionalities provided by them are usually the same. Finally, we have also shown many common software artefacts providing GIS features, both commercial and open source.

The information provided in this section serves to the purpose of introducing the reader to geographic information systems. During the process to define our software product line, we need to formally describe a generic GIS both from a functional and a non-functional point of view, and therefore this introduction is required to provide a better understanding of the process.

[49] http://leafletjs.com/
[50] http://openlayers.org/

5

Requirements Analysis: identifying our features

5.1 Introduction

After the definition of the methodology to guide the design of our software product line, and after providing some concepts to introduce the reader into geographic information systems, our domain, we are now ready to begin the actual process. This chapter is the starting point for the definition of our software product line in the domain of web-based geographic information systems. That is, in this chapter we begin the application of our methodology. The initial stage of this methodology is the *requirements analysis* (see Figure 5.1). This stage is composed by two steps: *domain analysis* and *product planning*.

The domain analysis step consists on the identification and definition of the requirements for our products. These requirements are contrasted with related work from literature on the topic of software product lines for GIS. In the second step, product planning, we analyse existing products of our domain. These analysis serve as validation for our requirements and, at the same time, to classify the features into mandatory, optional or alternative regarding their appearances in the existing products. The output of this phase is the feature model of our product line.

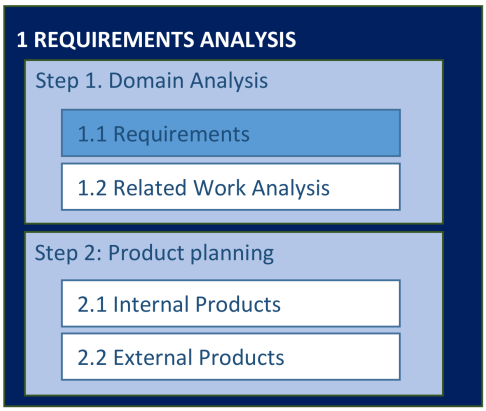


Figure 5.1: Requirements analysis stage

5.2 Domain Analysis

The first step of the requirements analysis stage is the domain analysis (see the step 1.1, Figure 5.1). It is composed by two tasks or sub-steps: requirements and related work analysis, which are detailed in the following sections.

5.2.1 Requirements for our products

Having the collaboration of Enxenio, and since one of the goals of the thesis is to guarantee the generation of products that can be really used in the industry, it makes sense that experts of the company help deciding the product family and the requirements of the platform. Therefore, we have based our decisions for this step (see the step 1.1.1, Figure 3.2) not only in our previous knowledge but also in the expertise on web-based GIS of project managers from Enxenio, to whom we have conducted discussions and interviews. The requirements of the platform were extracted from these meetings. We classified these requirements in four different groups:

R1) *Data Management*, which includes every requirement related to how the data, both alphanumeric and geometrical, is stored, introduced in the system and internally processed.

R2) *Graphical User Interface* requirements, related to the way the alphanumeric data is accessed through the web interface, and which data is provided to the final user.

R3) *Map Viewer* requirements, regarding how the geographic data must be shown in the web application.

R4) *User Management* requirements, since there are some required functionalities regarding authentication of users and management of their roles, as in any other web application.

We found 17 requirements belonging to these four categories or groups. We do not fully detail every requirement since it would be long and boring whereas it would not provide more value to this work. The complete list can be seen in Table 5.1.

Id	**Description**
Group R1: Data Management	
R1.1	The spatial database management system of each product can be chosen among the alternatives complying with the standard: PostgreSQL with PostGIS, MySQL and Oracle Spatial.
R1.2	The spatial data is complementary to alphanumeric data. Therefore, there must be a way to access the latter using current web standards such as REST services.
R1.3	Spatial data is hard to input manually. For example, manually defining the perimeters of the administrative divisions of a nation is a non assumable task. Therefore, data input procedures must be varied and comply with the standards in GIS, such as allowing the importation of shapefiles or the digitizing of maps.
R1.4	The products must be able to run typical GIS operations, such as Route Calculation or Addresses Geolocalization.
Group R2: Graphical User Interface	
R2.1	Every functionality of the product can be accessed through an element in the menu of the application.
R2.2	The web application, as any other application, should allow the input of data trough forms.
R2.3	Data of the elements handled by the application can be viewed using lists. These lists may be sorted, and the elements shown in them filtered. For each element, there can be links to a form to edit it, or a map to view its position if it is a geolocalized element.

Id	Description
R2.4	Web applications may have, apart from dynamic content, some static pages to show information not changing, such as a contact or welcome pages.
Group R3: Map Viewer	
R3.1	The map server can be chosen among GeoServer and Deegree, which are two popular alternatives.
R3.2	Typical GIS tools like zooming, panning, measuring distances or objects, geo localizing the user or showing the context information of the map should be available in the products.
R3.3	In the map viewer, there should be a way to select whether to show or not different layers, as well as setting their opacity, sort them, etc.
R3.4	The map centre can be set to the user position, or to a specific region of the world.
R3.5	When a user clicks on an element of a map, a popup must show the information regarding this element.
Group R4: User Management	
R4.1	Anonymous users can register and login. The activation of the accounts can be manual or automatic using an email.
R4.2	Information of the users can be updated, and the products may store additional information about them, such as the birthday or the social networks accounts.
R4.3	The type of security can be chosen among the most common alternatives used nowadays in the industry.
R4.4	LDAP server can be used for authentication.

Table 5.1: Requirement list

5.2.2 Features derived from the set of requirements

From these requirements we derived the set of features that the SPL platform must provide in order to generate the desired products. In Table 5.2, the features are identified and described, as well as associated with the requirement from which they are derived, keeping the traceability. At this point, the features are just listed but we still do not have the information regarding to whether they are mandatory or optional, nor which features are dependent to each others. This information is

obtained in a further step (Section 5.3) by analysing existing products.

Req.	Feature Id	Description
R1	DataManagement	Product stores data and provides functionalities for its management
R1.1	DM_SpatialDatabase	Existence of a Spatial Database Management System
R1.1	DM-SD_PostGIS	Using PostGIS (and PostgreSQL) as the DBMS
R1.1	DM-SD_MySQL	Using MySQL as the DBMS
R1.1	DM-SD_OracleSpatial	Using Oracle Spatial as the DBMS
R1.2	DM_DataServer	Existence of a service providing standard (non-spatial) data
R1.3	DM_DataInput	Users can input geographic and alphanumeric data
R1.3	DM-DI_DataFeeding	Existence of services allowing the user to import data
R1.3	DM-DI-DF_Shapefile	Importation of shapefiles: the user can load alphanumeric and spatial information from shapefiles into the application
R1.3	DM-DI-DF_Raster	Importer of raster files
R1.3	DM-DI-DF_Network	Importer of network files
R1.3	DM-DI_Digitizing	Existence of services allowing the user to digitize geographic data
R1.3	DM-DI-D_Form	Digitizing from forms: the user can edit the geographic information of new or existing elements using a map within a standard form
R1.3	DM-DI-D_Map	Digitizing from maps: the user can create batches of new geolocalized elements by drawing their shape on a map, and setting their alphanumeric data in a subsequent step
R1.4	DM_Algorithmics	Users can execute GIS procedures

Req.	Feature Id	Description
R1.4	DM-A_Connectivity	Users can execute connectivity procedures
R1.4	DM-A-C_RouteCalculation	Route calculation from an origin and destiny points within a map
R1.4	DM-A-C_NetworkTracing	Check which nodes of a network can be reach from a given node
R1.4	DM-A-C_ConectivityCheck	Check which nodes of a network are not connected
R1.4	DM-A_Geolocalization	Users can execute geolocalization procedures
R1.4	DM-A-G_Addresses	Geolocalization of textual addresses
R1.4	DM-A-G_Documents	Geolocalization of geographic named entities on textual documents
R2	GraphicalUserInterface	Existence of a graphical user interface
R2.1	GUI_Menu	Standard menu with links and submenus grouping menu items
R2.1	GUI-M_Top	An horizontal menu placed in the top of the page
R2.1	GUI-M_Bottom	An horizontal menu placed in the bottom of the page
R2.1	GUI-M_Right	A vertical menu placed in the right side of the page
R2.1	GUI-M_Left	A vertical menu placed in the left side of the page
R2.2	GUI_Forms	Forms for visualizing and editing of elements in the data model
R2.2	GUI-F_Editable	Capability of a form to modify the information of the associated instance
R2.2	GUI-F_Creatable	Capability of a form to create new instances of the associated entity
R2.2	GUI-F_Removable	Capability of a form to remove the currently loaded instance

Req.	Feature Id	Description
R2.2	GUI-F-R_ConfirmationAlert	Showing a modal warning alert when removing an element in a form
R2.3	GUI_Lists	Lists of elements of data from the data model
R2.3	GUI-L_Sortable	Lists can be sorted by the user
R2.3	GUI-L_Filterable	Lists elements can be filtered by the user
R2.3	GUI-L-F_RowFilter	The filters are applied for each row/property of the elements listed
R2.3	GUI-L-F_BasicSearch	There is a field to make search over all properties of the elements listed
R2.3	GUI-L_LocateInMap	For elements which have a geographical property, there is a link so an user can view the same element within a map viewer
R2.3	GUI-L_ViewListAsMap	For elements which have a geographical property, show all the elements within a map viewer
R2.3	GUI-L_FormLink	There is a link for each element that leads to a form where the element can be edited
R2.4	GUI_StaticPages	There are some static pages defined by the analyst
R2.4	GUI-SP_Management	The static pages can be created, modified and removed from the final application
R3	MapViewer	User can view geographic information using a map viewer
R3.1	MV_MapServer	Cartography is retrieved from an internal map server instead of using publicly available servers
R3.1	MV-MS_GeoServer	The map server used is GeoServer
R3.1	MV-MS_Deegree	The map server used is Deegree

Req.	Feature Id	Description
R3.2	MV_Tools	The map viewers provides tools so the user can interact with it
R3.2	MV-T_Pan	The map can be moved with the mouse, changing the portion of Earth viewed
R3.2	MV-T_Zoom	The user can zoom in/out the map
R3.2	MV-T_ZoomWindow	The user can zoom directly to a rectangle draw with the mouse
R3.2	MV-T_Measure	There is a tool that allows to measure different things in the map
R3.2	MV-T-M_Distance	Distances between two points can be measured
R3.2	MV-T-M_Line	The user can draw a line composed of multiple segments and measure its length
R3.2	MV-T-M_Polygon	The user can draw a polygon and measure its area
R3.2	MV-T-M_MapElement	The user can measure an element shown in the map, i.e., a geographic property of an instance
R3.2	MV-T_Export	The current view of the map can be exported to a file so the user can download it
R3.2	MV-T-E_SetScale	The user can choose the scale of the map before exporting it
R3.2	MV-T-E_ShowLegend	The user can choose whether the legend of the map is shown or not in the exported image
R3.2	MV-T-E_DRM	The user can add DRM protection to the created file
R3.2	MV-T-E_Type	File types that the user can choose to download the map, in case that there is more than one

Req.	Feature Id	Description
R3.2	MV-T-E-F_PNG	Current view of the map is exported as PNG
R3.2	MV-T-E-F_PDF	Current view of the map is exported as PDF
R3.2	MV-T-E-F_URL	A custom URL is created for the current view of the map
R3.2	MV-T_Filterable	Map elements can be filtered by the user
R3.2	MV-T-F_RowFilter	The filters are applied for each property of the elements listed
R3.2	MV-T-F_BasicSearch	There is a field to search over all properties of the elements in the map
R3.2	MV-T_UserGeolocation	The map can show the position of the user, geolocating him or her
R3.2	MV-T_InformationMode	Instead of panning the map on click, information about the element clicked is shown in this mode
R3.2	MV-T_ViewMapAsList	Shows a list with the current elements of the map
R3.2	MV_ContextInformation	The map viewer shows information regarding the current state of the view
R3.2	MV-CI_Map	The context information window has a mini map showing the position of the current view within the whole map
R3.2	MV-CI_Scale	Adds scale information to the context information
R3.2	MV-CI_CenterCoordinates	The coordinates of the center of the current view are shown
R3.2	MV-CI_Dimensions	The context information shows the current dimensions of the map
R3.3	MV_LayerManagement	Layers of the map can be manipulated from within the map viewer
R3.3	MV-LM_Order	Layers of the map can be sorted
R3.3	MV-LM_HideLayer	Layers of the map can be hidden

Req.	Feature Id	Description
R3.3	MV-LM_Opacity	Opacity of the layers can be set
R3.3	MV-LM_Style	Style of the layers can be set
R3.3	MV-LM_ExternalLayer	The user can add external layers to the map from a map server
R3.3	MV-LM_Clustering	Elements of the layers can be clustered, showing the number of elements when the zoom is out and the singular elements when the zoom is in
R3.4	MV_MapCenter	The initial view of the map can be set
R3.4	MV-MC_BBox	The initial view of the map is set to an specific bounding box
R3.4	MV-MC_UserPosition	The initial view of the map is centred on the location of the user
R3.5	MV_DetailOnClick	When the user click on the elements of the map, a popup with the element information is shown
R4	UserManagement	The web application distinguish between different types of users and handles access permissions
R4.1	UM_Registration	Users can be registered
R4.1	UM-R_ByAdmin	Users can be registered by the administrator
R4.1	UM-R_Anonymous	Anonymous users can register themselves
R4.1	UM_Authentication	Users can log in the web application
R4.1	UM-A_RememberPass	The users have the choice to store the session so next time the load the web application they do not have to write the password again
R4.1	UM-A_RecoverPass	The users can recover their password if they forgot it
R4.1	UM_AccountActivation	The user accounts have to be activated before the first log in

Req.	Feature Id	Description
R4.1	UM-AA_ByEmail	The users activate their accounts by receiving an email with a link
R4.1	UM-AA_ByAdmin	The accounts can be activated by the administrator
R4.2	UM_UpdateEmail	Registered users can update their email
R4.2	UM_UpdatePassword	Registered users can change their password
R4.2	UM-UP_ByUser	Registered users can change their password themselves
R4.2	UM-UP_ByAdmin	Administrator users are the only who can change the passwords of users
R4.2	UM_UserProfile	Extra information for user accounts is stored
R4.2	UM_UserCRUD	Administrator can access to lists of users and create, edit and remove users
R4.3	UM_SecurityType	The type of security of the web application
R4.3	UM-ST_Session	Security is provided by HTTP sessions
R4.3	UM-ST_JWT	Security is provided by JSON Web Tokens
R4.4	UM_LDAP	The web application is linked to a LDAP, so users do not need to register but they can use their existing accounts in the LDAP

Table 5.2: Feature list

Related work about SPLE applied on GIS is very scarce (see the step 1.1.2, Figure 3.2). We have found only one series of works regarding SPLE for GIS [PBC+12, BCA+13, BCP+14, BCPA14, BPCA16, BCPA16], and a work not strictly on the field of SPLE but which also explores automatic generation of web-based GIS [DMG13].

[BCP+14] is a good representative of the series of works by professor Bucella. It defines a SPL for GIS in the marine ecology, showing a non-exhaustive list of features.

We can see some similarities with our features, such as "Calculate distances (meters) between points in specific zones" or "Panning & zoom", but their SPL is focused on a much more concrete domain and most of the features are very specific to this domain, such as "Look for fishing areas with similar characteristics" or "Abundance of biological data of stations by tables". Therefore, we cannot use this work as reference for our generic GIS products.

In [DMG13], a tool for the automatic generation of web-based GIS from a data model is shown. However, in this case the design of the data model is the only variability of the generated applications, having all of them the same features. We have this kind of variability into account, but it affects to the domain of the application engineering, ergo, the derivation phase Chapter 7.

5.3 Product Planning: analysing existing products

The next step of the requirements analysis stage is the *product planning* (see the step 1.2, Figure 5.1). In this step we contrast the set of features identified and described in the previous section with existing products. Specifically, we use three products developed by Enxenio, which are representative examples of applications that our SPL should be able to produce. Therefore, the set of features identified and the ones existing in these products must match. These products are **webEIEL**, a GIS developed for the Provincial Council of A Coruña (Spain); **Galician Cultural Heritage**, a GIS to promote cultural and tourist points of interest designed to be used in desktop computers; and **Via Maps**, a mobile GIS to promote points of interest designed to be used in mobile devices with low bandwidth. Furthermore, we also contrast our decisions against two well-known external products, **Google Maps** and **OpenStreetMap**. Before the proper product planning step, we describe all the products taken into account.

5.3.1 Description of the analysed products

The three products by Enxenio, described below, cover a wide range of user needs in many different concerns. For example:

1) User expertise: webEIEL is expected to be used by experts, whereas both the Galician Cultural Heritage and Via Maps web applications are used by casual users.

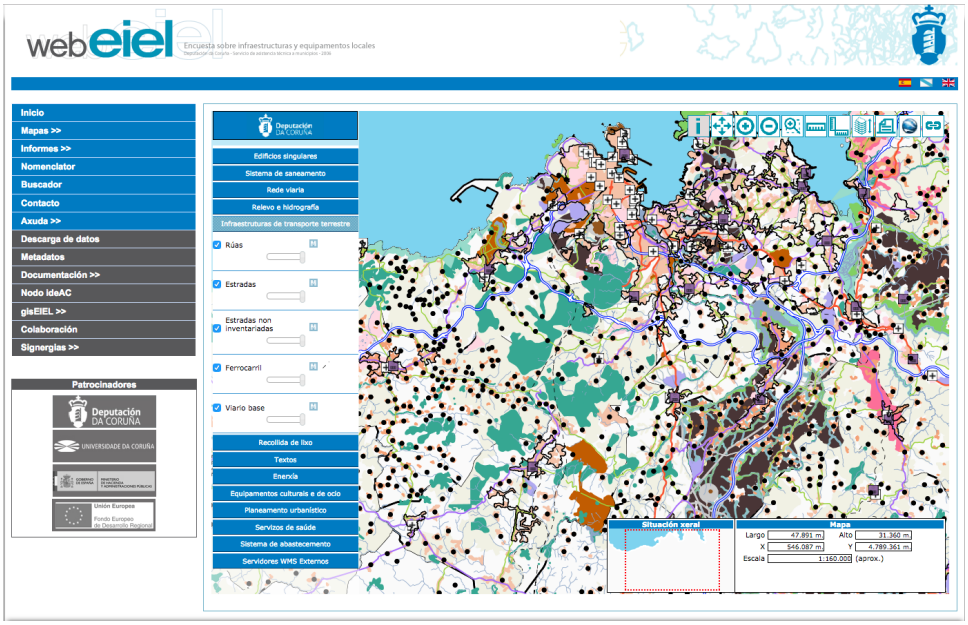


Figure 5.2: WebEIEL screenshot

2) Map viewer tools: the level of personalization of the map viewers vary among the applications, from "as many tools as possible" (webEIEL), to "some personalization tools" (Galician Cultural Heritage), to "almost none" (Via Maps).

3) Expected bandwidth of the users: webEIEL and Galician Cultural Heritage users are expected to have a proper broadband connection, whereas Via Maps was designed to be used with a limited bandwidth connection.

4) Device used: both webEIEL and Galician Cultural Heritage are designed for desktop computers, but Via Maps is preferably viewed with a mobile device.

WebEIEL[1] is a GIS developed for the Provincial Council of A Coruña (Spain) between 2000 and 2008. It is currently being used by five additional Provincial Councils. WebEIEL can be used to record and digitize infrastructures and facilities such as administration buildings, road networks, supply and sanitation networks, health centres, etc. The information is stored in a DBMS using PostGIS and it can

[1] WebEIEL. Encuesta sobre Infraestructuras y Equipamiento Local: http://webeiel.dicoruna.es/gl

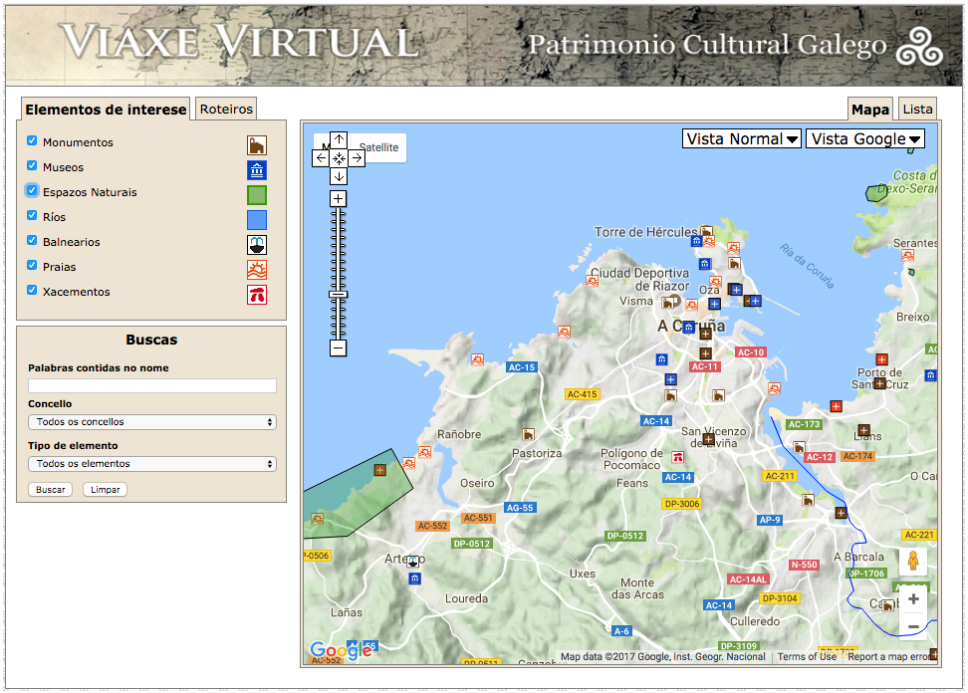


Figure 5.3: Galician Cultural Heritage screenshot

be managed using a desktop application. Information available in the GIS can be loaded massively from standard formats, such as *shapefiles* or manually digitizing using forms and maps.

The resulting GIS provides a huge amount of information, accessed through a web application. This web-based GIS application (see Figure 5.2) can be used to browse, query and print the information, with the common set of map tools of GIS applications (zoom and pan the map, select the visible layers, etc.). Furthermore, it provides additional tools such as measuring distances and surfaces, exporting information, including layers from external sources, etc.

The Galician Cultural Heritage web-based GIS application[2] is a system to promote cultural and tourist heritage that can be used to display on a map natural, cultural and tourist points of interest and to provide detailed cataloguing information (see Figure 5.3). It also displays routes, although unordered, and it provides a tool to search elements by name, type, and geographic location. All the

[2]Galician Cultural Heritage: http://www.patrimonioculturalgalego.org/ViaxeVirtual/AmosarViaxeVirtual.do

Figure 5.4: Via Maps screenshot

information is stored in a DBMS using PostGIS and it can be managed with a web interface that can be used to create and update the geographic and cataloguing information of the elements. The information can be browsed and filtered using lists and maps. The map viewer provides tools to select the visible layers, as well as zooming and panning the map, and retrieving the information of any element. However, the advanced map tools provided by webEIEL are not required in this application.

Via Maps[3] is another web-based GIS to promote cultural heritage. However, it is focused on mobile devices in low-bandwidth environments such as limited public Wi-Fi networks. It was developed as part of the ENVIA project[4], whose goal was

[3]Via Maps: https://madrid.via1101.pv.enxenio.net/Servicios/visorgis/m/mapa/verMapa.htm

[4]ENVIA project: Environment for the creation of a cloud of services for the intelligent pavement. Ministry of Industry, Tourism and Trade Avanza plan (Ref. TSI-020302-2011-6) - http://www.

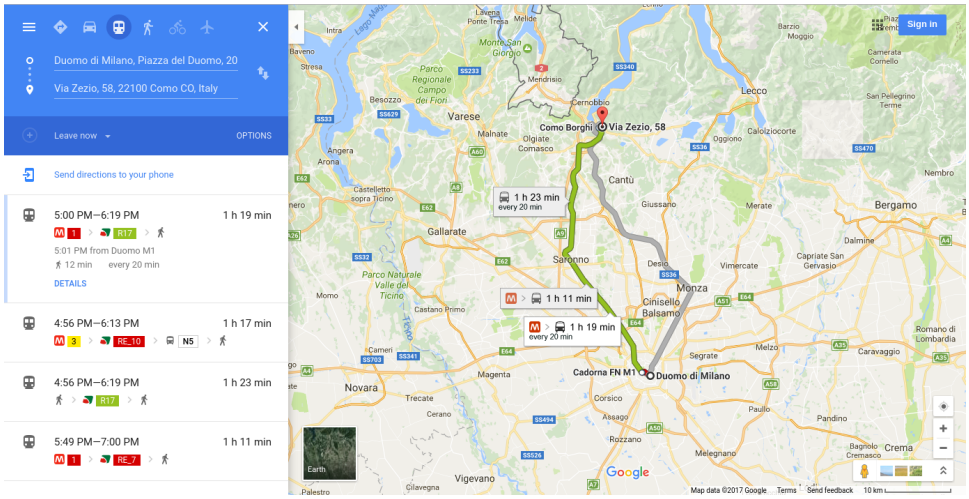


Figure 5.5: Google Maps screenshot

providing the cities with infrastructures and software that enabled the citizens to enjoy free Internet access (within the legislative limits) and different social, cultural and commercial services. The anonymous user view of the prototype that was deployed in Madrid can be seen in Figure 5.4. These users have tools to find resources, and they can also use the geopositioning capabilities of their mobile device to locate themselves on the map and find nearby elements. Given that the network connection was expected to have a low bandwidth, the user interface was designed as simple as possible. The information is stored in a DBMS using PostGIS and it can be managed with a complete web-based application that can be used to create, browse and update the elements.

Apart from these specific developed products, Google Maps and OpenStreetMaps are two of the most used GIS applications, and their set of features is an indicator of users expectations on GIS. Therefore, they are an important part of our process.

Google Maps[5] is a web mapping service provided by Google. It offers several different layers, such as a satellite imagery layer, a traffic layer, etc., and its main feature is route planning (Figure 5.5). Google Maps was launched in 2005, and it sooner became the most popular web-based GIS. Apart from providing cartography and route calculation, it has many advanced features which are linked with the

viainteligente.com/envia.html

[5] https://www.google.com/maps

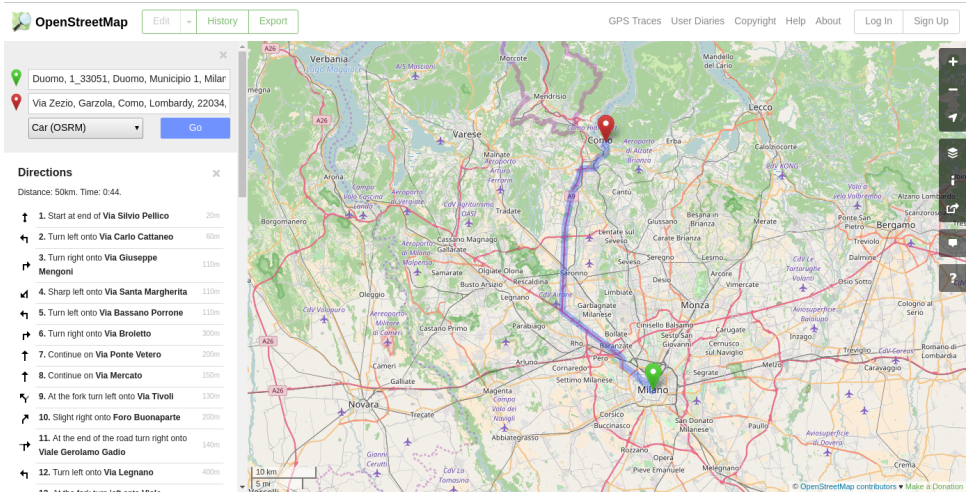


Figure 5.6: OpenStreetMap screenshot

Google ecosystem, and the application is used embedded in many other Google web applications. For example, an user can visualize its location history or edit positions on a map to identify the places where he or she travelled.

OpenStreetMap[6] (OSM) is a free, editable map of the whole world that is being built by volunteers largely from scratch and released with an open-content license. Its creation was motivated by restrictions on use or availability of map information across much of the world. The data of OSM, which is the main point of this GIS, can be generated by any user or linked application, as well as used by anyone. Therefore, it is presented as an alternative to proprietary tools such as Google Maps, and many mobile applications are using it as the main datasource. The site is supported by the OpenStreetMap Foundation, a non-profit organisation, and it only uses Open Source technology from the map interface to the underlying data access API. Apart from cartography visualization and geographic data managing, OSM also provides route calculation, as we can see in Figure 5.6.

5.3.2 Feature validation

Apart from validating the coherence of our identified features, with this step we are also able to set the priority of each feature, and to decide which ones are common to every product, therefore mandatory, and which appear only in some of them,

[6] https://www.openstreetmap.org/

being then optional features. In Table 5.3 we can see all the non-abstract features (features that effectively are part of the product, this is, they are not features to group other subfeatures) identified in Table 5.2. We have used in the table the number of times that each feature appears in the products is taken into account to calculate the priority of each feature. If the feature appears in any of the products, the priority is the sum of the number of appearances in the products, being 5 the maximum priority that a feature can achieve. In case the feature is not included in any of the analysed applications, the GIS experts in Enxenio have set the priority between 0 and 1 to determine how important the feature is for the SPL.

Feature	**SME existing apps**			**External apps**		**Pri.**
	Web EIEL	Cult. Her-itage	Via Maps	Google Maps	OSM	
DataManagement	X	X	X			3
DM_SpatialDatabase	X	X	X			3
DM-SD_PostGIS	X	X	X			3
DM-SD_MySQL						0
DM-SD_OracleSpatial						0
DM_DataServer	X	X	X			3
DM_DataInput		X	X		X	3
DM-DI_DataFeeding						0.9
DM-DI-DF_Shapefile						0.9
DM-DI-DF_Raster						0.6
DM-DI-DF_Network						0.3
DM-DI_Digitizing		X	X		X	3
DM-DI-D_Form		X	X		X	3
DM-DI-D_Map					X	1
DM_Algorithmics				X	X	2
DM-A_Connectivity				X	X	2
DM-A-C_RouteCalculation				X	X	2
DM-A-C_NetworkTracing						0.3
DM-A-C_ConectivityCheck						0.3
DM-A_Geolocalization				X	X	2
DM-A-G_Addresses				X	X	2

Feature	**SME existing apps**			**External apps**		**Pri.**
	Web EIEL	Cult. Her-itage	Via Maps	Google Maps	OSM	
DM-A-G_Documents						0.6
GraphicalUserInterface	X	X	X	X	X	5
GUI_Menu	X	X	X	X	X	3
GUI-M_Top		X	X		X	3
GUI-M_Bottom			X			1
GUI-M_Right						0
GUI-M_Left	X			X		2
GUI_Forms	X	X	X		X	4
GUI-F_Editable	X	X	X		X	4
GUI-F_Creatable	X	X	X		X	4
GUI-F_Removable	X	X	X		X	4
GUI-F-R_ConfirmationAlert		X				1
GUI_Lists		X	X			2
GUI-L_Sortable		X				1
GUI-L_Filterable		X				1
GUI-L-F_RowFilter						0.9
GUI-L-F_BasicSearch		X				1
GUI-L_LocateInMap			X			1
GUI-L_ViewListAsMap						0.8
GUI-L_FormLink						0.9
GUI_StaticPages	X			X	X	3
GUI-SP_Management						0.5
MapViewer	X	X	X	X	X	5
MV_MapServer	X	X	X			3
MV-MS_GeoServer	X	X	X			3
MV-MS_Deegree						0
MV_Tools	X	X	X	X	X	5
MV-T_Pan	X	X	X	X	X	5
MV-T_Zoom	X	X	X	X	X	5

Feature	**SME existing apps**			**External apps**		**Pri.**
	Web EIEL	Cult. Heritage	Via Maps	Google Maps	OSM	
MV-T_ZoomWindow	X					1
MV-T_Measure	X					1
MV-T-M_Distance	X					1
MV-T-M_Line	X					1
MV-T-M_Polygon	X					1
MV-T-M_MapElement						0.7
MV-T_Export	X			X	X	3
MV-T-E_SetScale	X					1
MV-T-E_ShowLegend						0.3
MV-T-E_DRM						0
MV-T-E_Type	X					1
MV-T-E-F_PNG						0.8
MV-T-E-F_PDF	X					1
MV-T-E-F_URL	X			X		2
MV-T_Filterable		X		X	X	3
MV-T-F_RowFilter		X				1
MV-T-F_BasicSearch		X		X	X	3
MV-T_UserGeolocation			X	X	X	3
MV-T_InformationMode	X					1
MV-T_ViewMapAsList		X				1
MV_ContextInformation	X				X	2
MV-CI_Map	X				X	2
MV-CI_Scale	X				X	2
MV-CI_CenterCoordinates	X					1
MV-CI_Dimensions	X					1
MV_LayerManagement	X	X	X		X	4
MV-LM_Order	X					1
MV-LM_HideLayer	X	X	X		X	4
MV-LM_Opacity	X					1

Feature	**SME existing apps**			**External apps**		**Pri.**
	Web EIEL	Cult. Heritage	Via Maps	Google Maps	OSM	
MV-LM_Style	X					1
MV-LM_ExternalLayer	X				X	2
MV-LM_Clustering						0.7
MV_MapCenter	X			X	X	3
MV-MC_BBox	X					1
MV-MC_UserPosition				X	X	2
MV_DetailOnClick	X	X	X			3
UserManagement		X	X	X	X	4
UM_Registration				X	X	2
UM-R_ByAdmin						0.9
UM-R_Anonymous				X	X	2
UM_Authentication		X	X	X	X	4
UM-A_RememberPass		X		X	X	3
UM-A_RecoverPass				X	X	2
UM_AccountActivation						0.5
UM-AA_ByEmail						0.5
UM-AA_ByAdmin						0.5
UM_UpdateEmail				X	X	2
UM_UpdatePassword		X	X	X	X	4
UM-UP_ByUser		X	X	X	X	4
UM-UP_ByAdmin						0.2
UM_UserProfile				X	X	2
UM_UserCRUD						0.9
UM_SecurityType		X	X			2
UM-ST_Session		X	X			2
UM-ST_JWT						0.3
UM_LDAP						0.5

Table 5.3: Product planning

It is somehow curious that Enxenios experts asked for some costly features that were not included in any of the analysed products, such as the possibility of use any of the three main spatial database management systems, MySQL, Oracle Spatial or PostGIS. The cost of implementing these features is very high, and it seems unreasonable to do that when in the analysis of existing products we have found that all of them use the same DBMS, PostgreSQL with PostGIS.

One of the advantages of a SPL is that the set of features provided can evolve over time. This way, not every feature of the SPL platform needs to be included at the same time, and every time a new product needs to be generated, if a new feature is required and it is generic and useful enough, then it can be added to the SPL platform. Therefore, the priority value on the table is very useful to determine which features should be included sooner into the platform, and which ones can be postponed. Consequently, MySQL and Oracle Spatial are clear examples of features that can be included in the platform as soon as a product requires them.

Moreover, we found some features in the products that were not initially included in the requirements, but that were interesting to the Enxenio's experts, so we have iterated over this step including the new features. This was the case of "Filter elements over a map" or "View a map as a list", for example.

5.4 Feature Model of our Software Product Line

Finally, we obtained the feature model of our SPL, which is very large so we have divided its representation in several figures. In Figure 5.7, the first level features of the feature model are shown without any of their subfeatures. We can see each of its features in its own figure: the *data management* feature is shown in Figure 5.8, the *graphical user interface* feature is shown in Figure 5.9, the *map viewer* feature is shown in Figure 5.10 and the *user management* feature is shown in Figure 5.11.

Globally, we can see that features appearing in every product are mandatory, features appearing in some of the applications are optional and the ones not appearing at all have been removed from the feature model. We also decided the types of aggregation (between XOR and OR) depending on the appearance of the features. For example, every product analysed included both the features of zooming and panning the map viewer. Therefore, both features are mandatory for products including a map viewer. Nevertheless, not all of the map viewers include the feature of showing a context information window, so this option remain optional.

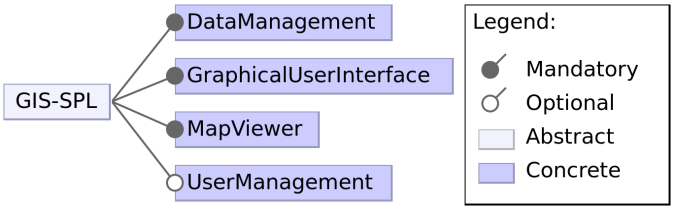


Figure 5.7: First level features of the resulting feature model

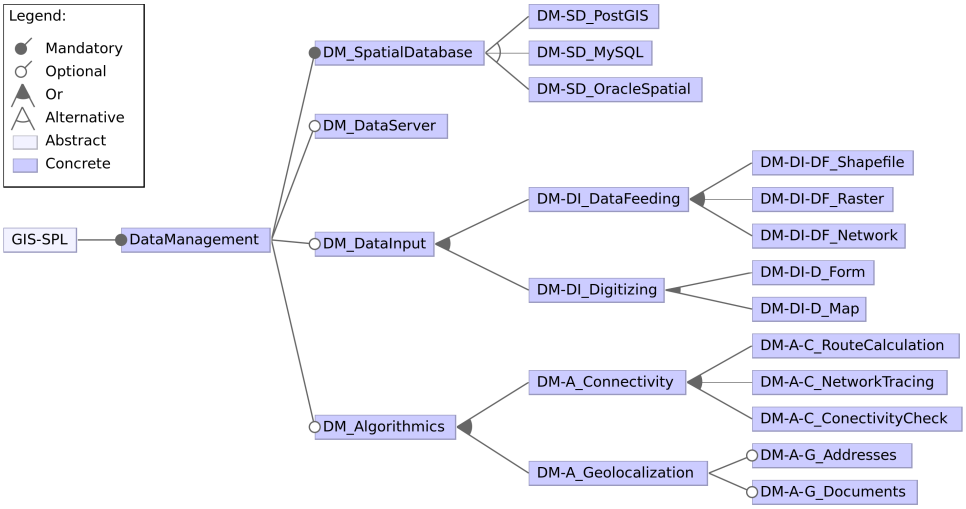


Figure 5.8: Data feature and its subfeatures

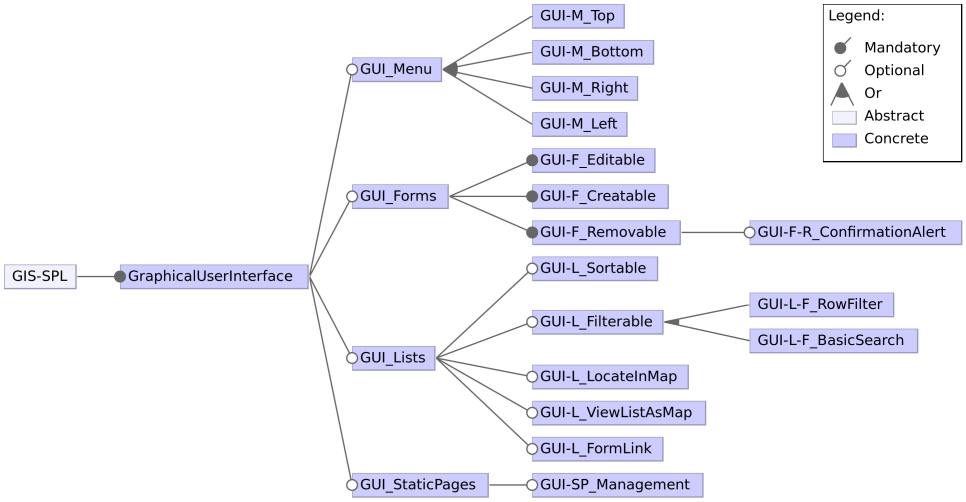


Figure 5.9: Gui feature and its subfeatures

5.5 Summary

In this section we have showed the results obtained after the first stage of our methodology, the requirements analysis. We have identified the list of requirements for our domain, web-based geographic information systems, from our experience and conversations with experts from Enxenio. From these set of requirements, we have derived the proper features, and we have studied existing web-based GIS in order to both validate the identified features and to establish the relationship between these features.

To be more concrete, during this step we have analysed products developed by the associated company, Enxenio, but also from companies external to our process. By doing this analysis, we have also obtained information to prioritize the features. This way, when the actual development of the components starts, we already have information about which feature should be available first due to its importance.

Lastly, we have shown the complete feature model of our product line.

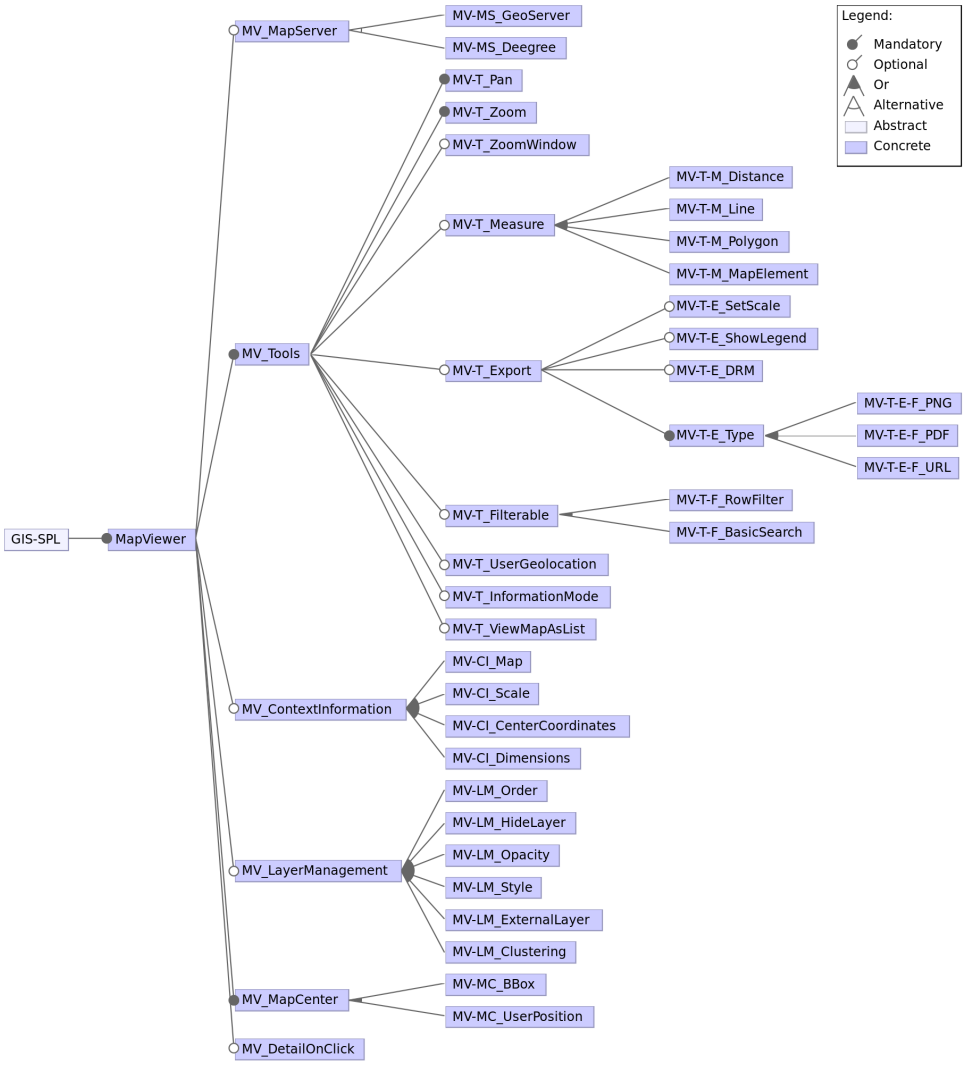


Figure 5.10: Map viewer feature and its subfeatures

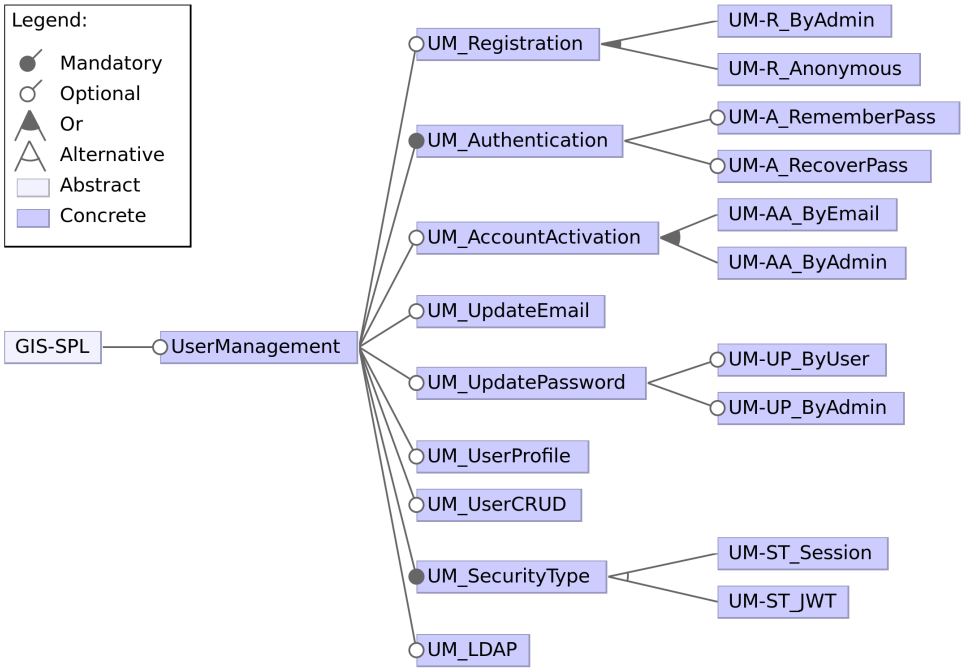


Figure 5.11: User management feature and its subfeatures

6

Architecture Design: generic architecture for Web-based GIS applications

6.1 Introduction

At this point we have showed the results obtained after the first stage of our methodology, the requirements analysis. It is important to remember that our process is iterative, and we have gone several times over every stage. Therefore, the results we address in this document are the final ones obtained when there were no more iterations. In this case, the output of the first stage is the feature model of our product line.

The current section corresponds with the second stage of our methodology (see Figure 6.1), the architecture design. The goal of this stage is to design the product line architecture (PLA) shared by all the products built within our SPL and to identify every component or service in it. In order to design the PLA, we mostly follow the guidelines from [NBM13], as we have already explained when describing our methodology (see Chapter 3), starting from the choice of the reference architecture, but we also have added our own tasks in which the experts from Enxenio would provide their knowledge: analysis of products architectures and technology analysis.

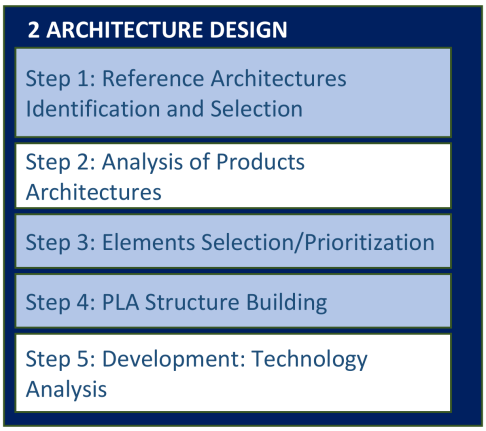


Figure 6.1: Architecture design stage

During this chapter, we consider the concepts of *service* and *component* as equivalent. We use mostly the former because it is the one employed by the Open Geospatial Consortium (OGC) in their documentation.

6.2 Reference architectures identification and selection

There have been many attempts to define a reference architecture for web-based geographic information systems. ESRI[1], the leading company of the sector, has been proposing system architectures that include their products for decades. A constant aspect of these architectures is that they are layer-based (3-tiered or n-tiered) [ESR17]. However, in addition to depending heavily on ESRI products, the architecture does not provide enough detail for the components.

Another major driving force has been the European Commission through the INSPIRE directive [INS] that establishes an infrastructure for spatial information in the European Community. The INSPIRE directive has been reflected in a collection of implementing rules that describe with much detail data models and web services to store and publish geographic information. Even though INSPIRE provides much detail on specific services, and it defines a service bus-based system architecture [INS08], it does not describe the way in which the services are expected to interact in the architecture and it does not provide any detail on the services

[1] http://www.esri.com/

that are out of the scope of the directive (e.g., the functionality of the web client).

The stakeholders that have defined a large collection of standards for GIS that are currently followed by most software libraries are the Open Geospatial Consortium (OGC) and the International Organization for Standardization (through ISO/TC 211 and the 19100 standard collection). Particularly, the OpenGIS Service Architecture Version 4.3 [Per02], which is the same document as ISO 19119:2005 [fSg], defines a geographic services architecture identifying architecture patterns for service interfaces and the relationships among them, and providing guidelines for the selection and specification of geographic services from platform-neutral and platform-specific perspectives. The system architecture is based on Reference Model of Open Distributed Processing (RM-ODP) [ISO09] and it is described from the five RM-ODP viewpoints: the *enterprise* viewpoint, the *computational* viewpoint, the *information* viewpoint, the *engineering* viewpoint and the *technology* viewpoint. Furthermore, a large number of services are identified, described and categorized. We selected ISO 19119:2005 as the reference architecture given the high level of detail provided. ISO 19119:2005 is revised in ISO 19119:2016 [fSh], but without any changes that would affect our work.

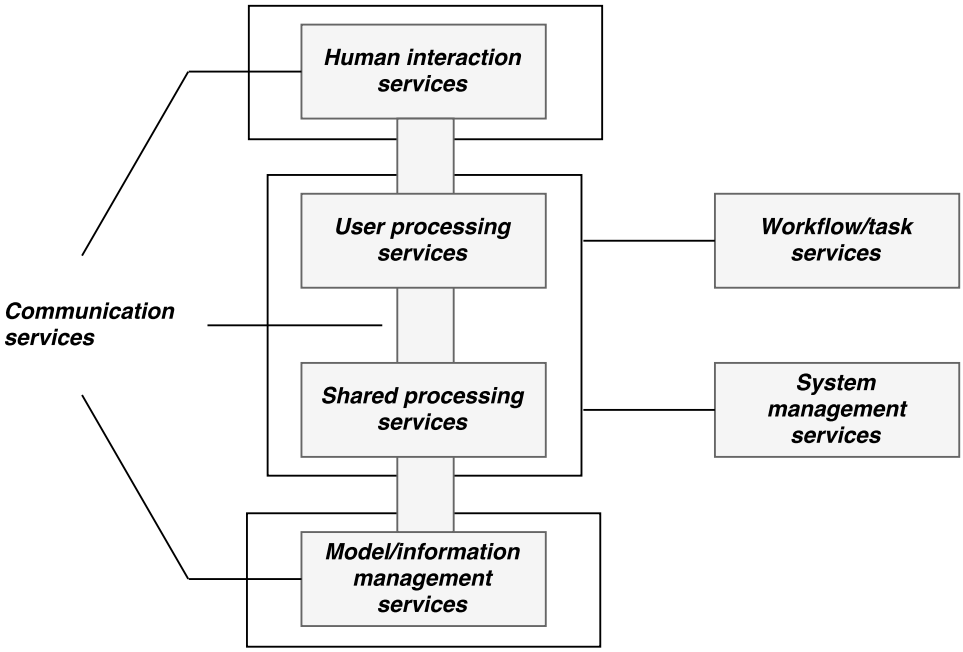


Figure 6.2: Web-based GIS architecture according to [Per02]

Figure 6.2 shows the logical architecture presented in the engineering viewpoint of [Per02]. It is a 4-tier architecture that categorizes the services into seven categories. Four of the categories are the four tiers of the architecture: *human interaction* services, *user processing* services, *shared processing* services, *model/information management* services. The processing services are further divided into spatial, thematic, temporal, and metadata services. The other three categories are auxiliary services that are used to connect services of different tiers (i.e., communication services), to manage service composition (i.e., workflow/task services), or to provide cross-cutting functionality (e.g., system management services). In [Per02], the logical 4-tier architecture is also mapped to different physical architectures (e.g., 2-tier, 3-tier with a thin client and 3-tier with a thick client). Furthermore, a large collection of example services are given for each category: 14 human interaction services, 18 spatial processing services, 16 thematic processing services, 4 temporal processing services, 2 metadata processing services, 12 model/information management services, 6 communication services, and 3 workflow/task services. No geographic-specific system management services are described, even though some general services like authorization and authentication are identified.

6.3 Analysis of architectures of existing products

Even when we follow a reference architecture, the expertise of the GIS experts from Enxenio can be used to enhance the PLA. To achieve, in this step, *analysis of architectures of existing products* (see the step 2.2, Figure 6.1), we consider a set of products developed by Enxenio and we study their features as well as the decisions made when they were designed to improve the reference architecture for our particular product line. But before, we explain the evolution of the architectures of geographic information systems to create the context for the decisions made.

GIS architectures have evolved since their first appearance. First generation of GIS tools were proprietary systems, designed to use specific data structures that store the information in files with proprietary format (Figure 6.3). This architecture is not used any more due to its numerous problems, such as using a proprietary geographic data formats, so interoperability with other GIS was not easy and the integration with other sources have to be done within the GIS application in an *ad-hoc* manner.

The second generation of GIS tools is consisted of systems that were integrated with Database Management Systems (DBMS). In these tools we can distinguish between dual architecture tools and layered architecture tools. Dual architecture

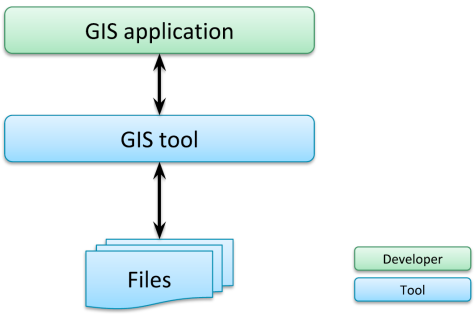


Figure 6.3: First generation architecture for GIS

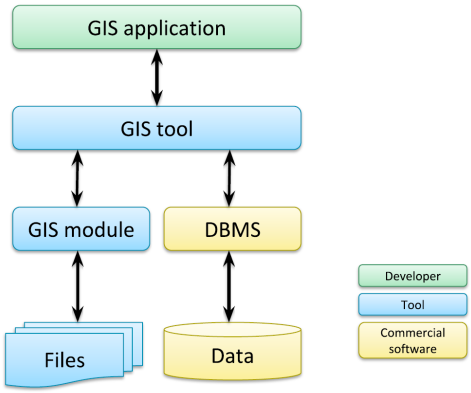


Figure 6.4: Second generation architecture for GIS: dual architecture

tools use an independent subsystem for each kind of information (Figure 6.4). This way, old software artefacts can still be used to handle some of the data types, and the access to each data type is very efficient. The big problem is to model, query, optimize and integrate all the information. Layered architecture tools (Figure 6.5) use a traditional DBMS to store all the geographic information, with the main advantage of using all the features from the DBMS. The problem in this case is the limited efficiency.

The third and current generation of GIS uses an extensible DBMS to implement the geographic features (Figure 6.6). DBMS used are Oracle, PostgreSQL and MySQL. The main advantage of this approach is that the implementation is integrated within the DBMS, so it is highly efficient. The disadvantage is that it requires a last generation DBMS and not every GIS feature can be implemented

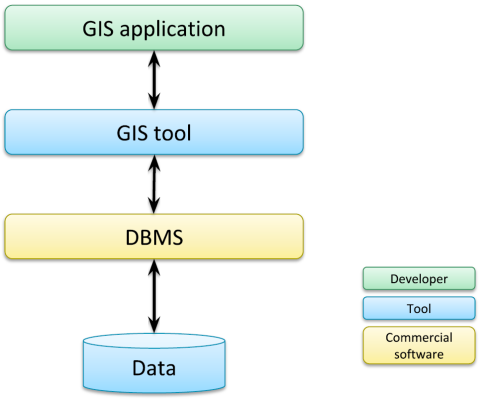


Figure 6.5: Second generation architecture for GIS: layered architecture

within the DBMS.

The three applications taken into account during this process (described in Section 5.3.1) follow this last kind of architecture, and so we do in our SPL as we can see in Section 6.6.

As something to highlight, from the analysis of these applications and taking into account the ideas extracted from conversations with experts from Enxenio, we have identified two characteristics that need to be added to the reference architecture:

i) The products must work both on desktop browsers and mobile browsers. Therefore, the client must be developed using responsive web design and a JavaScript web framework to ensure cross-browser support.

ii) Some clients require that the product conforms to the INSPIRE directive and/or international standards. The components of the architecture must follow these standards as closely as possible.

6.4 Elements selection/prioritization

Next step is the element selection (see the step 2.3, Figure 6.1). The GIS architecture from the standards (see Section 6.2) provide a very exhaustive set of services covering most of the possible features for any geographic information system. However, we do not pretend to maintain this kind of scope for our products, but we focus on products similar to the ones analysed in Section 5.3. Therefore, we need to identify the services that we require in our products. In Table 6.1 we can see

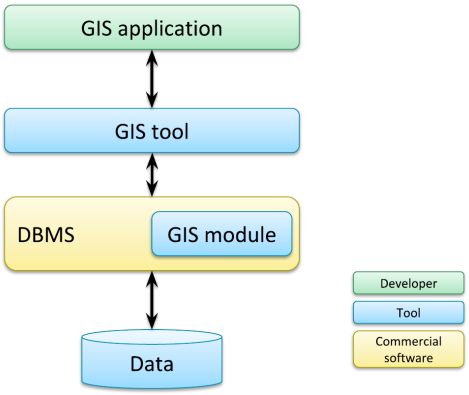


Figure 6.6: Third generation architecture for GIS

all the services identified in [Per02] as part of a GIS architecture, and the subset of components in which we have decided to focus.

Category	Service	Sel.
Geographic human interaction services	Catalogue viewer	
	Geographic viewer	X
	Geographic viewer - animation	
	Geographic viewer - mosaicing	X
	Geographic viewer - perspective	
	Geographic viewer - imagery	X
	Geographic spreadsheet viewer	
	Service editor	
	Chain definition editor	
	Workflow enactment manager	
	Geographic feature editor	X
	Geographic symbol editor	X
	Feature generalization editor	
	Geographic data-structure viewer	

Category	Service	Sel.
Geographic model/information management services	Feature access service	X
	Map access service	X
	Coverage access service	X
	Coverage access service - sensor	
	Sensor description service	
	Product access service	
	Feature type service	X
	Catalogue service	
	Registry service	
	Gazetteer service	X
	Order handling service	
	Standing order service	
Geographic processing services - spatial	Coordinate conversion service	
	Coordinate transformation service	
	Coverage/vector conversion service	
	Image coordinate conversion service	
	Rectification service	
	Orthorectification service	
	Sensor geometry model adjustment service	
	Image geometry model conversion service	
	Subsetting service	
	Sampling service	
	Tiling change service	
	Dimension measurement service	X
	Feature manipulation services	
	Feature matching service	
	Feature generalization service	
	Route determination service	X
	Positioning service	X
	Proximity analysis service	X

Category	Service	Sel.
Geographic processing services - thematic	Geoparameter calculation service	
	Thematic classification service	
	Feature generalization service	
	Subsetting service	
	Spatial counting service	
	Change detection services	
	Geographic information extraction services	
	Image processing service	
	Reduced resolution generation service	
	Image manipulation services	
	Image understanding services	
	Image synthesis services	
	Multi-band image manipulation	
	Object detection service	
	Geoparsing service	X
	Geocoding service	X
Geographic processing services - temporal	Temporal reference system transformation service	
	Subsetting service	
Geographic communication services	Encoding service	
	Transfer service	
	Geographic compression service	
	Geographic format conversion service	X
	Messaging service	
	Remote file and executable management	

Table 6.1: Services selection

6.5 Product Line Architecture Structure Building

After deciding which services we need to maintain in our product line, we start the fourth step of this stage, the *PLA structure building* (see the step 2.4, Figure 6.1).

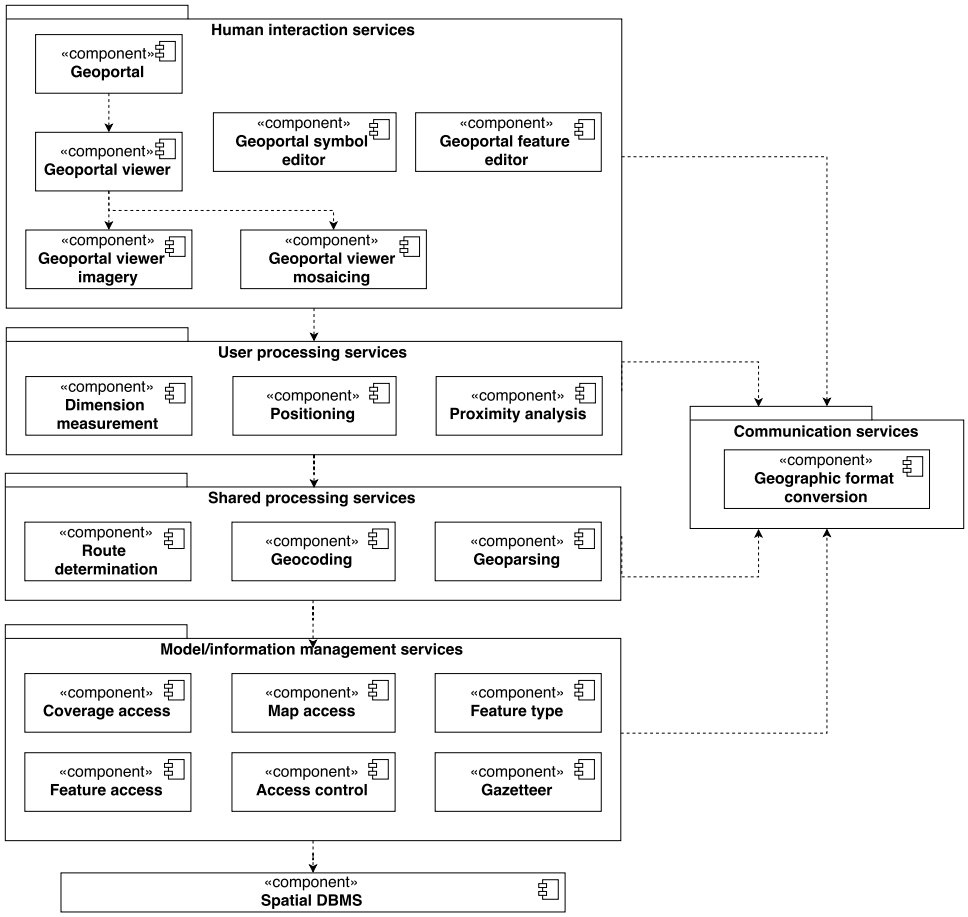


Figure 6.7: PLA Structure

The final product line architecture (PLA) of the SPL is shown in Figure 6.7. It provides all the services selected from the reference architecture, but also two other services added afterwards (namely Geoportal and Access control). The new services are motivated by inconsistencies discovered in the stage of Architecture Evaluation, as mentioned in Chapter 7.

We have maintained the same logical architecture from [Per02], and we have classified every new service within this architecture, showing also the relations among services.

- **Human Interaction Services**. Services for managing user interfaces, graphics, multimedia and presentation of composed documents. In our case, the web interface belongs to this group, as well as the possible documents that can be generated from the data of the applications, i.e., images of the maps.
  - *Geoportal*. As any web application, our products will have some features similar to the ones provided by any other management web application, such as listings, menus or pages providing static information. The geoportal service encompasses these kind of functionalities, and it serves as the main service through which the rest of the services can be accessed.
  - *Geographic Viewer*. The geographic viewer service allows a user to visualize geographic information within maps. It also provides features related to how this information is viewed. For example, panning and zooming the map, or some more complex operations such as managing the different feature collections shown.
  - *Geographic Symbol Editor*. This client service allows a human to select the symbol libraries, that is, the styles in which the geographic information will be rendered.
  - *Geographic Feature Editor*. Service that extends the geographic viewer allowing a user to interact with feature data. This is, the user is able to modify the geographic information of the geographic elements managed by the application. Geographic feature editor service is used in two ways: the first one allows the user to change an element already viewed in the geographic viewer; the second one is to change an element by triggering the geographic viewer from a alphanumeric element.
- **User Processing Services**. The set of processing services responsible for the functionality required by the user. This is, the processes that can be instantiated by an user in a given moment and whose result will only be provided to himself or herself in that moment.

  - *Dimension Measurement*. Service to compute dimensions of objects visible in an image or other geodata. In a geographic viewer, a user can measure a distance between two points or the area of a geographic element, for example.
  - *Positioning*. The positioning service shall be provided by a device that can obtain and unambiguously interpret the own device position information. A standard relevant to position services is ISO 19116 [fSf].
  - *Proximity Analysis*. Given a position or geographic feature, finds all objects with a given set of attributes that are located within an user-specified distance of the position or feature.

- **Shared Processing Services**. Processing services responsible for common services (both domain specific and general) usually required by many users. This is, processes that can be run by users but whose result is stored and shared among all the users. Also, this processes can be run in background after its initialization.

  - *Route Determination*. This service determines the optimal path between two specified points, taking into account the input parameters and a feature collection. It also can determine the measured distance between two points following a specific path, or the length of times it takes to follow the mentioned path.
  - *Geocoding*. Service to augment location-based text references with geographic coordinates (or some other spatial reference). This is, from a text document which contains geopolitical named entities or addresses, this service is able to identified the related geographic coordinates.
  - *Geoparsing*. Complementary to the geocoding service, this service scans text documents for location-based references such as geopolitical named entities or addresses.

- **Model/Information Management Services**. Set of services responsible for physical data storage and data management.

  - *Coverage Access*. Service that provides a client access to and management of a coverage store. An access service may include a query that filters the data returned to the client. ISO 19123 and ISO 19111 are relevant to coverage access.
  - *Map Access*. Service that provides a client access to geographic graphics, i.e., pictures of geographic data. ISO 19128 [fSj] is relevant to map access.

  - *Feature Type*. Feature type service provides a client access and management of features type definitions. The static and dynamic information models for a feature type catalogue are provided in ISO 19110 [fSb].
  - *Feature Access*. Service providing a client access and management of a feature store. This is, it allows the user to access the geographic and alphanumeric data related to a set of features. ISO 19125-1 [fSi], ISO 19107 [fSa] and ISO 19111 [fSc] are relevant to feature access.
  - *Access Control*. Every aspect related to the authentication and user management, which is common to many other web applications, is handled by access control services.
  - *Gazetteer*. Gazetter services provide access to a directory of geolocated instances of real world, therefore facilitating processes such as the provided by geocoding services. An information model for a gazetteer is provided by ISO 19112 [fSd].

- **Communication Services**. Responsible for connecting the different tiers of services.

  - *Geographic Format Conversion*. This service provides conversion features for geographic data. Usually it should cover most of standards formats used to share geographical information. However, some non-standard formats which are very popular can also be useful, such as shapefile.

- **Spatial DBMS**. The software artefacts in charge of storing the geographic data. These services also provide some low-level processing feature with high efficiency. For example, some of the services for geographic format conversion can be run within a DBMS.

6.6 Technology analysis: identifying state of the art technologies

After defining the PLA for our product line, we still have a last step pending, the *technology analysis* (see the step 2.5, Figure 6.1). Regarding technologies, we have four different types of technologies and tools in function of the part of the architecture where they are used: *DBMS*, *map server*, *data server* and *web client*. The selection of the technologies for the DBMS and the map server is already in the

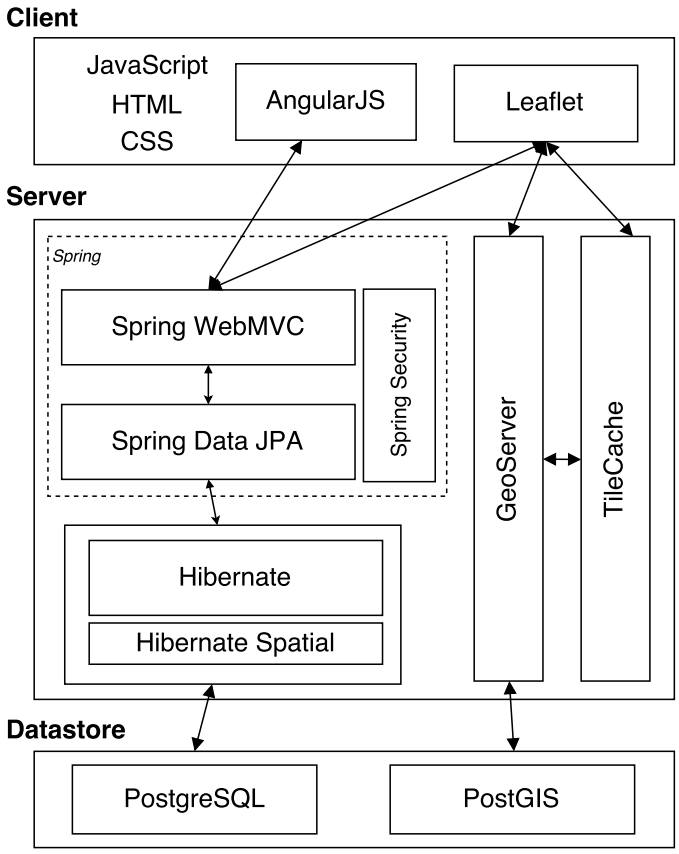


Figure 6.8: Technological architecture of the products

requirements of our SPL, so it belongs to the selection of features from the feature model for each product. This is, some of the technologies depend directly of the requirements of the SPL, and they can even vary in function of different products. Regarding the data server and web client, we try to use similar technologies to the ones applied in Enxenio, so the products can be used indistinguishably from the products manually developed in the company. Still, since a SPL is thought for the future, we choose the most updated but still stable variants of the frameworks and libraries. The technological architecture of our products can be seen in Figure 6.8.

As we have already explained in Section 4.4, there are three main variants regarding the software used as *database management system*: *PostgreSQL*[2] and *PostGIS*[3]; *MySQL*[4]; and *Oracle Spatial*[5] (we discarded SQLite since being portable is not desired feature for our products). Nevertheless, as mentioned in Section 5.3.2, the initial version of the SPL only implements one of the variants for it, specifically PostGIS, since the requirements validation reflected that the priority of the rest of them is very low.

From the requirements (Chapter 5) we have two variants to use as *map server*: *GeoServer*[6] and *Deegree*[7]. Both of the alternatives are Open Source, community-driven and comply to the standards, providing most of the common services such as WMC, WMS or WFS. GeoServer is the oldest of the two, and also the most-known. It is also the alternative used in Enxenio, so the initial version of the SPL uses it.

Regarding the *data server*, Enxenio is expert in Java technologies, which are also used in most of the applications developed by the company. Therefore, we use Spring and its set of libraries (such as Spring MVC, Spring Security, etc.) because it is the most known alternative for Java Data Server.

In the side of the *web client*, current web applications tend to follow the *single page application* (SPA) pattern. A single page application [MP13] is a web application that tries to replicate the behaviour and feeling of a desktop application, providing dynamic or real time interaction and feedback to the user. Traditionally, a web application is composed by HTML pages which are linked trough hyperlinks. Each time a link is clicked, the browser asks the server for a new page, and the server returns the complete page again, including the related style sheets and JavaScript libraries. Even when some of these resources will be stored in the cache of the browser, the page has to totally refresh and to show a new view. The objective of the SPA pattern is to avoid this refreshing because it only updates the part of

[2] https://www.postgresql.org/
[3] http://postgis.net/
[4] https://www.mysql.com/
[5] https://www.oracle.com/database/spatial/index.html
[6] http://geoserver.org/
[7] http://www.deegree.org/

the web view which needs to be updated. To achieve this, a SPA does not ask the server for every new view with GET HTTP petitions, but the JavaScript library related to the application is the one handling the rendering of the different sections. With every link clicked, an event is triggered and handled by JavaScript, and the HTML view is rendered directly within the browser. In these applications, the data transmission is usually made using REST services.

In current web application development, there are mainly two alternatives to build single page applications: *React*[8] and *Angular*[9]. React is a simple visualization library that allows the developer to build the interface from JavaScript with some specific components that represent every HTML component. Then, these components are built within a virtual DOM, and every time this DOM is stable, the virtual DOM is converted to real HTML. This way, only components from the DOM whose HTML representation has changed are rendered.

Angular, on the other way, is a complete framework that includes several utilities and libraries including everything needed to build a whole application just using the framework. This is, React only handles the rendering of the view, requiring external libraries to provide some other features needed for web applications, but Angular handles most of what is required to build a web application. Angular has two well differentiated versions: AngularJS, or Angular 1.x, and Angular, or Angular 2.x and upper. We have decided to use AngularJS, which has become a *de facto* standard library for web applications development.

For the web map viewer, most current GIS with web map viewers use *OpenLayers*[10] or *Leaflet*[11] as the map viewer library. OpenLayers is the most complete open source library for map visualization on web. It provides most of the features that a map viewer need, and it can be extended with plugins. However, its usage is pretty complex and requires a deep understanding of the library, as well as studying its documentation deeply, and usually most of its features are not needed. Leaflet is another map visualization library that provides basic features but a much simpler library to develop with. Also, Leaflet provides a simple API based on web standards to extend its features in any way needed. Its usage is more focused on mobile devices. For these reasons we choose Leaflet, specially because it is more lightweight and mobile-friendly, given that one of our characteristics is to build products that work both on desktop and mobile devices.

[8] https://facebook.github.io/react/
[9] https://angularjs.org/
[10] https://openlayers.org/
[11] http://leafletjs.com/

6.7 Summary

In this section we have described the different tasks regarding the second stage of our process, the architecture design. First we have chosen to base the architecture of our products in a reference architecture defined by a GIS standard, enhanced by some traits extracted from architectures of existing products developed by Enxenio. These two tasks are the two first steps of this stage: reference architectures identification and selection, and analysis of products architectures.

Once we have identified every component and service in our architecture, in the next step, elements selection and prioritization, we carried out a prioritization process to decide which features should have higher importance and therefore should be implemented first. Finally we have designed our full architecture in the step PLA structure building. The last step is development: technology analysis, in which the experts from the company Enxenio determined which is the technology that the implementation of our products must be based on.

7

Architecture Evaluation and Derivation of the products

7.1 Introduction

So far we have defined the feature model of our product line, as well as the product line architecture that the generated products need to follow. These two things should be related to each other, since the architecture should support and implement the features belonging the feature model, and the same way, each feature should be linked to a service or component within the PLA. In this chapter we focus precisely on this topic, analysing both things together to determine if the coherence in both parts is maintained. After that, there are no more steps in the context of the *domain engineering* and we need to focus on the actual generation of the products.

The last stages of our methodology, as seen in Figure 7.1, are the *architecture evaluation* and the *derivation of a specific product*. This chapter describes both of them. To evaluate the chosen architecture first we need to establish the relationship or traceability between the features identified in the *domain analysis* stage and the components or services defined in the *architecture design* stage. Then we can finally evaluate the product line architecture as a whole. The derivation of a specific product is described in the second section of this chapter. The derivation process is

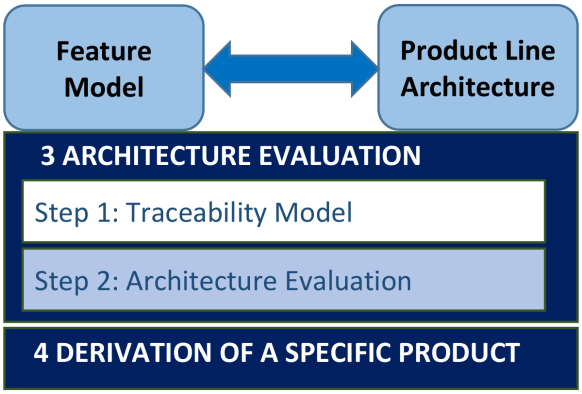


Figure 7.1: Architecture evaluation and derivation of a specific product stages

the actual generation of a product of the product line from a subset of features. In this stage we leave the context of the *domain engineering* to enter in the *application engineering*.

7.2 Architecture Evaluation: maintaining the traceability

Maintaining the traceability between the different layers of a SPL is important if we want to be able to evolve adequately the platform and the products when the requirements, components or technology changes. For example, if we need to add a new requirement to the platform, keeping the traceability allows to easily identify possible features related to this new requirement. These features can be matched with the set of components or services implementing them and therefore with the specific technologies used to implement them. This way we know all the elements involved in adding this new requirement.

In Section 5.2 we have already shown the traceability model between requirements and features as part of the *domain analysis* stage. In this section we complete this traceability model by linking features to components from the functional architecture. We show the results in Table 7.1.

Feature	**Service**
DM_SpatialDatabase DM-SD_PostGIS DM-SD_MySQL DM-SD_OracleSpatial	Spatial DBMS
DM_DataServer	Feature access
DM-DI-DF_Shapefile DM-DI-DF_Raster DM-DI-DF_Network	Geographic format conversion
DM-DI-D_Form DM-DI-D_Map	Geographic feature editor, Feature Type
DM-A_Connectivity DM-A-C_RouteCalculation DM-A-C_NetworkTracing DM-A-C_ConectivityCheck	Route determination
DM-A-G_Addresses	Gazetteer, Gazetteer
DM-A-G_Documents	Geocoding, Gazetteer
GUI-M_Top GUI-M_Bottom GUI-M_Right GUI-M_Left	Geoportal
GUI_Forms GUI-F_Editable GUI-F_Creatable GUI-F_Removable GUI-F-R_ConfirmationAlert	Geographic feature editor, Feature Type

Feature	**Service**
GUI_Lists GUI-L_Sortable GUI-L_Filterable GUI-L-F_RowFilter GUI-L-F_BasicSearch GUI-L_LocateInMap GUI-L_ViewListAsMap GUI-L_FormLink GUI_StaticPages GUI-SP_Management	Geoportal
MapViewer	Geographic viewer
MV_MapServer MV-MS_GeoServer MV-MS_Deegree	Map access
MV-T_Pan MV-T_Zoom MV-T_ZoomWindow	Geographic viewer
MV-T-M_Distance MV-T-M_Line MV-T-M_Polygon MV-T-M_MapElement	Dimension measurement
MV-T-E_SetScale MV-T-E_ShowLegend MV-T-E_DRM MV-T-E-F_PNG MV-T-E-F_PDF MV-T-E-F_URL	Geographic format conversion
MV-T_Filterable MV-T-F_RowFilter MV-T-F_BasicSearch	Feature access, Feature Type
MV-T_UserGeolocation	Positioning, Proximity analysis

Feature	**Service**
MV-T_InformationMode	Geographic viewer
MV-T_ViewMapAsList	Geoportal
MV_ContextInformation MV-CI_Map MV-CI_Scale MV-CI_CenterCoordinates MV-CI_Dimensions MV-LM_Order MV-LM_HideLayer MV-LM_Opacity	Geographic viewer
MV-LM_Style	Geographic symbol editor
MV-LM_ExternalLayer MV-LM_Clustering MV-MC_BBox MV-MC_UserPosition	Geographic viewer
MV_DetailOnClick	Feature access

Feature	**Service**
UserManagement	
UM_Registration	
UM-R_ByAdmin	
UM-R_Anonymous	
UM_Authentication	
UM-A_RememberPass	
UM-A_RecoverPass	
UM_AccountActivation	
UM-AA_ByEmail	Access control
UM-AA_ByAdmin	
UM_UpdateEmail	
UM-UP_ByUser	
UM-UP_ByAdmin	
UM_UserProfile	
UM_UserCRUD	
UM-ST_Session	
UM-ST_JWT	
UM_LDAP	

Table 7.1: Traceability feature - service

Although we started from a reference architecture (see Section 6.2), in the first iteration of this step we found that many of the features of the SPL platform were not provided for any of the services of our PLA structure, such as *Access control*. This reflects the importance of following a serious methodology and even more, the importance of the step of validation within it. When we found the inconsistencies described, we started a new iteration of the process, solving these problems in the first stages of the methodology with the feedback provided in this step by adding the required components (see Chapter 3). We considered the end of this iterative process when we do not found inconsistencies between the features and the set of components providing them.

7.3 Derivation process in our Software Product Line

The derivation process is the actual generation of a product from a SPL. This step of our methodology serves to analyse the particulars for the derivation of products within our product line.

In most SPL the analyst simply selects a set of features and then the derivation engine assembles the required components into the final source code. In the case of our SPL, given that each web-based GIS is built for a specific data domain, the selection of features is not enough to build the products. Hence, in the first step of the derivation process for the products of our software product line, the analyst needs to provide the data model definition for the product. This way, the code generated can adjust to the specific data model for the product. There is already another tool that creates web-based GIS applications from a data model, GENGHIS [DMG13], but, unlike our SPL, the data model is the only variability of the products generated by GENGHIS, being every product exactly the same application with a different data model.

The actual feature selection is the second step of our derivation process. The analyst selects a set of features from the feature model, and this selection must be validated within the derivation process (using operations defined in [BSRC10], for example), so the analyst receives feedback in case he or she makes a mistake in the selection.

At this point, the analyst has chosen the entities that the application will manage, and the set of features that the application will provide. However, not even this is enough for products of our domain: just deciding to have the feature "map viewer" does not provide information regarding how many map viewers the product requires or which layers should be shown in each map viewer. The analyst needs to answer these questions among others, and some of them need to be answered for each one of the map viewers of the product:

1) Which base layer uses the map viewer?

2) Which entity or entities are shown in the map viewer?

3) Can the user make a textual search over the elements shown in the map viewer?

4) Can the user click on the elements shown in the map viewer? What happens if he or she clicks?

5) Can the user update the geographic data related to a specific element?

6) ...

All these options do not belong to the feature model but are something specifically related to each one of the products. Thus, they belong to the application engineering domain and our derivation process needs to provide a way to define all these aspects. Moreover, there are more features susceptible of this kind of specialization. For example, the set of forms that the final product will have, the menu entries that we want to show (instead of just showing everything) or the sorting order of these entries.

To summarize, our domain requires a further step beyond the selection of features or the definition of the data model. We have called this step *parametrization* of the product. It is important to note that this step does take into account both the data model and the feature model defined previously, since the parameters or "questions to ask" depend on which features are selected or which entities are defined. For example, to configure a map to visualize the cars managed by the application, we first need to include an entity "Car" within our data model.

So far, we have described the requirements of the process regarding the "inputs" required, but we also need to point out some requirements about how this derivation is produced and how should be the code produced by it. These requirements are the result of discussion with project managers from Enxenio, and they reflect some requirements that the product must comply with in order to integrate the SPL within their organization:

- *Static binding [ABKS13].* The product is the generated code itself, so it is important that the implementation technique allows derivation in the earliest stage, before the compilation step, to get the product variant source code.
- *Clean output code.* After generating each product, the code may evolve independently from the platform because developers may modify it with custom features not belonging to the platform. To allow this, the final code must be *maintainable* and as *well-written* as possible, and must not depend on anything beyond the own product (e.g., implementing variability with *aspects* makes the product dependant of them).
- *Allow different programming languages.* The implementation of web applications, nowadays, involves lots of different programming languages. Just in the client side, HTML, JavaScript and CSS are the main ones, but there are many frameworks that do not work directly with these but uses many other languages to generate them, such as template engines like Jade[1], JavaScript

[1] http://jadelang.net/

derived languages like Dart[2] or TypeScript[3], or stylesheets preprocessors like Sass[4]. In the server side, there are many choices to implement the features depending on the ecosystem: Java for Spring, JavaScript (or any variant) for Node, PHP, Ruby, etc. Therefore, our derivation engine has to handle different programming languages.

- *IDE independent.* The technique used cannot impose any IDE or tool restriction to the developers of the source code. Many of the existing tools [ADT07, MSC+14, MTS+14] share the same problem: they depend on an outdated IDE. We want to avoid this dependency, making our derivation engine independent of any other software as much as possible.

- *Derivation independent of the design of the products.* This is, the products should be very close to the same products if they were manually developed; "sacrifices" in the code just to fit the derivation engine shall be minimum or null.

7.4 Summary

In this section we have described the last two steps in our methodology: the architecture evaluation and the derivation of a specific product.

In order to do a proper architecture evaluation, we have gone through each one of the different components and services identified for our architecture and we have joined them with features extracted from our set of requirements. Every time there were miss-matches between both elements, it was a time for a new iteration in our process, adjusting both the feature model and the product line architecture until both things were totally coherent with each other.

Afterwards, once the PLA is finished and the traceability is maintained, the SPL is ready to be used to generate products, and we have proposed a set of requirements for the derivation process. These requirements are decisive for the evolution of the platform, and the derivation engine used to generate the products for this line must comply them.

[2] https://www.dartlang.org/
[3] https://www.typescriptlang.org/
[4] http://sass-lang.com/

Part III

GISBuilder: a tool for the semi-automatic generation of web-based GIS applications

8

SPL implementation techniques & Scaffolding: state of the art

8.1 Introduction

In the previous part, we have fully detailed the definition of our software product line for web-based geographic information systems. We have identified the requirements of the products, we have designed an architecture for them, and finally we have described how the derivation of these products must be done. Once this definition is finished, the next step is its actual implementation. In SPLE, the actual implementation is usually a very simple process (or at least a non-interesting process) in which developers must code the different features. However, we defined some very specific and advanced requirements for the derivation of each product (see Section 7.3) that are not contemplated by any state of the art SPLE implementation technique. In this part we show the particular implementation of our SPL (see Chapter 9), but first, we describe the state of the art regarding SPLE implementation techniques and their issues with the requirements of our derivation process, as well as basic concepts about *scaffolding*, that will help the reader to better understand the importance of our solution.

8.2 SPLE: Approaches and tools

SPL implementation techniques can be classified into two approaches: compositional [ABKS13] and annotative [KAK08]. The compositional approach requires having the code for each feature separated from the code of the rest of the features until the product is built. At this point, all the code for the selected features is assembled together by the derivation engine. The annotative approach is just the opposite: the code for all the features is together and the developers use annotations to delimit the code for each one. During the derivation process, the derivation engine removes the code for the features not selected.

Apel et al. [ABKS13] presented an extensive list of tools for implementing SPL that can be complemented with [ADT07, MSC+14, MTS+14, Dat]. We do not describe each of these tools and techniques in this work, but we describe the issues we have found in the most popular for their usage in our context (see Section 7.3). We classify the tools by their approach. The code examples are from a simple web calculator [BCLP16] implemented using several techniques to illustrate their problems.

8.2.1 Compositional or positive approach

The compositional approach implements features as distinct modules. That is, the particular code affecting each feature is separated from the code of the rest of the features. At the moment of the derivation of a product, the modules of the selected features are composed into the final source code. The main advantage of this approach is the clear separation of the code of the different features and that the traceability between the code and the features is explicit. Besides, having the code organized in features makes the maintenance of this code easier. However, there are some big problems with this approach [ABKS13]:

- Tools have to be constructed for every language. There have been some attempts of tools supporting more than one language, but they require anyway extra plugins or enhancements to support them.
- Lack of tooling support: most tools are academic and outdated, and they do not support composition on the set of programming languages used nowadays in a web application.
- The development team needs to adopt a language extension, which may be different for every language used in their developments, and they need to learn how to use unfamiliar composition tools as part of the development process.

- Composition mechanisms are usually limited regarding to where the variability can be applied. For example, variability in Java can usually only be applied at the level of "methods", and there is no support for variability on other points, such as annotations, classes or method parameters.

There are several tools following this approach, being the most used AHEAD and FeatureHouse, described next.

AHEAD

The first and main problem of AHEAD [BSR04] is that **generics are not supported**. Generics [Bra04] were added to Java in the version 5.0, and they have became one of the main artefacts to create good and extensible code strongly checked during compilation. An example is shown in Figure 8.6. Nowadays, we cannot imagine to create Java code without generics.

```
List<Integer> list = new ArrayList<Integer>();
```

Figure 8.1: Example of a generic in Java

AHEAD **does not support variability when defining enums**. An enum type is a special data type that enables for a variable to be a set of predefined constraints[1]. We can see an example of enum in Figure 8.2. Enums are very common in Java software development, but we cannot compose them with AHEAD. In fact, enums cannot even be defined at all on *jak* files, which is the format used by AHEAD.

```
public enum Operation {
    ADD,
    SUBTRACT,
    MULTIPLY,
    DIVIDE
}
```

Figure 8.2: Example of a enum type definition in Java

Java annotations are not fully supported. For example, class and method annotations can be used, but they must be repeated in all the features refining the same item in order to make the composed code to maintain the annotation. Besides, parameter annotations are not allowed, such as the one shown in Figure 8.3.

[1] https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

```
public ResponseEntity<ResponseJSON> calculate(@Valid @RequestBody RequestJSON
    request) {...}
```

Figure 8.3: Example of a parameter annotations in Java

To generate the output code, AHEAD converts the refinements in inheritance of abstract classes. If several features are implied in the same class, each refinement constitutes an abstract class except the last one. Hence, **the output code is hard to understand and to extend** with new functionalities, as we can see in Figure 8.4.

```
@RestController
@RequestMapping("/api/calculator")
abstract class CalcResource$$Base$...$controller {

    @RequestMapping(method = RequestMethod.POST)
    public ResponseEntity calculate(
        RequestJSON request) {

        return response entity;
    }
}

@RestController
@RequestMapping("/api/calculator")
public class CalcResource extends CalcResource$$Base$...$controller  {

    @RequestMapping(method = RequestMethod.POST)
    public ResponseEntity calculate(
        RequestJSON request) {

        if operation is division {...}

        return super.calculate(request);
    }
}
```

Figure 8.4: Example of code generated by AHEAD

The issues explained in this section affect not only to the AHEAD *composer*, but also to other tools included in the AHEAD Tools Suite, like *reform* or *jak2java*.

FeatureHouse

FeatureHouse [AKL09] shares some of the problems of the previous tool, AHEAD. For example, **annotations are not supported**. In this case, parameter annotations are partially supported, since only one annotation is allowed. Again, as in AHEAD, in order to use annotations in methods we need to repeat the annotation for every feature related to the particular method, so it is not possible to apply variability within the annotations. That is, even when we can use some annotations in our code, we cannot apply variability to them. So we cannot choose to use one or another annotation depending on the selected feature.

To generate code, FeatureHouse converts the refinements in different chained private methods. As in the previous case, **the generated code is hard to read and edit**, as we can see in Figure 8.5.

```
@RestController
@RequestMapping("/api/calculator")
public class CalculatorResource {

    @RequestMapping(method = RequestMethod.POST)
    private ResponseEntity<ResultJSON> calculate__wrappee__Base(@RequestBody
     RequestJSON request) {

        return response entity;
    }

    @RequestMapping(method = RequestMethod.POST)
    public ResponseEntity<ResultJSON> calculate(@RequestBody RequestJSON
     request) {

        if operation is division {...}

        return calculate__wrappee__Base(request);
    }
}
```

Figure 8.5: Example of code generated by FeatureHouse

XAK

We have not found any major limitation with XAK [ADT07] regarding the capabilities of the derivation of the products. The only minor issue found is that its scope is limited to well-formed XML files. In order to apply XAK to HTML5 files, which are not well-formed XML files, the developers would not be able to write the

following code, which is valid HTML5:

```
<div class="a-class" required/>
```

Figure 8.6: Example of HTML5 code

However, the big issue with XAK is the complexity to implement the refinements. Although this alternative is based on a compositional approach, in order to *refine* the different XML artefacts the developer also needs to annotate the XML files with determined tags, so the implementation effort is doubled since it is necessary to maintain both a compositional structure and the annotations.

8.2.2 Annotative or negative approach

The annotative approach consists on having the code for all the features together and using some annotations with a particular syntax to delimit the code related to each feature. This way, in the moment of derivation of a product, the engine does not assemble or compose the features together but it does just the opposite, removing all the code related to the features that have not been selected.

The main advantage of using annotations is that the code produced by the SPL can be as good as the annotated source code is, without any added complexity imposed by the technique. That is, in the compositional tools we have reviewed, such as AHEAD or FeatureHouse, it does not matter how nice the code that implements the features is, because the derivation process will mess the final source code. In the annotative approach, the derivation process is not as intrusive and the quality of the annotated code is maintained in the final code.

However, in this case it is the code of the platform the one that is hard to read and maintain, specially if the annotations are undisciplined and complex or the variability is implemented with fine-grained extensions [KAK08]. Fine-grained extensions, or fine granularity variability, are points of variability that are implemented at the level of lines of code. That is, in the compositional approach one of the restrictions is that the developer can usually only implement variability at the *method* level, modifying the behaviour of a method depending on the selected features. In the case of the annotative approach, we do not have this restriction and variability can be applied at any level to the point we can even change a line of code depending on the selected features. This is a big advantage, but abusing this kind of annotations only leads to the *#ifdef hell* [SC92]. Traceability is also hard to maintain beyond the file-level, since the annotations take place at the different files, and this approach has no notion of modularity [ABKS13].

Preprocessors are the most popular alternative to implement a SPL following the annotative approach.

Preprocessors: CPP, GPP and GNU M4

CPP[2] is one of the most common alternatives used by the industry [GLA+09, JB09, PO97, SLB+10, TSSPL09, BRN+13] to implement SPL, even when there are many works criticising CPP annotations [SC92, LST+06, SLSA13]. Annotated code is prone to simple errors and makes type-checking and debugging process an arduous task, even when the developers try to maintain discipline for their annotations [SLSA13]. However, recent research works are focusing on this approach, trying to analyse the use of annotations [HZS+15, LKA11, LAL+10] and to improve it [KAK08, FKA+13, KATS12].

One known downside when working with CPP is that only a few programming languages (C and variants, mostly) support these annotations. So, working on Java code annotated with CPP, for example, is tricky since the common code editors will not be able to check for syntax and grammar compliance. The same happens with JavaScript or CSS code annotated with CPP or any other kind of code not related to the C family. There are code editors like FeatureCommander [FPK+11, FKA+13] that facilitate working with CPP annotated code by colouring the different features, but still do not help solving this kind of issues.

GPP[3] and GNU M4[4] are general preprocessors. That is, they are independent of any programming language and mostly they provide the same functionality than CPP. Both of them allow custom delimiters so the annotated code does not interfere with concrete language IDEs or tools.

The downside of these alternatives is the difficulty of using them to implement a SPL, specially for our context, web-based GIS, because:

1. These alternatives are old software assets. CPP is the only that can still being installed and configured "easily", but making GPP or GNU M4 work is really hard even in Unix-based systems. In any case, all of these tools are strange to the current development of web applications, and they cannot be easily integrated with the tools that are being used nowadays.

2. CPP annotations interfere with any IDE or compiler of any language code that is not based on C. GPP and M4 allow for the definition of customs annotations, but doing it requires a hard effort.

[2] https://gcc.gnu.org/onlinedocs/cpp/
[3] http://en.nothingisreal.com/wiki/GPP
[4] http://www.gnu.org/software/m4/m4.html

3. They are not designed to be used to implement SPL. They all lack of any variability managing beyond the simple annotations, and of course they do not validate the specifications of the products.

Antenna and Munge

Antenna[5] and Munge[6] are tools that enable conditional compilation on Java. That is, they are similar to CPP but for Java. In fact, the annotations mimics the CPP syntax, but they are written in Java comments to not interfere with code editors. Besides, there are plugins for Eclipse to facilitate the use of them.

The big difference between the two alternatives is that Antenna comments out code annotated with features not selected, so the compiler omits it, while Munge acts like a proper preprocessor and the annotated code is processed to obtain code without annotations and ready to compile.

FeatureJS

FeatureJS [MSC+14] is the only tool we have found that is focused on the JavaScript language, one of the most popular languages nowadays. We agree with the analysis made by Machado et al. in [MSC+14] regarding the state of SPL researching and tools for web-based applications: the domain of web systems is not often considered as a potential domain for developing SPLs, unlike other domains such as embedded systems or mobile applications, and most of the literature proposing new research initiatives handling SPL for web systems [BRPD05, TBD07, PJ05, LGBH09, GG02, FAB+13, CD03] are mostly concerned with modelling the variability at a high-level abstraction but without doing the same at the implementation level.

FeatureJS mixes composition and annotations, but, as far as it is detailed on [MSC+14], it does not provide any way of refining artefacts, so the compositional feature is equivalent to annotating the whole JavaScript code file. Anyway, refining and composing JavaScript code does not seem feasible due to its own characteristics and its flexibility, as they also remark.

CIDE

CIDE [KAK08] is an eclipse-based IDE which annotates the features directly on the Abstract Syntax Tree and shows the different features with different colours. CIDE appeared as a way to improve the legibility of the annotated source code to overcome some of the issues of fine granularity variability.

[5] http://antenna.sourceforge.net/
[6] http://weblogs.java.net/blog/tball/archive/2006/09/munge_swings_se.html

Using CIDE imposes an obvious IDE restriction, since it is based on Eclipse. Moreover, the last version available is built for Eclipse Galileo, from 2009, which does not support Java 8 syntax, among other things. Moreover, the community support for this version of Eclipse is minimum.

8.2.3 Alternatives using other approaches

Besides the two main approaches for the implementation of SPL, there are some other alternatives that were not really designed to be a SPL implementation technique but that can be used that way.

AspectJ[7] is an aspect-oriented programming (AOP) extension for Java. It can be used through an Eclipse plugin. It has become a *de facto* standard for AOP due its popularity. **DeltaJ**[8] is a Java-like new language which allows to organize classes into modules, so then the code can be specified by adding, modifying or removing delta modules. The big issue of both these alternatives is that they deeply change the way the projects are implemented, forcing the development teams to change their procedures and even to get formation on new techniques that they do not require.

Another alternative to create *ad hoc* SPLs is through *version control systems* (VCS) since they highlight as evolution managing systems for both the platform and the variant products when applying methodologies like [MD15]. However, we see VCS more as a complement to handle evolution, through a similar methodology, than a technique to implement the SPL.

8.2.4 Summary

To summarize, we have described several tools of the main approaches followed to implement SPL and we have discussed the main problems against using them for our SPL.

For compositional tools, the main issue is that the output source code is objectively bad. This happens with all the alternatives tested to a greater or lesser extent, and we show some examples above. Besides, there are not compositional tools for all the languages taking part on the development of a web application with state-of-the-art technologies.

The annotative tools and, more concrete, the general purpose preprocessors (GPP), seem to be more adequate to our context even with their problems, very present in the literature. The code generated with these tools is clean, they

[7] https://eclipse.org/aspectj/
[8] http://deltaj.sourceforge.net/

allow templates of any programming languages and the generated products are independent of the products. However, even when they are not strictly dependent on any IDE, installing and using these alternatives is very complicated. Nowadays, it is maintained by one person who does not provide a binary installer for non-Unix based operating systems, so we need to compile it in our own (or look for non official sources) if we want to make it available for every operating system. Furthermore, this software cannot be easily integrated with the set of tools that are currently used for development of web applications. Finally, GPP does not provide functionalities for managing the variability or validating it, which is quite important.

Regarding the rest of the approaches, like aspect-oriented programming, our requirement for making the product code totally independent of the SPL technique and our need to support several languages at the same time disable the use of this kind of techniques.

Some of the issues mentioned could be overcome with some extensions or enhancements over them, but the conceptual problems behind these issues are harder to solve. Besides, instead of going through the effort of modifying of these tools, we consider that it is more profitable to design and develop a new tool from scratch that uses current technology and that complies all our requirements.

8.3 Scaffolding: industrial generation of code

In the last years, new techniques and methodologies have emerged in software development industry to speed up and to improve the development processes. Scaffolding is one of these new techniques, mainly popularized by Ruby on Rails[9] from 2005, but currently used by many trending software development frameworks, such as Yeoman[10] or Spring Roo[11]. Scaffolding consists on generating code from predefined templates and some specification provided by the developer. This way, most of the repetitive and generic code of the applications is automatically written, and the developer can start its work from a mid-stage or half-built architecture. Scaffolding is not limited to any context, and it can be used to generate any kind of code or text, from code related to the user interface of an application to its documentation.

Although the scaffolding concept applied to software development is relatively new, in fourth generation programming languages (4GL), fashionable in the 1980s, there was a similar feature in database code generators, such as Oracles CASE Generator or dBase IV APPSGEN tool, which were able to generate forms directly

[9] http://rubyonrails.org/
[10] http://yeoman.io/
[11] http://projects.spring.io/spring-roo/

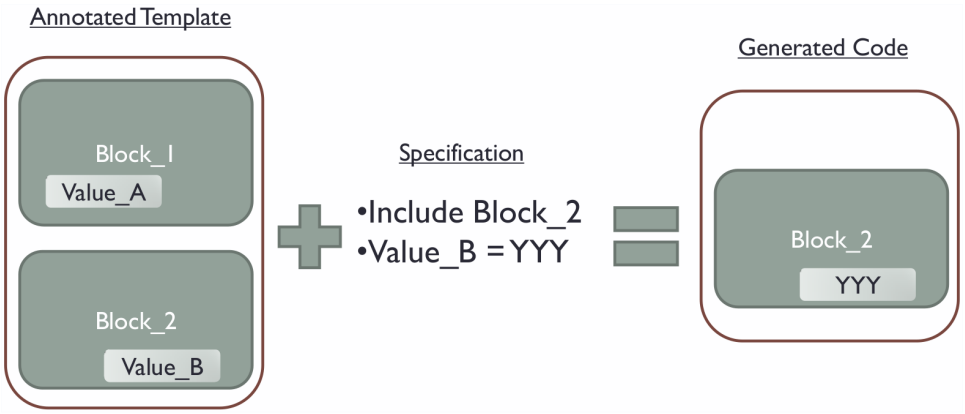


Figure 8.7: Scaffolding example

from the data model of the database. Likewise, we can see *scaffolding* as an informal application of some concepts from Model Driven Development. MDD combines domain-specific modelling languages, which formalize the application structure, behaviour, and requirements within particular domains, with transformation engines and generators that analyse certain aspects of models and then synthesize various types of artefacts [BCW12]. This is, from a set of models defined for an application, and through a series of transformations, some code is generated specifically for the product. However, this code is usually not replicated in any other product unless the models are the same. Therefore, MDD is useful to reduce the cost of developing a single product, but it is not as useful in the case of families of similar products and it does not provide for any mechanism to explicitly handles variability. We have also found that the tooling support for MDD is mostly academic and most tools are not ready to be used in industry, which also explains the fact that MDD application in industry is not common [BCW12].

In Figure 8.7 a conceptual diagram of how scaffolding works is shown. In the left part, we can see an annotated template with two blocks, identified by "Block_1" and "Block_2". Each one of this blocks have themselves a variable, identified as "Value_A" and "Value_B". If the developer provides the specification shown in the middle of the figure, he or she is configuring the code to include only the second block and it provides a value for one of the variables. Once the scaffolding process is done, the resulting code is exactly the demanded one.

8.3.1 Scaffolding vs SPLE

Even when they are not the same, we can make an informal comparison, at a practical level, between scaffolding and SPL. There are four different points in which there are conceptual differences:

- **Product specification**. In a SPL, the specification of a product is something as simple as the selection of a subset of the features provided by the product line. We can have an idea of how simple it is by thinking in its representation, which would be a list of strings. As opposite, the specification in a scaffolding-based tool can be as complex as required by the predefined templates. For example, Spring Roo[12] allows the analyst to define the data model for the domain of an application (classes and properties), and the source code of a web application with five layers (view, controller, service, DAO[13] and model) is automatically generated.

- **Code generation**. The way a final product is assembled within a SPL is by adding or excluding the different assets. This is, the code implementing an asset is already created exactly as it will be used in the final product, and it can be added to the final product or not. In the case of a scaffolding tool, going along with our previous example of a Java class, there is no way of having the code for it prior to the definition of it by the analyst. This is, the code for a class has to be *really generated* from the specification, not just taken and assembled together with the rest of the code.

- **Development stage for its application**. Usually, a developer uses a scaffolding tool in the first steps of the development of an application. Once the code is generated, he or she will continue with the development of the application and probably he or she will modify the generated code once the product evolves. Just the opposite is what occurs in the case of a SPL, when usually the generated code is really to be deployed and used by final users.

- **Maintainability**. As a derived effect from the previous point, the code generated by a SPL is not thought to be modified. Therefore, it is not important its quality, regarding maintainability, nor its extensibility. This is not the case for Scaffolding generated code, which needs to be well-formed and clear so the developer can understand and extend it properly.

[12] http://projects.spring.io/spring-roo/
[13] http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

8.3.2 Libraries and frameworks using scaffolding

The first framework using the term *scaffolding* was **Ruby on Rails**[14], a server-side web application framework written in Ruby[15]. The scaffolding concept was not the only thing that this framework popularized, but also the own Ruby language was not very known before Rails. It is based on the model-view-controller (MVC) pattern but it also follows other well-known patterns and paradigms such as *convention over configuration* [Che06], *don't repeat yourself* [WAB+14] and the active record pattern. Rails provide default structures for the database, web services and web pages, allowing the developer to focus on coding itself. With the same philosophy, Ruby on Rails uses scaffolding to automatically construct the models and views for a basic web application so the developer can skip these generic and repetitive tasks.

Grails[16] is a web application framework that uses **Groovy**[17]. Groovy is a dynamic language, optionally typed, with static-typing and static compilation capabilities, for the Java platform. Grails framework targets to increase the developer productivity by following the paradigm of *convention over configuration* with sensible defaults and opinionated APIs. Its goal is similar to the Rails one, allowing the developers to focus on coding and, in fact, they both appeared in the same year, 2005. Grails lets developers to use both dynamic and static scaffolding. With dynamic scaffolding, the models and views are created when the application is deployed, whereas with static scaffolding the code is generated when the developer requires it. It also allows the developer to customize the templates used by the scaffolding engine.

Another Java framework that benefits from the scaffolding technique is **Spring Roo**[18], a software developer tool that again follows the paradigm of *convention over configuration* to provide rapid application development. With Spring Roo, a developer only needs to run a series of terminal commands to have the full source code of a complete web application with the data model specified, with views and models and everything ready to be deployed. Some of the code related to Spring Roo projects is implemented with aspects. However, in any point of the development a project created with Spring Roo can be converted in an independent project and be treated as any other Java development.

So far we have described three different frameworks focused on more or less the same: improving the developers productivity by generating source code. **Yeoman**[19]

[14] http://rubyonrails.org/
[15] https://www.ruby-lang.org/en/
[16] https://grails.org/
[17] http://groovy-lang.org/
[18] http://projects.spring.io/spring-roo/
[19] http://yeoman.io/

is a totally different thing, a library that facilitates the implementation and usage of generator projects. This is, Yeoman is somehow a "scaffolding platform", a library that allows you to implement scaffolding tools in any context or domain, and to use them. While Yeoman itself does not create anything, there are currently over six thousand different public generators. **jHipster**[20] is one of the most known, and it can create the code of a complete web application using current technologies (Hibernate, Spring, Angular, Sass, etc.) and providing the typical set of features for web applications (authentication, lists, forms, metrics for the services, internationalization, etc.) in a matter of seconds.

8.4 Summary

In this chapter we have presented the most popular approaches for implementing a software product line: the compositional approach and the annotative approach. We have described the details regarding each approach and we listed the most used tools for both of them. For each of these tools, we have shown how they do not comply the particular requirements of our context, described in Section 7.3.

Afterwards we have introduced a technique called scaffolding, common in industry practice. Scaffolding is used nowadays to accelerate the development of web applications by generating parts of the code that are easily abstracted and reproduced. A tool based in this concept can satisfy all the requirements for our context, and it can be implemented and delivered with updated technologies to reduce the effort required to implement it on a real company.

[20] https://jhipster.github.io/

9 GISBuilder's design

9.1 Introduction

Once the main goal of this thesis, the definition of a software product line for the domain of web-based geographic information systems, is achieved, we need to use this definition to actually design a tool able to generate GIS applications. In the previous section we have already described the problems found in the existing tools to implement a SPL, and how a tool based on scaffolding can comply all of our requirements and be the most adequate solution in our context. This chapter describes the different aspects of the tool designed and implemented from the results of our previous process. This tool is called GISBuilder.

In the design of GISBuilder we have taken into account all the requirements detailed in Chapter 7 since we want to guarantee the evolution of our product line. The rest of the chapter is organized in three sections: the design of the architecture for our tool based on the typical architecture for the software product lines; our derivation engine, based on scaffolding technique, that is not only able to assemble components in the classical way but also to generate product-specific code; an enhancement made to GISBuilder in which all of its components are able to run within the browser and that enables a previewing component so the analyst can view the product without any deployment.

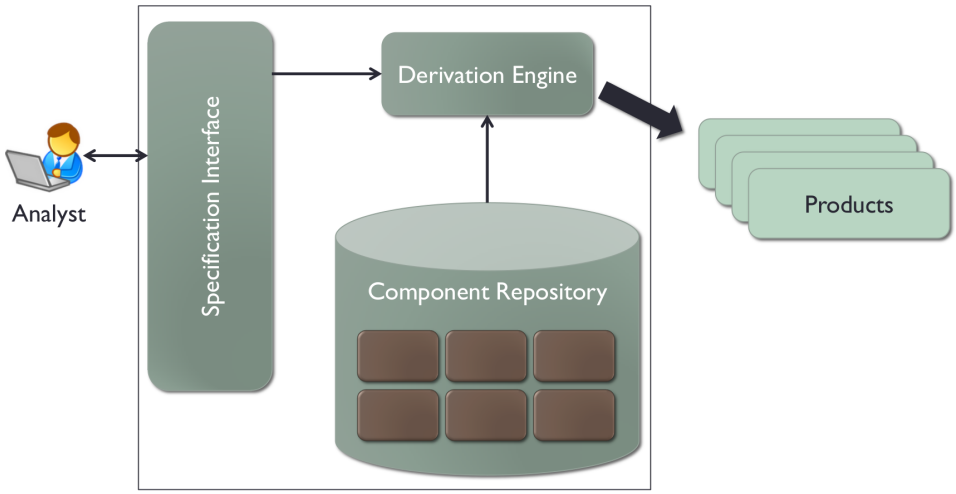


Figure 9.1: Classical architecture for a SPL

9.2 Architecture

The architecture of our tool is very similar to the typical architecture for a software product line, which we can see in Figure 9.1. The *analyst* interacts with the *specification interface* to select the features that he or she wants in a concrete product. Once the configuration of the product is finished, the *derivation engine* takes the set of selected features and assembles the related components into a final product. The components are stored in the *component repository* so the derivation engine can access and take the ones it needs to assemble each product.

The architecture for our tool is quite similar, as we can see in Figure 9.2. Most of the modules are analogous to the previous architecture, with the exception of the *project repository*. This module stores the specification of all the generated products, as we describe below together with the motivation of this new module. Next we describe each one of this modules.

The *specification interface* allows an *analyst* to define different products using two strategies: on the one hand, he or she can select the features included in the product as in any other SPL; on the other hand, the interface provides tools for the analyst to define the data model, the menu structure, and the lists, forms or map viewers that are included in the final product. Behind the scenes, the interface builds the product specification, an instance of the GISBuilder domain specific language (DSL) represented as a JSON document, and validates it using

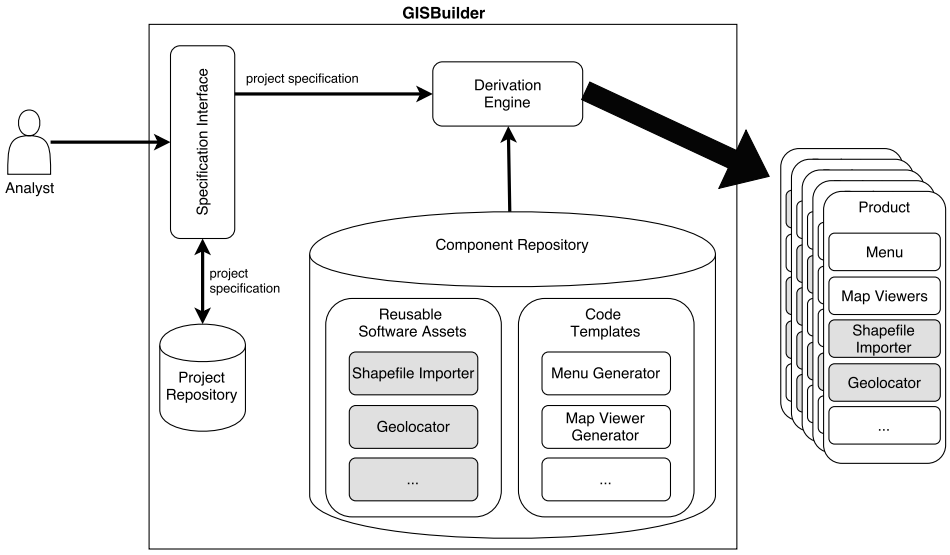


Figure 9.2: Functional architecture of GISBuilder

JSON Schema [WA17]. JSON Schema is a vocabulary to describe the data format that a JSON document should have. This way, we can later check if any JSON document complies the particular structure and therefore validating it is adequate to be processed by our derivation engine. The schema currently used by GISBuilder is shown in Appendix A.

The *project repository* is a database where all the product specifications are stored as JSON documents. This way, an analyst can restore a previously defined product and generate it again, refine it, or create a new version of it. This is important as the products built will evolve over time, and keeping the specification prevents the analyst from having to configure the product again from scratch when probably a new version will have small changes over the previous one. In the case of a classical SPL in which the configuration of a product corresponds to the selection of features for it, this module would not have sense. However, the specification of our products is more complex than that, and the process for the definition of a product can be pretty long and complex. The reader can get an idea about the complexity of the process in Chapter 10.

When the analyst decides that the specification of a product is finished and the product is ready to be generated, the JSON document representing the product specification is sent to the *derivation engine*. Then the *derivation engine* assembles

the different components and generates the required product-specific code, using the reusable software assets and code templates from the *component repository* to achieve this. The output of the engine would be the source code of the product specified by the analyst. Being the *derivation engine* the most important part of our system, we carefully explain it in Section 9.3.

In software product lines it is common that the specification interface is simple or almost non-existent, since it is only used to select which features are included in the product. However, due to the complexity of our products and their definition, we decided that the specification interface of GISBuilder should be a proper application, and particularly a web application in order to be able to use the tool without any installation, from everywhere and with any device.

In terms of the technology, using a framework that allows us to decouple the different modules is very beneficial. Also, there is more need for flexibility than stability, since our platform is an evolving set of artefacts whose components are supposed to grow in size and quantity, and it is not expected to be used by many people at the same time. That is the reason to implement it on Node.js[1], a platform build on Chrome's JavaScript runtime[2] for easily building fast applications. It is lightweight, independent of operating systems and IDEs and currently one of the most important platforms, with growing popularity. Node.js provides for a huge flexibility that facilitates the integration of its libraries and applications.

Therefore, GISBuilders *specification interface* is a web application implemented with Angular that communicates with a Node.js server via REST. This server handles the interaction with the *project repository* and with the *derivation engine*. Since the project specification is represented with a JSON document, the technology chosen to store these specifications is MongoDB, a document oriented database that handles the data precisely in this format. The *derivation engine* is also a Node.js tool, so the integration is straightforward using an API provided by the engine. Lastly, the *component repository* is nothing more than a directory with files of the code of every asset and template, which are accessed directly by the *derivation engine*.

9.3 Derivation Engine

Our derivation engine consists on three different components, as we can appreciate in Figure 9.3: the *variability handler*, the *file manager*, and the *template engine*. The *derivation engine* itself defines an API used by the *specification interface* to

[1] https://nodejs.org/en/
[2] https://developers.google.com/v8/

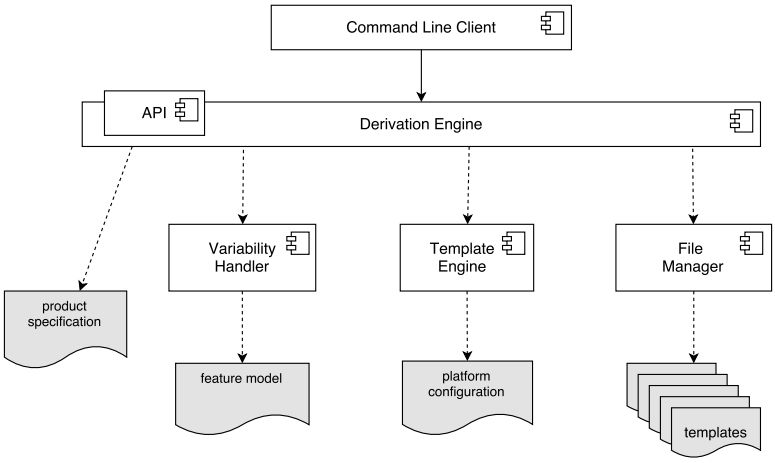


Figure 9.3: Component diagram of the derivation engine

generate the different products. It also provides a small command line utility so the engine can be used independently of the tool, which is specially useful when developing or debugging the platform.

The *variability handler* is the component in charge of validating the feature selection made by the analyst. To do that, we first need to load the feature model of our SPL. We can do that by providing the feature model in two different formats, both of them allowing to define cross-tree constraints: as a JSON document following a schema designed by ourselves, or as a XML document with the same schema used by Feature IDE [KTS+09]. Therefore, we can design our feature model graphically with this tool and then take the XML file that represents it. The feature model can also be described programmatically using an API provided by the *variability handler*. In Figure 9.4 we can see a excerpt of the feature model of our tool, whereas in Figure 9.5 the XML of its definition is shown. This component validates the feature selection made by the analyst in real time: as soon as the analyst selects one feature or another, it evaluates the feature model and the cross-tree constraints and returns the proper warning message. If this evaluation concludes that any other feature must be selected, such as a mandatory child feature or any other feature that should be selected due to the constraints, this information is also returned so the specification interface can act accordingly. This library has been designed independently of the rest of the tool so it can be integrated in any other tool that requires feature model validation. Also, there are many other operations for feature models [BSRC10] that can be added in the future.

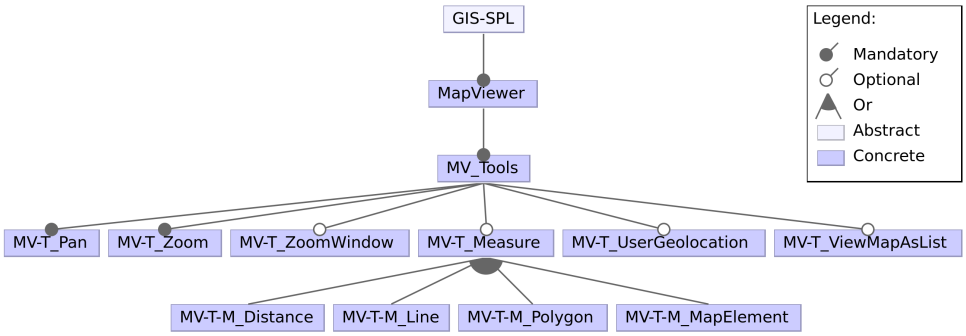


Figure 9.4: Excerpt of the GISBuilder feature model

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<featureModel>
    <properties/>
    <struct>
        <and abstract="true" mandatory="true" name="GIS-SPL">
            <and mandatory="true" name="MapViewer">
                <and mandatory="true" name="MV_Tools">
                    <feature mandatory="true" name="MV-T_Pan"/>
                    <feature mandatory="true" name="MV-T_Zoom"/>
                    <feature name="MV-T_ZoomWindow"/>
                    <or name="MV-T_Measure">
                        <feature mandatory="true" name="MV-T-M_Distance"/>
                        <feature mandatory="true" name="MV-T-M_Line"/>
                        <feature mandatory="true" name="MV-T-M_Polygon"/>
                        <feature mandatory="true" name="MV-T-M_MapElement"/>
                    </or>
                    <feature name="MV-T_UserGeolocation"/>
                    <feature name="MV-T_ViewMapAsList"/>
                </and>
            </and>
        </and>
    </struct>
    <constraints/>
    <calculations Auto="true" Constraints="true" Features="true"
     Redundant="true" Tautology="true"/>
    <comments/>
    <featureOrder userDefined="false"/>
</featureModel>
```

Figure 9.5: Excerpt of the XML representing the feature model

The ***file manager*** handles all the tasks related to the access to the templates. For example, it allows the *derivation engine* to walk through every template of a directory recursively and apply changes to them. It also detects when a binary file is found in order to skip processing it and just copy it to the output.

The latest and more important part of the *derivation engine* is the ***template engine***. In a SPL, there is usually a derivation engine handling the process of assembling the product from a set of features using the reusable components. In MDD, transformations are applied to models to generate new models in different levels of the architecture, until one of these transformations generates source code. Our ***template engine*** is able to handle these two kinds of operations, being able to generate products from:

- **SPL-like assemblable components**: for each product, it can be specified a set of features to be included in it, like in a classic SPL. Our engine takes the components implementing these features and assembles them into the final product.
- **MDD-like model transformations**: a set of models are used to generate each product. These models specify the data model, layers and maps of each product, as well as many other generic parameters like graphical interface options. Our engine transforms these models into specific modules included in the final product.

In order to handle this duality, we have designed the *template engine* to be based on the *scaffolding* technique. It treats every module as a set of templates, no matter whether the module is a reusable component that needs just to be assembled or whether its code must be specifically generated from the application models. Thus, the *component repository* described in Section 9.2 is nothing but the templates of every module, including the code for the *reusable software assets* and the *code templates* that are used to transform the modules into product specific code.

In Figure 9.6 a excerpt of the Java code for the geographic data importer is shown. In the *static* block we can see some variation of the code to produce depending on the selected features. Template annotations are defined as comments of the programming language in which the template is written and its content can be any JavaScript code. Thus, they do not interfere with the compiler, IDE or validation tool used by the developer of the platform code.

Besides annotations related to which code blocks are included in the product, our template engine allows using variables and complex control sequences in the code. Moreover, the developer of the platform can even create JavaScript functions to use within the annotations, or use temporal variables to store data used more than once.

```
private static final Set<String> validExtensions = new HashSet<String>();

static {
    /*% if (feature.DM-DI-DF_Shapefile) { %*/
    validExtensions.add("shp");
    /*% }  %*/
    /*% if (feature.DM-DI-DF_Raster) { %*/
    validExtensions.add("tiff");
    validExtensions.add("gif");
    /*% }  %*/
    /*% if (feature.DM-DI-DFNetwork) { %*/
    validExtensions.add("pbf");
    /*% }  %*/
}

@Component
public class DataImporter() {
    // ...
}
```

Figure 9.6: Annotated Java class

```
/*%@ return entities.map(function(en) {
    return {
        fileName: en.name + '.java',
        context: en
    };
}) %*/
package es.udc.lbd.gisbuilder.model.domain;

@Entity
@Table(name = "t_/*%= context.name.toLowerCase() %*/")
public class /*%= context.name %*/ {
    /*% context.properties.forEach(function(prop) {
            var propertyClass = prop.class;
            var validGeomTypes = ['Point', 'MultiLineString', 'MultiPolygon'];

            if (validGeomTypes.find(propertyClass) != null) { %*/
    @JsonSerialize(using = CustomGeometrySerializer.class)
    @JsonDeserialize(using = Custom/*%= propertyClass %*/Deserializer.class)
    @Column(columnDefinition="geometry(/*%= propertyClass %*/, 4326)")
        /*% } %*/
    private /*%= propertyClass %*/ /*%= normalize(prop.name) %*/;
    /*% }); %*/
}
```

Figure 9.7: Simplified excerpt of model transformation template

In Figure 9.7 a simplified template to generate entities of a product is shown. The template specifies how to create Java classes for each entity defined by the analyst. Inside the class, the code for each property of the entity is created depending on its specification. The latter is an example of a *code template* generating product specific code, whereas the former example is just a *reusable software asset* with an small variation depending on the sub-features selected. Even with the latter template simplified, we can see the difference of complexity between the two types of templates.

As we could see along this section, our scaffolding engine works somehow like a preprocessor but the rich semantic of JavaScript provide a high level of capabilities in the annotations in a very flexible way without compromising the easiness of usage. By basing our tool in this technology we are providing the developers a way to implement the variability without requiring to learn any new language, and since our engine is implemented in Node.js it can be integrated with any other tool for web development since most of them (Gulp[3], Grunt[4], NPM[5], etc.) are based also in Node.js.

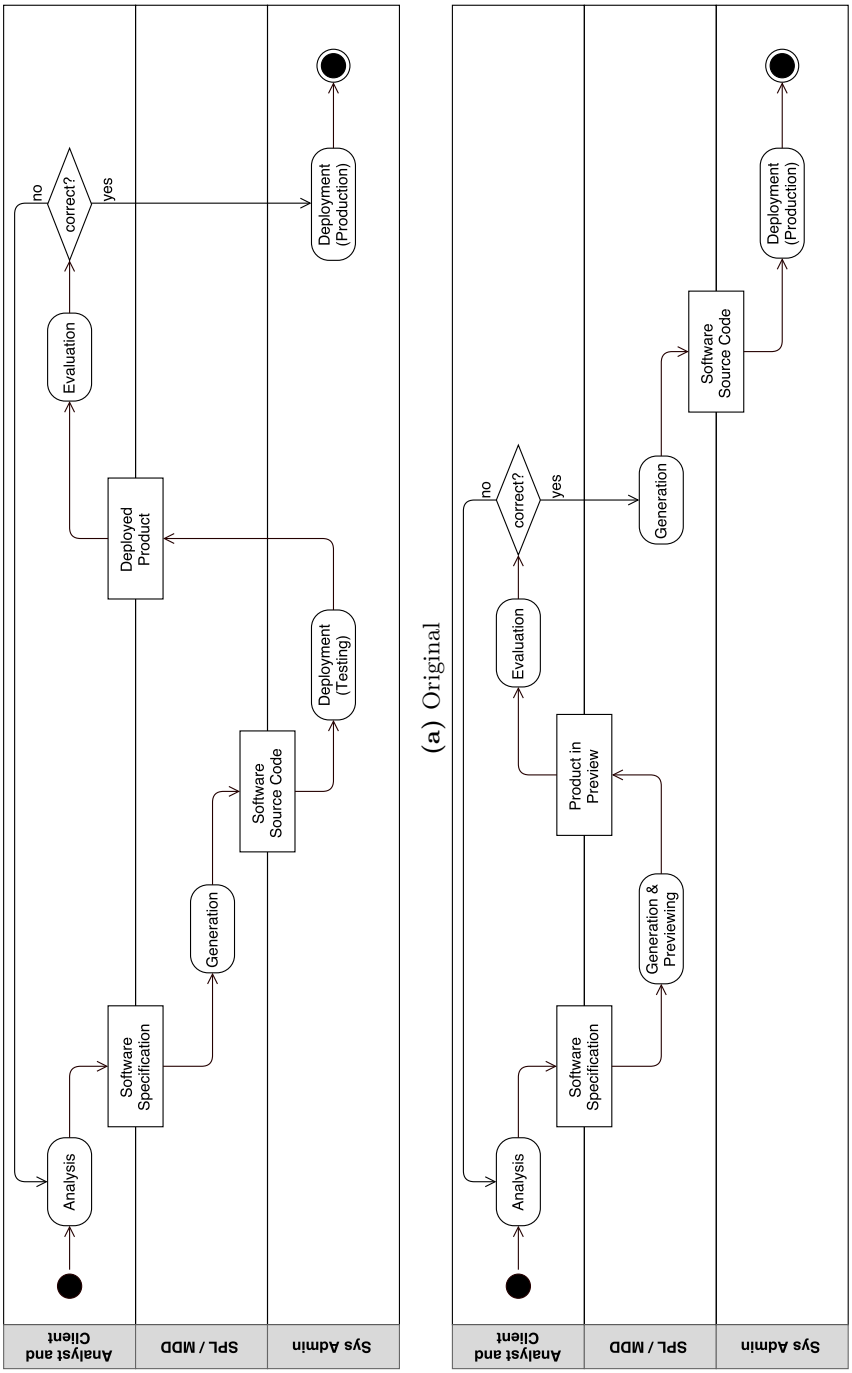
9.4 Runtime Product Preview

9.4.1 Motivation and conceptual approach

SPLE, MDD or similar methodologies for the semi-automatic generation of software tackle intrinsic repetitive structures in the development of software products, either full software components like the ones assembled within a SPL, or actual generation of code from high level descriptions and model-to-model/code transformations like in MDD. These approaches are particularly suited to the iterative evolution of the project, by means of small improvements which can be easily tested and validated. However, the nature of web applications makes the evaluation of these new features complex, due to the fact that a new full deployment is necessary after each change.

Continuous deployment [CSA15] is particularly suited to bring these new functionalities to a set of evaluators, but it is not suitable for an integration in the development cycle itself due to its complexity. The main problematic in trying to integrate an evaluation of the final product within SPLE or MDD is related to the full code generation intrinsic in these approaches. Changes to the final product cannot be applied directly to the deployed application, but instead the high level description needs to be updated and a full code generation triggered.

[3] http://gulpjs.com/
[4] https://gruntjs.com/
[5] https://www.npmjs.com/



(a) Original

(b) After applying our approach

Figure 9.8: Activity diagram of the development process

We show in Figure 9.8a a simplified workflow with processes and stakeholders involved in the development of a web application with automatic or semi-automatic generation of software. We can see how the analyst, with input from the client, elaborates the *Software Specification*; then the generation of the product, which can be based on SPLE, MDD or any other methodology, begins. The generated software has to be deployed for evaluation by a different actor, the system administrator. At this point, the analyst and the client can, together, evaluate the product and determine whether the product is the one required by the client or some changes are required. If the product is finished, then the system administrator can deploy the final version into production environment and the process ends. If the product is not complete, then the process starts again and the system administrator is involved, again, to make a redeployment in the testing environment. There are some cases when the analyst is the one that deploys the product for evaluation. In these cases, even when there is not an extra actor involved, this deployment still needs a complex environment and it is time consuming.

We propose an approach to simplify and improve the workflow for the development of web applications through automatic generation techniques by removing the necessity of an actual deployment during the development process. Only when the analyst decides that a project is correct, the full-source is generated and deployed.

The workflow, when our approach is applied, is shown in Figure 9.8b. It is analogous to the previous one but, in this case, the same tool that generates the product provides a preview of it to the analyst and/or to the client. Therefore, it does not involve a real deployment of the product at this point, so the third actor, a system administrator, is not required, nor a complex environment or any extra tool for the evaluation deployment of the project. This tool we are talking about is nothing but a web browser.

Modern web browsers are able to execute code generation frameworks by means of JavaScript code, and they are able to fully or partially execute the generated applications by means of *iFrame*s and *WebWorker*s. Of course, we cannot expect that absolutely every feature of the product can be previewed this way but, using mocks and similar techniques, the client can have a real idea of how the product is and introduce the required changes on the earliest stage. The generated application can be evaluated directly inside the browser following three different strategies, complemented using mocks responding to any XHR request.

Code execution. Applications fully developed with JavaScript can be fully executed inside the browser. The client-side part of the application is executed inside an iFrame which is able to fully resemble a standalone browser. The server-side part of the application is executed inside a properly instrumented WebWorker

able to resemble a NodeJS environment.

Full emulation. Applications developed with different technologies can exploit the full code generation intrinsic to SPLE and MDD to trigger a different version of the transformation. This way, it can generate a functionally equivalent version of the application in JavaScript which can be tested using the previous strategy.

Partial emulation. Applications developed with a mixture of JavaScript and other technologies can use a mixed strategy; this is, the non JavaScript parts can be replaced by a functionally equivalent version and then the product is executed within an iFrame.

All the three strategies can be used to evaluate the final application without the need of complex server side deployed infrastructures, increasing productivity and reducing tools configuration complexity.

We have tried our approach within two different tools [CBLF17], one based in SPLE and the other one in MDD, but our approach can be applied to any other software generation technique as long as the engine is built with JavaScript, which is the programming language that can be run on a web-browser[6]

9.4.2 Improving GISBuilder with Runtime Product Preview

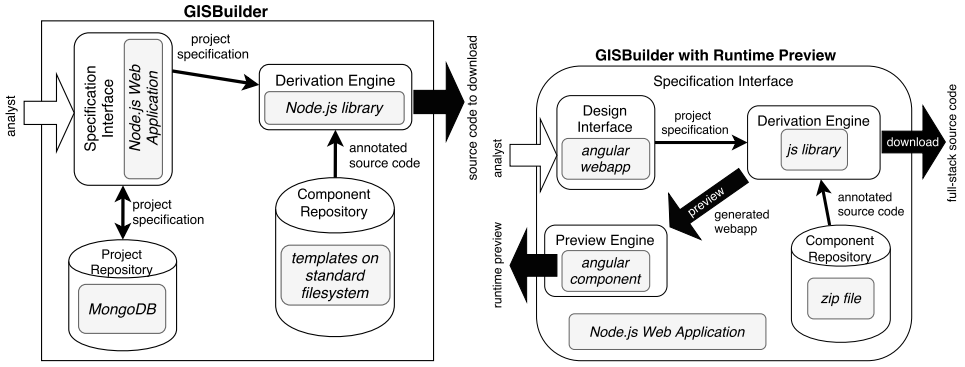
GISBuilder highly reduces the time to market by reusing components and generating the applications from a set of specifications. However, the products generated, web-based GIS applications, require complex development environments in order to allow users to test and evaluate their design, since every time the product is built it needs to be deployed in a web server with all the required software available. This process is slow and it usually requires more than one person, since the analyst is not in charge of preproduction deployments. In the case the analyst is building an application for a client who provides feedback which derivates on changes on the definition of the product, the product needs to be fully redeployed. Therefore, the interaction between the analyst and client will be slow also, even for small changes.

We want this interaction between clients and analyst to be in real-time. To achieve this, we have applied the approach presented in the previous section an we have enhanced GISBuilder to generate and show a preview of the designed products at runtime, directly on the browser, without the need of any server-side structure.

The changes required affect to the very architecture of GISBuilder, as we can see in Figure 9.9b. The main changes made are:

1) We have implemented a new version of the **derivation engine** that is able to run entirely on the web browser. Instead of getting the component templates

[6] In any other case our approach is still conceptually valid but the generation must occur in the server side and after it, the source code must be loaded within the web client.



(a) Original architecture **(b)** Adapted architecture

Figure 9.9: Architecture changes in our tool

from the file system, they are loaded from a zip file. Similarly, it can generate the products in memory and provide a zip file with them.

2) The **derivation engine** is integrated within the **specification interface**, as well as the **component repository**, provided as a ZIP file.

3) Our preview component intercepts XHR request of the previewing application and returns mocks responses to each specific REST petition.

4) GISBuilder produces full-stack web applications with Spring in the server side and Angular in the client side. In the adapted version, GISBuilder creates two different versions of the products, depending on whether the analyst wants to preview them or to download the full-stack version.

5) To make the tool runnable without any deployment structure, the **project repository**, a MongoDB instance, is now optional. Since project specifications are just JSON documents, our tool now provides features for downloading and uploading them.

This way, after a reconfiguration of the product, the analyst can simply run its preview and show the client its aspect. Of course, the previewing component does not run every feature of the SPL, but it can still show enough to provide the customer a realistic view of the application. When the client is satisfied with the preview, the actual full-stack version of the product can be generated and deployed, just as before.

10 Validation of GISBuilder

In this section we describe the usage of GISBuilder. First we introduce a case of use, an example of an application that belongs to the family of products that our SPL can generate. We have designed a transport management application that requires route calculation and digitizing, among other features. This application is described in the next section. Afterwards, GISBuilder specification interface is shown, describing each of its parametrization options and using the mentioned application as an example.

10.1 Case of use

Let us assume we have a transportation company with a set of warehouses. For each warehouse, there is a number of trucks, which can be more or less depending on how big is the warehouse, its location or many other factors. Every day the company has a number or orders for picking up merchandise or packets and delivering it at other points. Each warehouse controls its near points of picking/delivering packets (we will call them "pick up locations") and every morning the route for each truck is calculated depending on the orders. During the night, the packets are transported to the warehouses of their area of destination, so another route calculation takes place. Therefore, the trucks make two routes in a day, one during the day and one

during the night. However, not every truck is needed every daytime or night time and, in fact, the company want to minimise the number of routes required every day. To summarise, this is the process from the moment of placing the order to the moment of final delivering:

1. The order is placed.
2. The next day, the picking up location is taken into account when the routes are calculated. A truck is assigned to the route and picks up the packet.
3. At the end of the route, the truck comes back to the warehouse and stores the packet.
4. The routes for the transportation between warehouses are calculated, and a truck moves the packet to the destination area.
5. The next day, the delivering location is taken into account when the routes are calculated in the new warehouse. The truck assigned to the route delivers the package.

The warehouse managers want a system that helps with this workflow, handling the route calculation, providing a way to import the warehouses and pick up locations and to edit this locations manually afterwards. They also want to manage the trucks associated with each warehouse and they want to know the position of each truck and their planned routes in real time in order to decide which is the most suitable one to pick up an order that appears during the day. Of course, they want that only a set of users can manage the trucks, warehouses and routes, but they want that the drives can view their routes.

Using our SPL we can speed up the development of this example application, whose features are not totally basic, by following the next steps. The first step when creating a new product is to choose the name, version, identifiers (maven-like group and artefact ids) and languages of the product. After that, the features included in the product are selected. In this case, a non-exhaustive list of the features required would be:

- User management with registration by admin. The user accounts for truck drivers should only be created by the administrators, not by anonymous people accessing the application.
- Static pages. Probably some documentation needs to be provided to the users of the application.

- Geographic data import. The first time the application is deployed the managers need an easy method to load all the information about the warehouses and pick up locations because digitizing all the information manually seems a hard labour.
- Digitizing. They probably need a way to introduce a new geolocated element without needing to import it.
- Route calculation. From the description of the problem it is obvious that this feature is required.
- Map viewer. We need to add this feature to allow visualising the warehouse, the pick up locations and the drivers.

For the data model, the analyst may define a structure with the next entities:

1. Warehouse. Each warehouse, apart from the autogenerated identifier, has a name that serves as a textual identification. Each warehouse also has an address, an email and a telephone as contact information. Of course, the location of the warehouse must be also stored. The system also has to keep the set of trucks belonging to each warehouse, and the pick up locations associated with it.
2. Truck. Each one has a brand, a model and a license number. Thinking already in the real-time viewer of the position of the trucks, the analyst also wants to store the location of each truck. It is possible that some trucks have some problems and are in the garage being repaired, so they are not active. The analyst can represent this is information with an enum property. Besides, every day the trucks are associated with a different set of pick up locations, so this information also needs to be stored as a relationship with the next entity.
3. Pick up location. The pick up locations can be any kind of business, mostly shops and offices, so the analyst needs to store the name and opening times for every location. The address and its location must be also saved.

The application does not require additional geographical information apart from the location of the items from the data model. These locations can be stored as points in the database and loaded using an external source, such as OpenStreetMap[1], as the base layer of the map. Therefore, this application does not need a map server.

For every entity described, the analyst has to define lists, forms and maps viewers that allows the administrator to view and edit the different elements.

[1] openstreetmap.org

The specific optimization for the route calculation must be designed and implemented manually extending the route calculator components. Since the routing calculation occurs always at two particular moments of the day, probably a scheduler that runs it should be also implemented manually. A mobile application or some method to get the real-time position of the trucks must also be implemented, but the REST service provided by our product is able to handle the data collections task in the server side.

In this case, our SPL may not be able to generate the full code of a specific application. However, this is not the objective as much as to speed up the time to market of new products. By generating everything except the few mentioned details, GISBuilder saves the time spent on implementing the most general functionalities and gives developers a quality product to extend with the specific functionalities.

10.2 Using the specification interface

We have described the product that we want to generate, and the analyst, at this point, should have analysed the requirements and designed the solution. Therefore, he or she is able to start configuring the product by accessing the starting page of the specification interface, seen in Figure B.1. The left menu is used to navigate through the different sections of the application, where we can configure the product to build in many ways. The different sections of the specification interface are described next. All the figures referenced along this chapter are group in Appendix B to improve their clarity.

- *Project*: general information of the project, including some global parametrization.
- *Features*: selection of features to be included in the product.
- *GUI*: graphical user interface design.
- *Menus*: definition of the menus of the product to be built.
- *Data Model*: design of the data model specification for the product.
- *Static Pages*: creation of the non-dynamic content of the final product.
- *Forms*: definition of the forms.
- *Lists*: definition of the lists.
- *Maps*: definition of the maps.

- *Generate*: runtime previewing of the product and full source code generation.

Note that the first section allows the analyst to parametrize and globally configure the project, whereas the second section corresponds to the typical feature selection in any SPL. The next seven menu entries correspond to what we have called "parametrization" of the product (see Chapter 7), in which the analyst specifies the aspects more specific to each particular product, such as the data model design or the organization of the menus. Finally, the last menu entry allows the analyst to preview the product in construction directly on the browser, so he or she can check the result of its specification before really generating the product. In this section, the analyst can also download the full-source of the product.

Now, we will detail all the functionality of the interface specification, using the case of use described in the previous section as the running example.

10.2.1 Project: basic data and languages

In this section of the interface, shown in Figure B.1, the analyst indicates certain parameters that affect the project in a global way, such as its identifier and version or the welcome page that will be displayed to the end user when he or she accesses the web application.

In software development there are some common tools that are used to manage the projects themselves. These tools are called *build managers*. Among other functions, they are responsible for managing the set of dependencies of the source code. In order to carry out this task it is necessary that all the projects are uniquely identified. By convention, this unequivocal identifier consists of three parts: a *group identifier*, which is usually the same in projects of the same development team or company, an *artefact identifier*, unique among the projects of the same group, and a *version identifier*. All three identifiers are alphanumeric and cannot contain any spaces or odd symbols except the hyphen symbol (-). The most common two tools of this nature for Java projects are Maven[2] and Gradle[3]. The products generated by our SPL use a build manager. Therefore, the analyst must indicate these group, artefact and version identifiers.

Next we list the parameters to indicate in this section beyond those already mentioned.

- *Project Id*: the identifier of the project, which is used when we address the project. In our example the project id is "Route Manager".

[2] https://maven.apache.org/
[3] https://gradle.org/

- *Version*: the version of the loaded project. Each project can have many versions, each one with its specific configuration. The version is used to differentiate among them, since all of the versions associated with a project share the project identifier. GISBuilder does not force any convention for this parameter, so any alphanumeric string can be the version of a project. Examples of valid versions are "1.0-SNAPSHOT", "1", "4.3.2", "v1.0" o "alpha-0.2". In Figure B.1 the version is "0.1.0".

- *Package Info*: the two fields within this group are used to identify the product within the build manager, as explained above.

  - *Group Id*: the group identifier. In the example, "es.udc.lbd".
  - *Artefact Id*: the artefact identifier. In the example, we use the project identifier in lower-case and changing the space for a hyphen, "route-manager".

- *Index Page*: the generated product is a web application and, as such, it has a web page called *index* which is the first page to load when a user accesses the web URL. The index page can be anything, like a static page with some welcome text, a list of articles in the case of a blog or the user authentication page in case the web has restricted access. In our tool, the index page can be a view of any component, so it is defined by two fields:

  - *Component*: the component to which the desired view belongs.
  - *View*: the specific view. At this point of the specification process it is possible that this view is still undefined. For example, if the analyst wants to show a static page, it may not be defined yet. In this case, the analyst can leave this field blank and come back afterwards to this section to set it. In the example we have choose the authentication page from the user management component. So the first page that a user will see when accessing the application will be a page to login.

- *Languages*: the generated product supports internationalization, so the analyst indicates the languages required. In Figure B.1, the analyst selects English, Spanish and Galician. This means that in the final product there will be three sets of *localization files*, one for each language. The localization files provide pairs key/value for every string in the web application, from the menu items to the properties of the entities.

10.2.2 Features: variability selection

In a classic SPL, the selection of features to be included in a product is usually done just by checking which features are desired. In our case, some of the features can be selected exactly that way whereas some other feature require some complex parametrization. In this section of the specification interface, the analyst selects the features that do not need to be parametrized. The parametrization of the rest of the features, which are actually design tasks, is addressed through the next sections.

The variability selection of our SPL is essentially a list of checkboxes of different levels that are associated with the different features that a product may or may not have. For each checkbox, the name of the feature it represents is displayed. In addition, each may have a set of derived features, which can only be selected if the "parent" feature has been previously selected.

The specification interface checks the validity of the feature selection. It also selects those dependent on each other automatically. For example, the "Static Pages Management" feature (*GUI-SP_Management*) requires the "User Management" feature (*UserManagement*). Therefore, if we select the first one, the second one is marked automatically, since it is a dependency. The interface also prevents that more than one characteristic of the same level is marked if they are alternative features (choose only one feature). In fact, the feature model initially shows more than the features of first level, since some of them are already mandatory and therefore they are automatically selected, as we can see in Figure B.2.

Once the analyst selects one feature, its subfeatures are shown. For example, if we select the "User Registration" feature (*UM_Registration*), two new features, "User Registration By Admin" (*UM-R_ByAdmin*) and "User Registration By Anonymous" (*UM-R_Anonymous*) are displayed, anchored to it. We can see that in Figure B.3.

In Figure B.4 and Figure B.5 we can see the full selection of features for our example.

10.2.3 GUI: designing the interface

The graphic aspect of the product to be generated is specified in the section labelled GUI. This menu entry shows the analyst different base templates to choose among, as we can see in Figure B.6. The chosen template conditions the number and position of the menus of the final application.

After choosing the desired template, the analyst configures it with the following parameters:

- *Header*: configuration regarding the header. It has the next options:

  - *Type*: it can be a text or a image.
  - *Header Id*: if the header is a text, the analyst indicates in this field the specific text that should appear in the header. The text indicated here will be also the default key for the localization files.

- *Image*: if the header is an image, the analyst selects which image he or she wants to be displayed as a banner of the final application.

- *Font*: settings relative to the font size. It has the following options:

  - *Family*: the analyst indicates which source family wants to use the product. The source family is the type of typography used for all the texts of the final application generated.
  - *Size*: from *x-small* to *x-large*, it allows the analyst to choose the relative font size. The larger you choose, the larger the texts of the interface of the generated product will be.

- *Colors*: although the analyst does not choose each color of the product individually, he or she can choose a set of colors from a set. These colors will be used in the style of the final application.

- *Authentication on Menu*: this option only appears if the user management feature is selected. If the analyst activates it for a product, there will be a section in the menu of the product reserved to display information about the logged user or some inputs to log in. That is, an icon to access the authentication page if the user is not authenticated, and once it is, a link to access your personal profile and another to close your session. In case the analyst does not want to have this features in the menu of the product, he or she can associate it with some menu entry.

10.2.4 Menus

By choosing a template in the GUI section, as described above, the analyst determines the number and position of menus in the product. For each one, a new submenu item is shown under the "Menus" entry. For example, in Figure B.6, the analyst chose a design that only has a top menu. Therefore, under the "Menus" entry there is only one element called "Top", as we can see in Figure B.7. Clicking on this item he or she can start parametrizing this menu. If the analyst had chosen the second template, then we would have two elements in the side menu to configure the menus, "Top" and "Left", and clicking on each one would configure these menus.

In Figure B.7 we see the definition of the first menu item, a static page that will appear with the default title of *How to use the application?*. We can see that this menu item is a "View", i. e. a web page associated with one of the components that will be included in the final product. In this case, the view is from the component "Static Page" and its name is "Instructions" (defined below in Section 10.2.6). In Figure B.8, we can see another menu entry of this kind with a view from the component "List".

Another type of menu entry is the submenu, used to group a set of other menu entries itself. We can see an example in Figure B.9, "Lists", in which we can see a set of menu entries linking with views that are, all of them, lists. In Figure B.10 we show the last kind of menu entry: a link to an external URL.

The form in which the analyst defines each menu entry consists on:

- *Item Id*: identifier of the menu entry, which is used both to identify the entry itself and to generate its default label text in the localization files.
- *Type*: menu entry type, to choose between:
  - *Menu*: entry that opens a submenu with a set of other menu entries, set also by the analyst. The entries belonging a submenu can be also of any kind.
  - *URL*: this menu entry creates a link to a web address that the analyst indicates in a text field.
  - *View*: entry linked to a view provided by a component. The analyst in this case needs to select:
    - *Component*: the component to be visualized, among those available depending on the selected variability.
    - *View*: the view itself from the ones available in the selected component. There are views that are generated in the specification interface itself, such as static pages, forms, lists or maps. If the analyst wants to link one view that has not been created yet, he or she may leave this field blank and come back later, after the creation of the view.
- *Access*: if the user management feature is activated, it is possible to indicate, for each menu item, which user roles can view and access it. This way, there are three different checkboxes to control the visibility and access to each entry that corresponds with the roles of the users: *Admin*, *Logged Users* and *Anonymous*. The checkboxes are not exclusive and, in fact, in a menu entry is available

to more than one of the user roles, the analyst needs to select more than one role. For example, if a menu item is visible to anyone, then the analyst should check the three checkboxes. If the analyst only activates the checkbox for the anonymous user, then the menu entry is only visible before the user authenticates. This is useful to show, for example, the "Sign in" menu.

10.2.5 Data Model

The analyst specifies the data model of every product by defining the entities, properties and relationships between them. Since the use of the enum data type in Java applications is quite common, the analyst can also define enums and then associate them with the entities.

For our example, only one enum is required, shown in Figure B.11. This is the page with the list of enums of the application, initially empty until the analyst adds a new enum using the button labelled with the plus (+) symbol. This would be the way to operate in most of the next sections of this interface. These enums can be used as classes in the form to set the properties of the entities.

To design the entities of the product the analyst has a complete form like the one we see in Figure B.12. In this form the analyst indicates the name of the class itself, its set of properties and what we have called the *display string*, which is useful to indicate how each instance of an entity is represented when it needs to be referred in the web application. We explain more about how the display string works and its usefulness after describing the fields to input for each property of an entity.

- *Property.* The name of the property. It also serves as key and default value for the localization files.
- *Class.* The class of the property. The available options are:
  - *Basic types and common Java classes*: Boolean, Long, Long auto-incremental (mainly useful for primary keys), Integer, Double, Decimal, String, Date and DateTime.
  - *GIS-related classes*: Point, Line, Polygon, MultiPolygon...
  - *User-defined entities*: entities that were previously defined for the product. In this case, a relationship is established with this entity.
  - *An enum* previously defined.
- *Primary Key.* The property is the primary key of the entity. Only Long, Integer or String properties can be primary keys.

- *Required.* Property is required. That is, the value of the property cannot be null.
- *Multiple.* The property is a collection of elements (a list).
- *Unique.* The value of the property is unique. In case the property is not multiple, the analyst may want the values of this property to be unique. For example, if we have a list of trucks, we want the string of the license number to be unique.
- *Default.* If the property is not multiple and its class can be serialized as a string (i.e, string, integer, enums...), the analyst can indicate a default value.
- *Bidirectional.* If the property is another entity, the relation can be bidirectional. In this case, the analyst indicates the name of the property of the target entity, which must also be indicated as bidirectional in the form of the other entity.
- *Relationship owner.* If the relationship is bidirectional, one of the sides must be the owner of the relationship.

Regarding the display string field, we can explain its usefulness better with an example. In the web application that will be generated from this specification, it is possible that, at some point, an instance of a truck needs to be displayed as a "readable" string which provides the user enough information as to identify the specific instance. One choice would be to use always the primary key, as we know that this property will always identify an instance of an entity unequivocally. However, we consider that allowing the analyst to set which is this text benefits the final user. For example, for identifying a truck probably it is better to use the license number than the identification number, automatically generated by the database. Therefore, the analyst can set the display string to "$licenseNumber", and in the final application the trucks will be referred as "1234BCD" or "6666JJJ". If the analyst prefers to represent each truck with its brand and model, the display string would be "$brand $model".

Other examples of entities definition are shown in Figure B.13 and in Figure B.14, and we can see the complete list of entities defined for this example in Figure B.15.

10.2.6 Static Pages

The static pages are defined using a "WYSIWYG" editor that provides the classical options of any HTML editor of the style: bold, italic, underlined, hyperlinks,

numeric and dot lists, citations, tables... This kind of editors are very simple to use and allow the user, in this case the analyst, to design the page using graphical elements instead of directly editing the HTML code. In addition, the content of the page is displayed as it would be in the final application at the same the edition occurs. For advanced use cases, the possibility of directly editing the HTML is also provided.

In the page listing the static pages, shown in Figure B.17, the analyst can see which elements are already linked with some menu entry. This serves as a visual aid to the analyst when configuring the various menus of the application, as he or she can easily notice the unlinked elements. This behaviour is repeated in the rest of the lists.

10.2.7 Forms

The analyst can define a set of forms so that users can view, create, edit and delete elements manually from any of the previously defined entities. Next we describe the different options:

- *Form Id*: the identifier of the form to be generated, so it can be referenced in other sections of the specification interface.
- *Entity*: the name of the entity of the data model, which must be already specified, for which the analyst is creating a form. When an entity is selected, the list of properties or attributes that can be shown in the form are updated, and the analyst can select which ones the final user can view and/or edit using this form.
- The analyst has a number of options to indicate the functionality of the form:
  - Creatable: the form serves to create new elements of the corresponding entity.
  - Editable: using the form the end user can edit existing elements of the corresponding entity.
  - Removable: the form allows deleting existing items. In case this option is checked, the analyst also decides if the end user has a confirmation modal alert before proceeding to the effective deletion.

In Figure B.18 we can see the complete specification of a form that will allow the user to visualize, create, edit and eliminate elements of the Truck entity. We have explained in Section 10.1 that in this application the information about the

location of the trucks needs to be updated in real-time using some kind of mobile device. Therefore, the analyst does not want that the location of the trucks can be changed manually, and he or she can disable that feature using the options provided, as we can see in Figure B.18. The same way, the pick up locations in which the truck has to stop are automatically set when calculating the routes, and the analyst does not want the user able to change them manually. The rest of elements remain active for edition, except for the automatically generated identifier that cannot be edited by default.

In Figure B.19 we can see the specification of a form for the Warehouse entity. In this case, the properties "trucks" and "pickUpLocations" cannot be showed nor edited in the form because they are both bidirectional relationship and the owner side is not this entity. Allowing to view or edit these properties in the form would require some *ad-hoc* implementation that cannot be automatically abstracted and generated without causing negative side effects. So the analyst, in this case, lets the user to edit all the properties that can be enabled, but he or she disables the possibility of removing a warehouse.

In some occasions the analyst would like to define a form only allowing the visualization of a small set of properties, as the one in Figure B.20. This is useful to use the form to show, as a pop-up, in a map associated with the same entity. An example is shown in Section 10.2.9.

10.2.8 Lists

The listings available in the application are also defined by the analyst in the specification interface. For each list the analyst must indicate:

- *List Id*: the identifier of the list. This field is used to identify the list in the other sections of the specification interface and to generate the pair key/value in the localization files corresponding to the title of the list.
- *Entity*: the entity to list. That is, the list will show the elements of the selected entity.
- *Form*: optionally, a form associated with the list. If the analyst chooses one, next to each listed item appears a link to the form showing the particular element.
- *Link to remove*: the analyst can add a link to each row that allows deleting the item directly.
- *Static Filter*: the analyst has the possibility to filter the visible elements in a list. To do this, he or she uses this property to define logical statements, using

a basic logical grammar that can receive the element properties as parameters, that can be evaluated to true or false. For example, in Figure B.21 we show the definition of a list of the trucks which have a concrete status property. The property status of the entity Truck is of the type TruckStatus, an enum that can have the following values: ACTIVE and GARAGE. If the analyst wants to show only the trucks whose status is ACTIVE, he or she can do it with the *static filter* field, using the statement `$status = ACTIVE`. In addition to the equal operator (=), there are operators such as greater than (>), less than (<) and the conjunctions "and" and "or". To reference properties of the entity, as in the previous example, we can write its name with the dollar symbol ahead $. If we want to list only active trucks but only those whose id is less than 300, we could indicate the following filter: `$status = ACTIVE AND $id < 300`.

- *Properties*: the analyst must indicate which properties of the entity are displayed in the list. In addition, if any of the properties chosen is in turn an entity previously defined in the data model, the analyst can associate a form (using the "Form" drop-down), so the final user can access the form of the related instance by clicking on the value of the property within the list. In Figure B.22, each row of the generated list shows a pick up location, and the value of column "warehouse" will be a link to the associated form.

- *Sorting*: the analyst can decide if the list is sortable or not by columns.

- *Searching*: the list provides the functionality to perform basic searches on the elements of the list.

- *Filtering*: the list allows filtering the elements by column.

10.2.9 Map viewers

The analyst can configure map viewers on entities that include some geographic property. Figure B.23 shows the design of the form to define a map with all the options that the analyst has to input. The different options are described next.

- *Map Id*: identifier of the map, which is unique and serves to identify the map in the other sections of the specification interface. It is also used to generate the pair key/value in the localization files.

- *Entity*: entity shown on the map. Obviously, only those that have some geographic property can be chosen, since otherwise they could not be represented on a map.

- *Geometry*: geographic property that will be used to represent the entity in the map, since it may happen that the same entity has more than one geographic property. Depending on the class of property chosen, the element will be represented accordingly. For example, if it is a point, it will be represented by a marker, whereas if it is a polygon we will see an area drawn on the map.

- *Base Layer*: base map on which to represent the elements. That is, the background map, being able to use servers like Open Street Map or similar ones.

- *Popup View Form*: the end user, on each item shown in the map, can click to get specific information about it. This information will be displayed in a floating window over the map and particularly it will display the properties that have been configured in the form indicated in this field.

- *Form*: the analyst can add a link on each element to access a form related to them. In that case, in the floating window that appears when clicking on an element it will be shown a link that takes the user to the complete form, from where you can edit or delete the item.

- *Static Filter*: as we showed before for lists, the analyst can filter the elements that are shown on the map using logical operators on the properties of the entity. In the previous section we have already explained how the value of this filter is specified. An example of this feature is shown in Figure B.24, where the analyst decides to show only the active trucks.

- *Searching*: the analyst can enables a searching engine to allow the user to perform textual searches on the elements of the map, as we can see in Figure B.25.

10.2.10 Product Preview

In the last section, the analyst can generate the source code of the resulting product using the "Download the Source Files" button that we see in the upper right. However, the analyst has also the possibility of running a mock version of the web interface directly on the browser, without actually installing or deploying anything on the server side. For example, we can see the welcome screen chosen for our example (user authentication) in Figure B.27.

Of course, not all options will be available in this mode, since it is running without server and all requests are being simulated in memory directly. That is, if we use a form to create a truck (Figure B.28), the element is stored into memory,

and the application can show it like the final application should behave (Figure B.29. We can also preview the static pages, for example, as seen in Figure B.30.

Although it is not possible to test the functionality of most components but simply its visual aspect, this preview interface component will be very useful to the analyst when designing the applications to generate, since it will allow to save a lot of time redeploying the products for changes related to the interface design.

Part IV

Summary of the thesis

11

Conclusions and Future Work

11.1 Summary

Software product lines engineering (SPLE) is a field trying to industrialize the software development. Its application, when feasible, not only improves the quality of the code but also decrements the cost both in time and in work. Geographic information systems (GIS) is a field in which there is a strong standardization and many software components can be reused as they are in different products. This is a nice scenario to pursue the benefits of SPLE. However, there are modules in GIS applications that need to be generated specifically for each product depending on some specification, which is, in exchange, a scenario more adequate for model driven engineering (MDD). Our main goal was to combine these two fields, SPLE and MDD, to produce web-based GIS applications more efficiently and with greater quality.

In order to prove that we have had the collaboration of GIS experts from a Enxenio, a software development company, and we have defined a methodology to benefit from its contribution. The first contribution of this work is the definition of a new methodology for the application of software product line engineering over a new domain. This methodology takes into account previous methodologies that combined work in synergy to complement each others, as well as specific tasks to

explicitly handle the knowledge from experts.

Using this methodology, we have designed a software product line for web-based geographic information systems, which includes itself two other contributions:

- We have identified a complete and exhaustive set of features for generic web-based geographic information systems, taking into account different existing products.
- We have defined an architecture for web-based geographic information systems based on an architecture of reference, specified in GIS standards, and enhanced with features from architectures of existing products.

Our last contribution is the design and the implementation of a tool, GISBuilder, following the specification extracted from our previous process. This tool has been developed with web technologies, it can be used by analyst with almost no knowledge about SPLE and the generated products can be incorporated to the software development processes without any problem. GISBuilder uses two libraries that are contributions themselves:

- A variability handler able to define, import and export feature models and that can run operations such as checking if a feature selection corresponds to a valid product.
- A derivation engine that follows the annotative approach with many advantages compared to the state of the art in SPL implementation techniques.
- A runtime preview component that allows our tool to show, in the browser, a preview of the projects before actually generating them.

11.2 Future Work

The next steps of our research are:

- Implementation of the features not implemented yet, as well as evolution and maintenance of the current components taking into account the products using them.
- Design of a methodology to confront the evolution of the products and the platform in a synchronized way. One approach in this direction is the usage of a version control system, like *git*, to support this methodology.

- Design a integrated system based on continuous integration to make the tool completely independent of the deployment, even for full-stack deployment. This system must work in sync with the evolution methodology so a development team can work on a product, move the upgrades done to the platform code and easily redeploy the product again with the new assets.
- Upgrade the variability handler with more operations from [BSRC10], since there is not any tool that provides all of them.
- Apply the same methodology on the context of a different product family.
- In the context of web engineering, it would be interesting to go beyond in the runtime live preview capability and to provide a framework for live programming the products, showing the running products on one window in the browser and the specification interface in a different one.

A

JSON Schema for our tool

In this appendix we show the JSON Schema that describes and validates the GISBuilder DSL.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "A GISBuilder Project Specification",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "features": {
      "description": "List of the selected features for the product",
      "type": "array",
      "items": {
        "description": "Feature name",
        "type": "string"
      }
    },
    "parameters": {
      "title": "Parameters",
      "description": "Component parameters and configuration",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "global": {
          "$ref": "#/definitions/global"
```

```
        },
        "gui": {
          "$ref": "#/definitions/gui"
        },
        "menus": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/menu"
          }
        },
        "dataModel": {
          "$ref": "#/definitions/dataModel"
        },
        "statics": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/staticPage"
          }
        },
        "forms": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/form"
          }
        },
        "lists": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/list"
          }
        },
        "maps": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/map"
          }
        }
      },
      "required": [
        "global",
        "dataModel",
        "gui",
        "menus",
        "statics",
        "forms",
        "lists",
        "maps"
      ]
    }
  },
  "required": [
```

```
    "features",
    "parameters"
  ],
  "definitions": {
    "global": {
      "title": "Basic project data",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "name": {
          "description": "Project name",
          "type": "string"
        },
        "version": {
          "description": "Current version",
          "type": "string"
        },
        "packageInfo": {
          "description": "Maven-like package info",
          "$ref": "#/definitions/packageInfo"
        },
        "index": {
          "description": "Welcome page of the application",
          "$ref": "#/definitions/componentView"
        },
        "languages": {
          "description": "List of the languages enabled for the project",
          "type": "array",
          "items": {
            "description": "Language name",
            "type": "string"
          }
        }
      },
      "required": [
        "name",
        "version",
        "packageInfo",
        "index",
        "languages"
      ]
    },
    "packageInfo": {
      "title": "Package info",
      "description": "Maven-like package info to identify the application",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "group": {
          "description": "Maven-style group for the project",
          "type": "string"
```

```
        },
        "artifact": {
          "description": "Maven-style artifact for the project",
          "type": "string"
        }
      },
      "required": [
        "group",
        "artifact"
      ]
    },
    "componentView": {
      "title": "Component view",
      "description": "Reference to a view of a component. It can refer an
     static view, like the authentication page, or an previously specified
     element, like a form",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "component": {
          "description": "Component name",
          "type": "string"
        },
        "view": {
          "description": "Name of the element or view",
          "type": "string"
        }
      },
      "required": [
        "component",
        "view"
      ]
    },
    "dataModel": {
      "title": "Data model",
      "description": "Specification of the data model, includying entities and
     enums",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "enums": {
          "type": "array",
          "description": "List of the enums that exist in the product",
          "items": {
            "$ref": "#/definitions/enum"
          }
        },
        "entities": {
          "description": "List of the entities to be managed by the product",
          "type": "array",
          "items": {
```

```
            "$ref": "#/definitions/entity"
          }
        }
      },
      "required": [
        "enums",
        "entities"
      ]
    },
    "enum": {
      "title": "Enum",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "name": {
          "description": "Name of the enum",
          "type": "string"
        },
        "values": {
          "description": "Different values of the enum",
          "type": "array",
          "items": {
            "type": "string"
          }
        }
      },
      "required": [
        "name",
        "values"
      ]
    },
    "entity": {
      "title": "Entity",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "name": {
          "description": "Name of the entity. Must be unique",
          "type": "string"
        },
        "properties": {
          "description": "Properties of the entity",
          "type": "array",
          "items": {
            "$ref": "#/definitions/property"
          }
        },
        "displayString": {
          "description": "String representation of an instance of the entity.
    This will be used in the final product in every ocassion where this entity
    is referred (in forms or lists). Any string can be set for this value, and
```

```
    properties of the entity can be included by naming them with the prefix $.
    For example, if the entity has the properties \"id\" and \"name\", a
    possible value for \"displayString\" may be \"entity $id: $name\", which be
    translated, for example, as \"entity 123: A name\" in the final product web
    interface",
222         "type": "string"
223       }
224     },
225     "required": [
226       "name",
227       "properties",
228       "displayString"
229     ]
230   },
231   "property": {
232     "title": "Property",
233     "type": "object",
234     "additionalProperties": false,
235     "properties": {
236       "name": {
237         "description": "Name of the property. Must be unique within the
    entity",
238         "type": "string"
239       },
240       "class": {
241         "description": "Class of the property. It can be one of the
    following: \n* **Basic types**: Boolean, Long, Long (autoinc), Integer,
    Double, Decimal, String, Date, DateTime \n* If GIS is enabled, **geographic
    types**: Poing, MultiPolygon,... \n* Any other previously defined
    **entity** \n* A previously defined **enum** \n",
242         "type": "string"
243       },
244       "pk": {
245         "description": "True for the property that is the primary key of the
    entity. Only Long, Integer or String properties can be primary key, and it
    implies *unique* and *not multiple*",
246         "type": "boolean",
247         "default": false
248       },
249       "required": {
250         "description": "True if the property must not be null. It implies
    *not multiple*",
251         "type": "boolean",
252         "default": false
253       },
254       "multiple": {
255         "description": "True in case this property is a collection. Otherwise
    it will be a single element. It implies *not unique*",
256         "type": "boolean",
257         "default": false
258       },
```

```
259         "unique": {
260           "description": "True if the value of the property cannot be repeated
    for any other instance of the entity",
261           "type": "boolean",
262           "default": false
263         },
264         "default": {
265           "description": "Default value of the property. It only can be set on
    basic types and enums",
266           "type": "string"
267         },
268         "bidirectional": {
269           "description": "In case the property type is another entity, the
    relation can be bidirectional, being this value the name of the property
    that matches the relationship in the other entity. The other property must
    be specified as well, and its \"bidirectional\" value must be accordingly
    set",
270           "type": "string"
271         }
272       },
273       "required": [
274         "name",
275         "class"
276       ]
277     },
278     "gui": {
279       "title": "GUI",
280       "description": "Parametrization on the graphic interface of the product.
    This section is very flexible depending on each one of the existing
    designs. The number and position of menus is related to the design chosen,
    but the menu configuration is done in the \"menus\" section",
281       "type": "object",
282       "additionalProperties": false,
283       "properties": {
284         "design": {
285           "description": "The chosen design among the existing ones",
286           "type": "integer"
287         },
288         "settings": {
289           "title": "Design Settings",
290           "description": "Configuration for the design",
291           "type": "object",
292           "additionalProperties": false,
293           "properties": {
294             "font": {
295               "title": "Font Settings",
296               "description": "Parametrization on the font style of the
    application",
297               "type": "object",
298               "additionalProperties": false,
299               "properties": {
```

```
300                 "family": {
301                   "description": "Family of the font to use",
302                   "type": "string"
303                 },
304                 "size": {
305                   "description": "Size of the fonts within the application. All
    the font sizes are related to this configuration, meaning not every one
    will have exactly this size but a size relative to it",
306                   "type": "string"
307                 }
308               }
309             },
310             "colorset": {
311               "description": "Set of colors used in the style of the
    application",
312               "type": "string"
313             },
314             "authenticationOnMenu": {
315               "description": "If user management is active, whether there is
    user session related info on the menu",
316               "type": "boolean"
317             },
318             "header": {
319               "title": "Header settings",
320               "description": "Parametrization of the header of the product",
321               "additionalProperties": false,
322               "properties": {
323                 "type": {
324                   "description": "If the header is just a text or if it is an
    image",
325                   "type": "string",
326                   "pattern": "TEXT|IMAGE"
327                 },
328                 "text": {
329                   "description": "Default text of the header, and identifier of
    the language messages",
330                   "type": "string"
331                 },
332                 "image": {
333                   "description": "URI of the image of the header",
334                   "type": "string"
335                 }
336               }
337             }
338           }
339         }
340       },
341       "required": [
342         "design",
343         "settings",
344         "header"
```

```
      ]
    },
    "menu": {
      "title": "Menus",
      "description": "Configuration of the initial menus of the application",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "id": {
          "description": "Identifier of the menu to configure. It must be one of the existing in the design",
          "type": "string"
        },
        "elements": {
          "description": "Configuration of the items of the menu",
          "type": "array",
          "items": {
            "$ref": "#/definitions/menuItem"
          }
        }
      },
      "required": [
        "id",
        "elements"
      ]
    },
    "menuItem": {
      "title": "Menu item",
      "oneOf": [
        {
          "$ref": "#/definitions/viewMenuItem"
        },
        {
          "$ref": "#/definitions/urlMenuItem"
        },
        {
          "$ref": "#/definitions/menuMenuItem"
        }
      ]
    },
    "menuAccess": {
      "title": "Menu element access control",
      "description": "If the user manager is enabled, using this object the access to the different elements of the menu can be restricted",
      "additionalProperties": false,
      "properties": {
        "admin": {
          "description": "Whether the menu element is shown to admin users",
          "type": "boolean",
          "default": true
        },
```

```
394         "logged": {
395           "description": "Whether the menu element is shown to logged users",
396           "type": "boolean",
397           "default": true
398         },
399         "anonymous": {
400           "description": "Whether the menu element is shown to not logged
    users",
401           "type": "boolean",
402           "default": true
403         }
404       },
405       "required": [
406         "admin",
407         "logged",
408         "anonymous"
409       ]
410     },
411     "viewMenuItem": {
412       "title": "View",
413       "description": "This menu item links a view of the existing componentes",
414       "type": "object",
415       "additionalProperties": false,
416       "properties": {
417         "order": {
418           "description": "Order of the item within the current menu",
419           "type": "integer",
420           "minimum": 1
421         },
422         "id": {
423           "description": "Identifier of the menu item",
424           "type": "string"
425         },
426         "type": {
427           "type": "string",
428           "description": "Type of the menu",
429           "pattern": "VIEW"
430         },
431         "view": {
432           "description": "View to link",
433           "$ref": "#/definitions/componentView"
434         },
435         "access": {
436           "description": "Element access from users",
437           "$ref": "#/definitions/menuAccess"
438         }
439       },
440       "required": [
441         "order",
442         "id",
443         "type",
```

```
        "view"
      ]
    },
    "urlMenuItem": {
      "title": "Link",
      "description": "This menu item links an URL",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "order": {
          "description": "Order of the item within the current menu",
          "type": "integer",
          "minimum": 1
        },
        "id": {
          "description": "Identifier of the menu item",
          "type": "string"
        },
        "type": {
          "type": "string",
          "description": "Type of the menu",
          "pattern": "URL"
        },
        "url": {
          "description": "URL to link",
          "type": "string"
        },
        "access": {
          "description": "Element access from users",
          "$ref": "#/definitions/menuAccess"
        }
      },
      "required": [
        "order",
        "id",
        "type",
        "url"
      ]
    },
    "menuMenuItem": {
      "title": "Submenu",
      "description": "This menu is itself another menu",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "order": {
          "description": "Order of the item within the current menu",
          "type": "integer",
          "minimum": 1
        },
        "id": {
```

```
495           "description": "Identifier of the menu item",
496           "type": "string"
497         },
498         "type": {
499           "type": "string",
500           "description": "Type of the menu",
501           "pattern": "MENU"
502         },
503         "elements": {
504           "description": "Elements of the submenu",
505           "type": "array",
506           "items": {
507             "$ref": "#/definitions/menuItem"
508           }
509         },
510         "access": {
511           "description": "Element access from users",
512           "$ref": "#/definitions/menuAccess"
513         }
514       },
515       "required": [
516         "order",
517         "id",
518         "type",
519         "elements"
520       ]
521     },
522     "staticPage": {
523       "title": "Static page",
524       "description": "A static page, which is an HTML that can be link from the
      menu",
525       "type": "object",
526       "additionalProperties": false,
527       "properties": {
528         "id": {
529           "description": "Identifier of the page",
530           "type": "string"
531         },
532         "html": {
533           "description": "HTML code of the page",
534           "type": "string"
535         }
536       },
537       "required": [
538         "id",
539         "html"
540       ]
541     },
542     "form": {
543       "title": "Form",
544       "type": "object",
```

```
545       "description": "Specification of a form",
546       "additionalProperties": false,
547       "properties": {
548         "id": {
549           "description": "Identifier of the form",
550           "type": "string"
551         },
552         "entity": {
553           "description": "Entity related",
554           "type": "string"
555         },
556         "creatable": {
557           "description": "Whether this form can be used to create new items",
558           "type": "boolean",
559           "default": true
560         },
561         "editable": {
562           "description": "Whether this form can be used to edit existing items",
563           "type": "boolean",
564           "default": true
565         },
566         "removable": {
567           "description": "Whether the item viewed with this form can be
    removed",
568           "type": "boolean",
569           "default": true
570         },
571         "confirmation": {
572           "description": "True if a confirmation warning must be shown before
    removing the element",
573           "type": "boolean",
574           "default": false
575         },
576         "properties": {
577           "description": "List of the properties that appear on the form",
578           "type": "array",
579           "items": {
580             "$ref": "#/definitions/formProperty"
581           }
582         }
583       },
584       "required": [
585         "id",
586         "entity",
587         "creatable",
588         "editable",
589         "removable",
590         "properties"
591       ]
592     },
593     "formProperty": {
```

```
      "title": "Form property",
      "description": "Each property of the entity shown in the form. For each one, it can be selected to allow its viewing and/or its edition/creation",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "property": {
          "description": "The name of the property",
          "type": "string"
        },
        "viewing": {
          "description": "Whether if the property is viewable",
          "type": "boolean",
          "default": true
        },
        "editing": {
          "description": "Whether if the property is editable and creatable, in case the form is",
          "type": "boolean",
          "default": true
        }
      },
      "required": [
        "property",
        "viewing",
        "editing"
      ]
    },
    "list": {
      "title": "List",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "id": {
          "description": "Identifier of the list",
          "type": "string"
        },
        "entity": {
          "description": "Entity related",
          "type": "string"
        },
        "form": {
          "description": "The form that is linked to the list for viewing, creating, editing or removing elements",
          "type": "string"
        },
        "removeLink": {
          "description": "If true, then a link to remove each element appears in the list",
          "type": "boolean",
          "default": false
```

```
641         },
642         "staticFilter": {
643           "description": "The analyst can chose to apply static filters
    affecting all the data listed. For example, the value for a enum property
    can be set so only the elements satisfying that condition are shown",
644           "type": "string"
645         },
646         "sorting": {
647           "description": "True if the user can sort the list by column",
648           "type": "boolean",
649           "default": false
650         },
651         "filtering": {
652           "description": "True if the user can apply filters to the columns of
    the list",
653           "type": "boolean",
654           "default": false
655         },
656         "searching": {
657           "description": "True if the user can do simple searches over the
    listed data",
658           "type": "boolean",
659           "default": false
660         },
661         "properties": {
662           "description": "List of the properties that are shown on the list",
663           "type": "array",
664           "items": {
665             "$ref": "#/definitions/listColumn"
666           }
667         }
668       },
669       "required": [
670         "id",
671         "entity",
672         "form",
673         "properties"
674       ]
675     },
676     "listColumn": {
677       "title": "Column",
678       "type": "object",
679       "additionalProperties": false,
680       "properties": {
681         "property": {
682           "description": "Name of the property",
683           "type": "string"
684         },
685         "form": {
686           "description": "If the property is an entity, a form can be set and
    it will appear as a link",
```

```
687           "type": [
688             "string",
689             "null"
690           ]
691         }
692       },
693       "required": [
694         "property"
695       ]
696     },
697     "map": {
698       "title": "Map",
699       "type": "object",
700       "additionalProperties": false,
701       "properties": {
702         "id": {
703           "description": "Identifier of the map",
704           "type": "string"
705         },
706         "entity": {
707           "description": "Entity related",
708           "type": "string"
709         },
710         "geom": {
711           "description": "Geographic property of the entity used to represent
    it in the map",
712           "type": "string"
713         },
714         "baselayer": {
715           "description": "Base layer of the map",
716           "type": "string"
717         },
718         "popupForm": {
719           "description": "A viewing form shown as a popup with a link to the
    propper form, if not null",
720           "type": "string"
721         },
722         "form": {
723           "description": "The form that is linked to the map for viewing,
    creating, editing or removing elements",
724           "type": "string"
725         },
726         "staticFilter": {
727           "description": "The analyst can chose to apply static filters
    affecting the data loaded in the map",
728           "type": "string"
729         },
730         "searching": {
731           "description": "True if the user can do simple searches over the
    listed data",
732           "type": "boolean",
```

```
733           "default": false
734         }
735       },
736       "required": [
737         "id",
738         "entity",
739         "geom",
740         "baselayer"
741       ]
742     }
743   }
744 }
```

B

GISBuilder screenshots

In this appendix we have grouped together all the screenshots mentioned in Section 10.2 to show them as clearly as possible.

Figure B.1: Global data and parametrization of the project

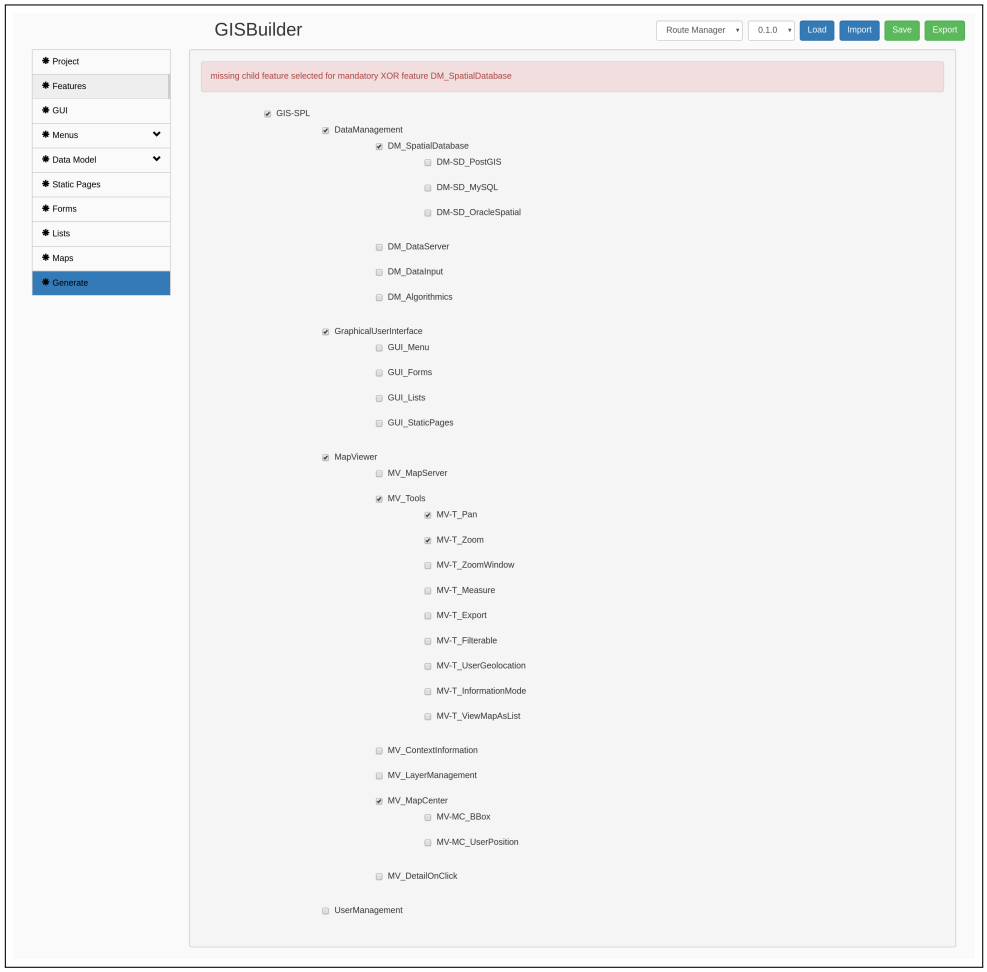


Figure B.2: Initial feature model

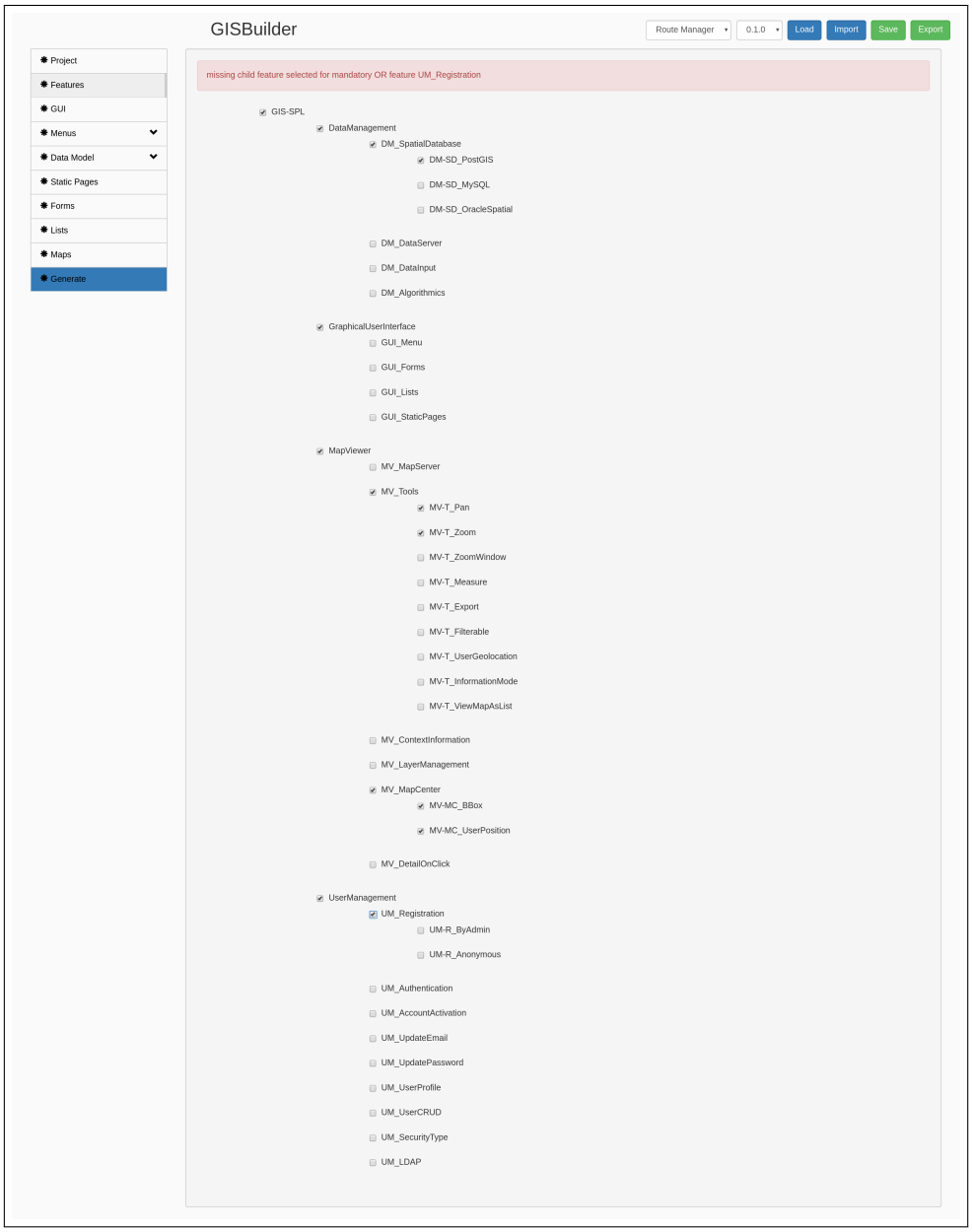


Figure B.3: Enabling user registration feature

Figure B.4: Full variability selection (I)

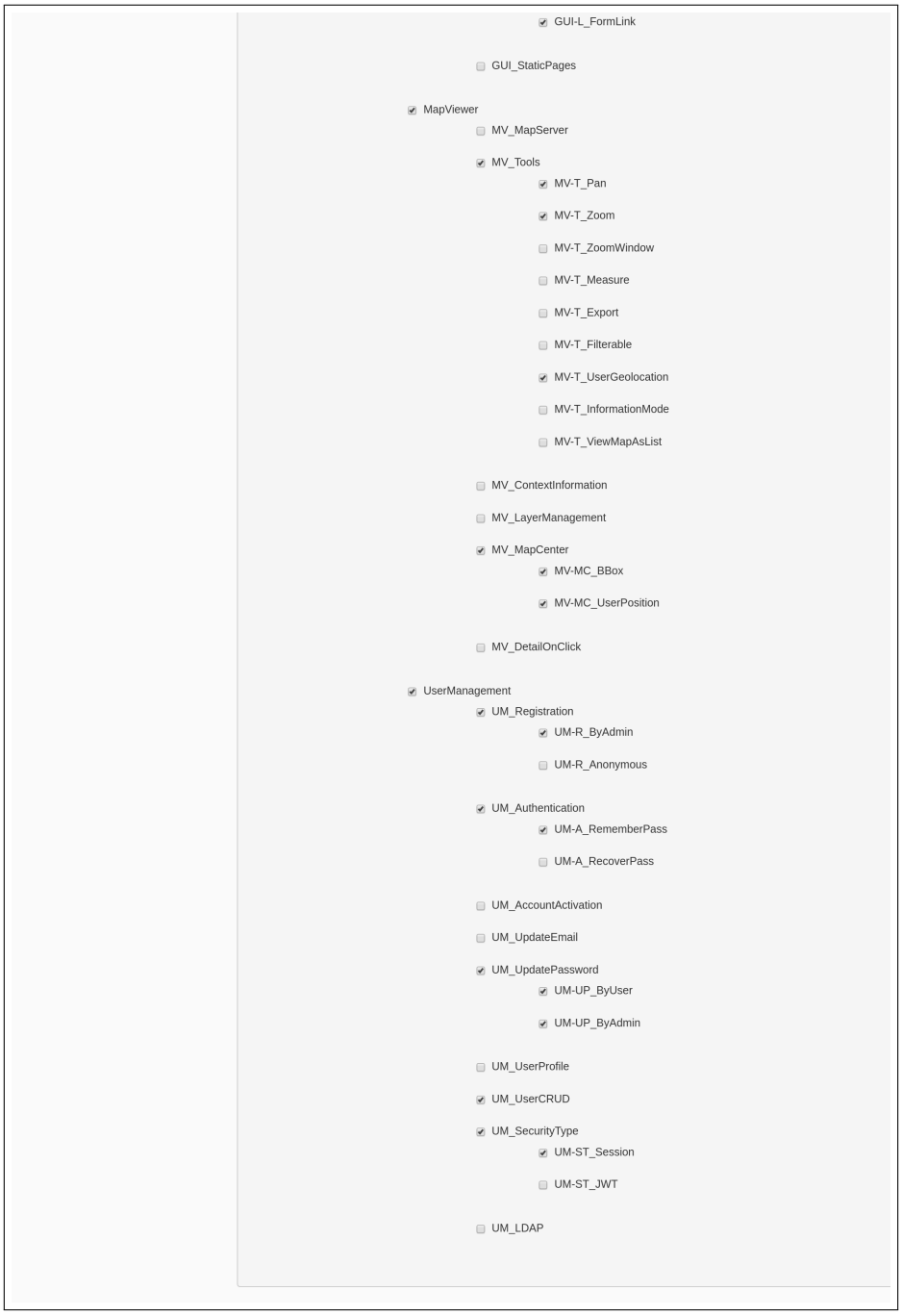


Figure B.5: Full variability selection (II)

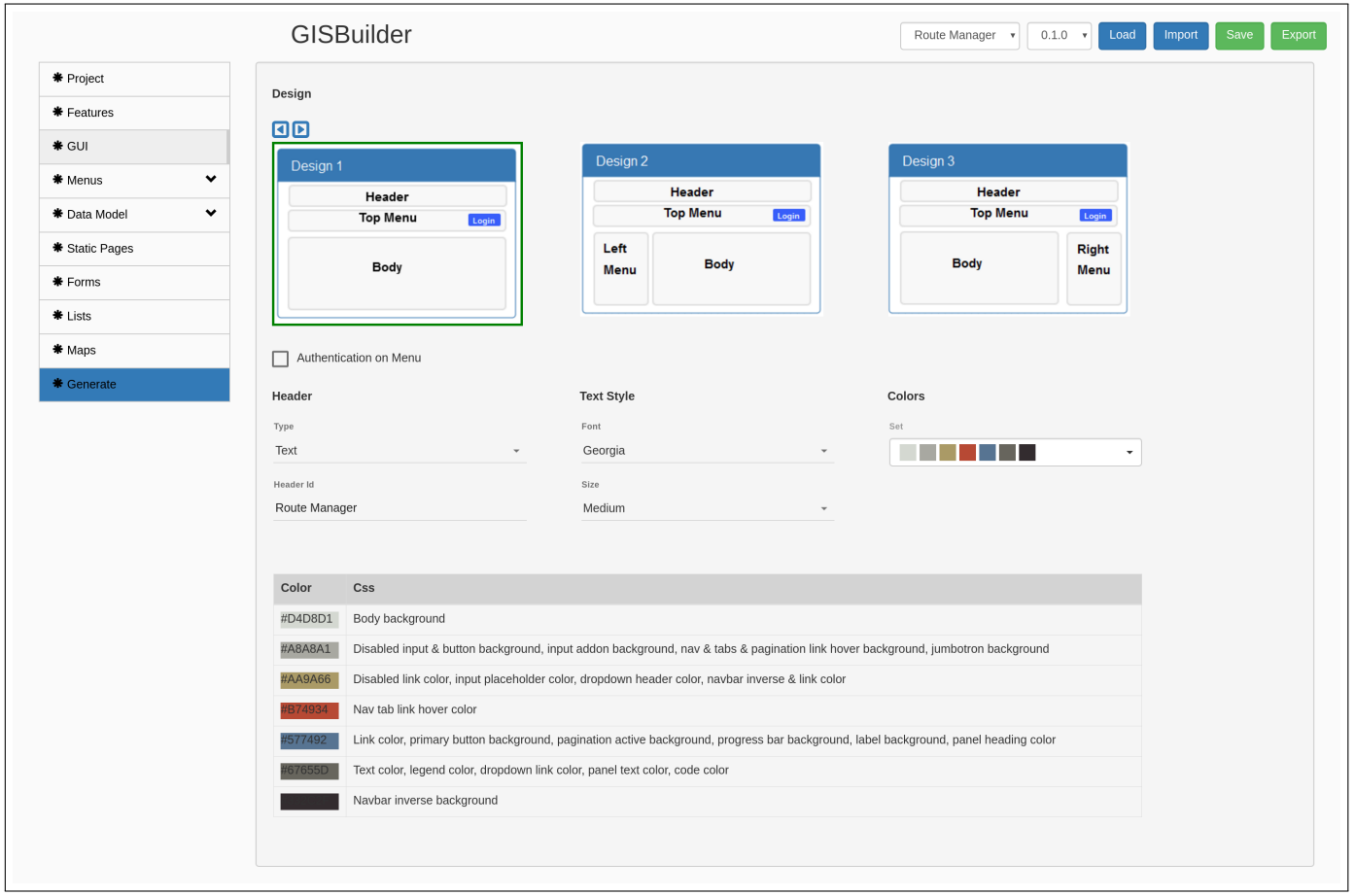


Figure B.6: Parametrization of the graphical user interface

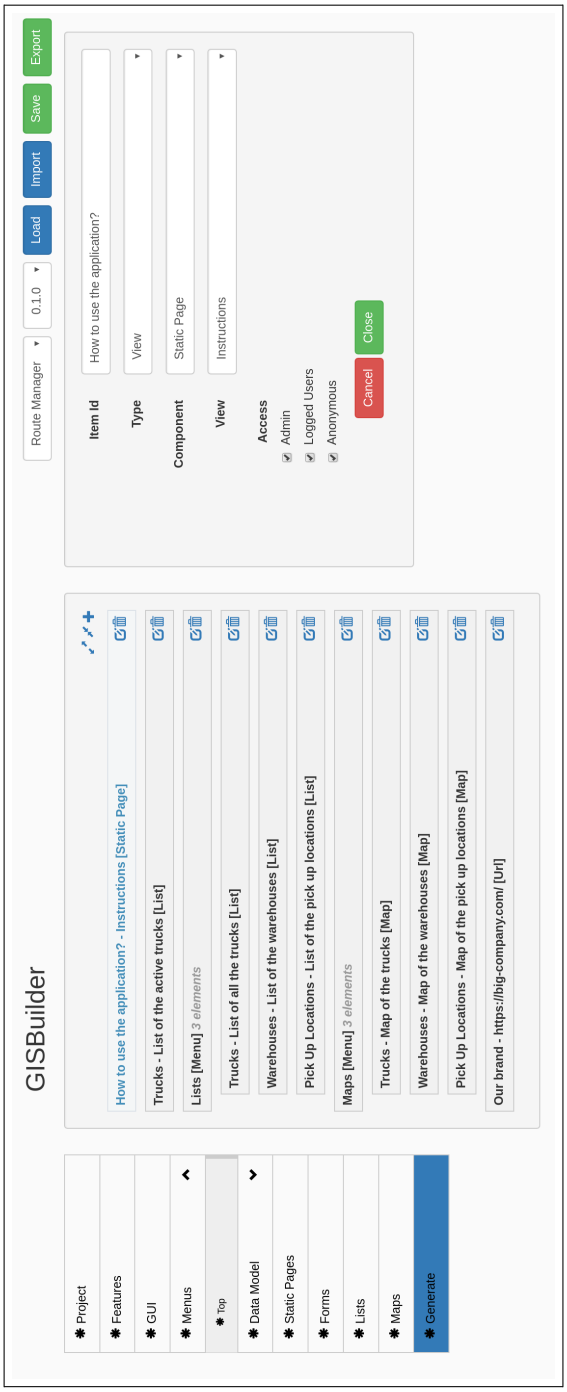


Figure B.7: Example of menu configuration - editing a *View* element

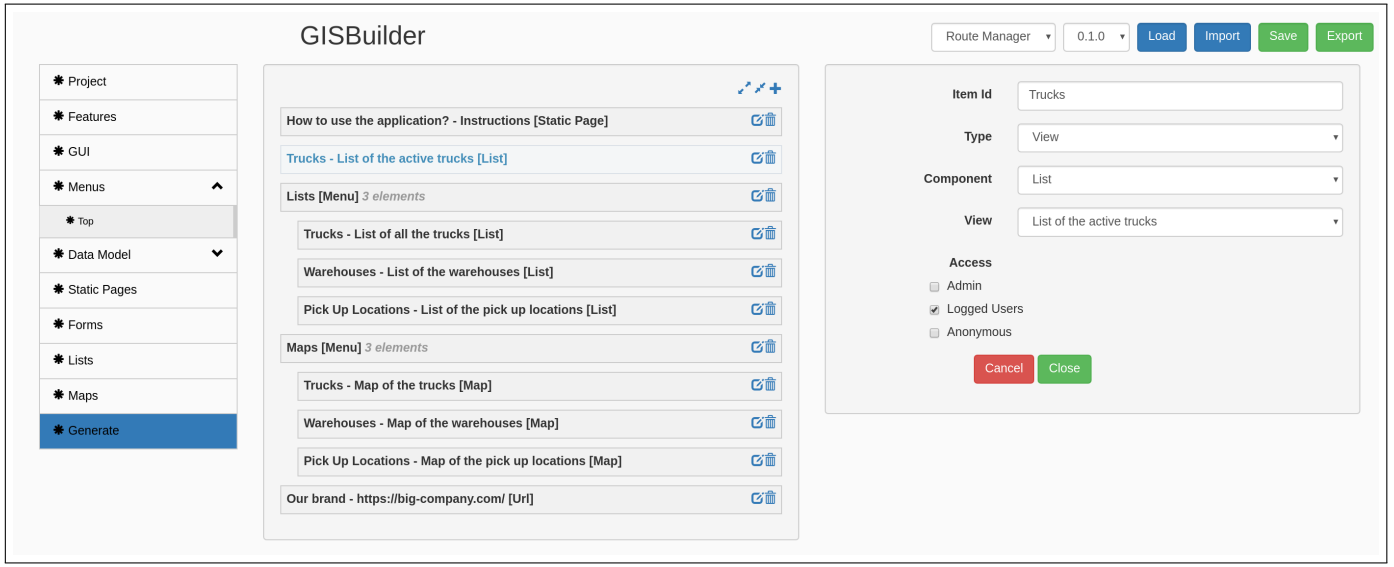


Figure B.8: Example of menu configuration - editing a *View* element with restricted access

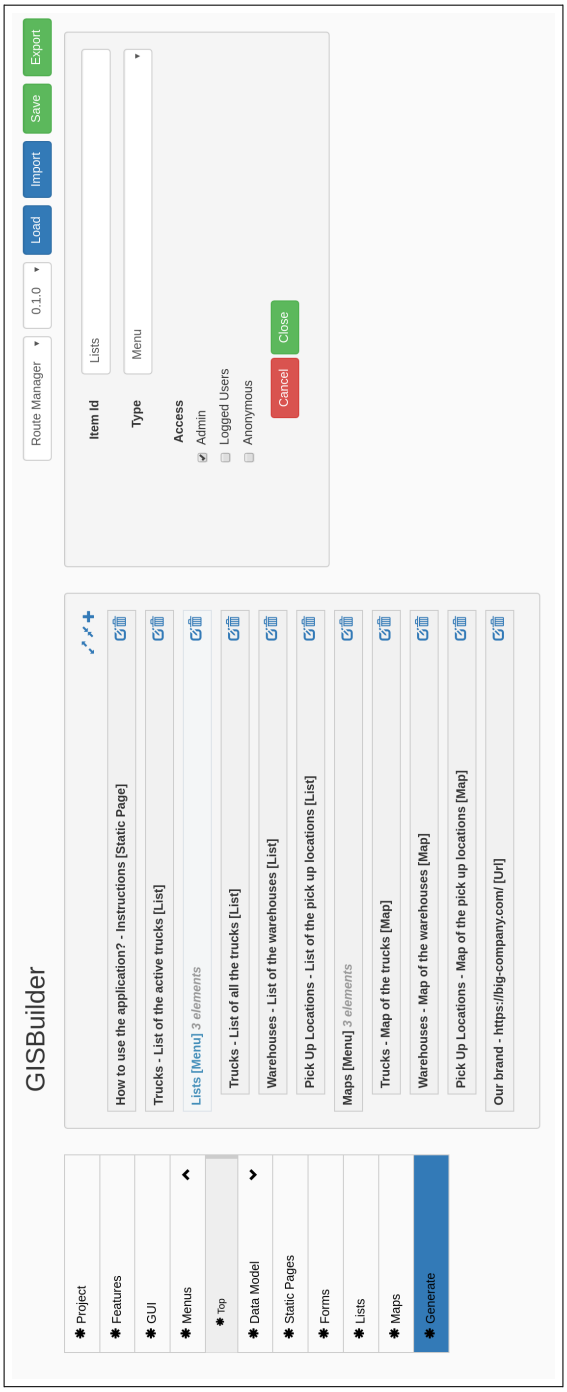


Figure B.9: Example of menu configuration - editing a *Menu* element

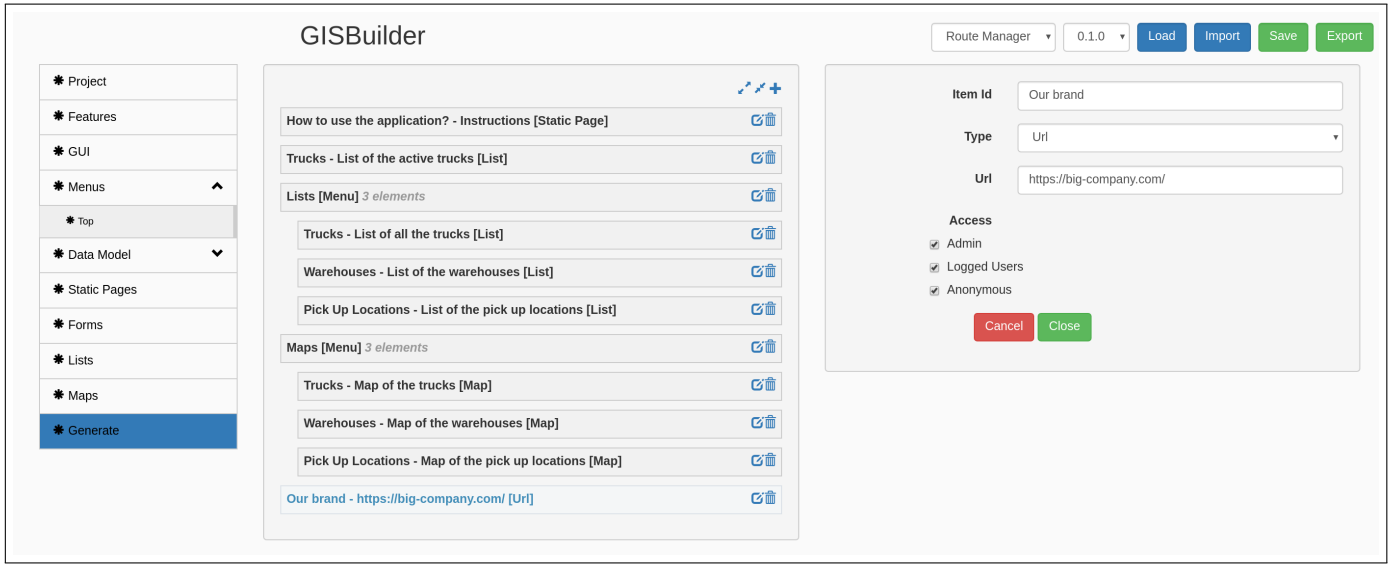


Figure B.10: Example of menu configuration - editing a *Url* element

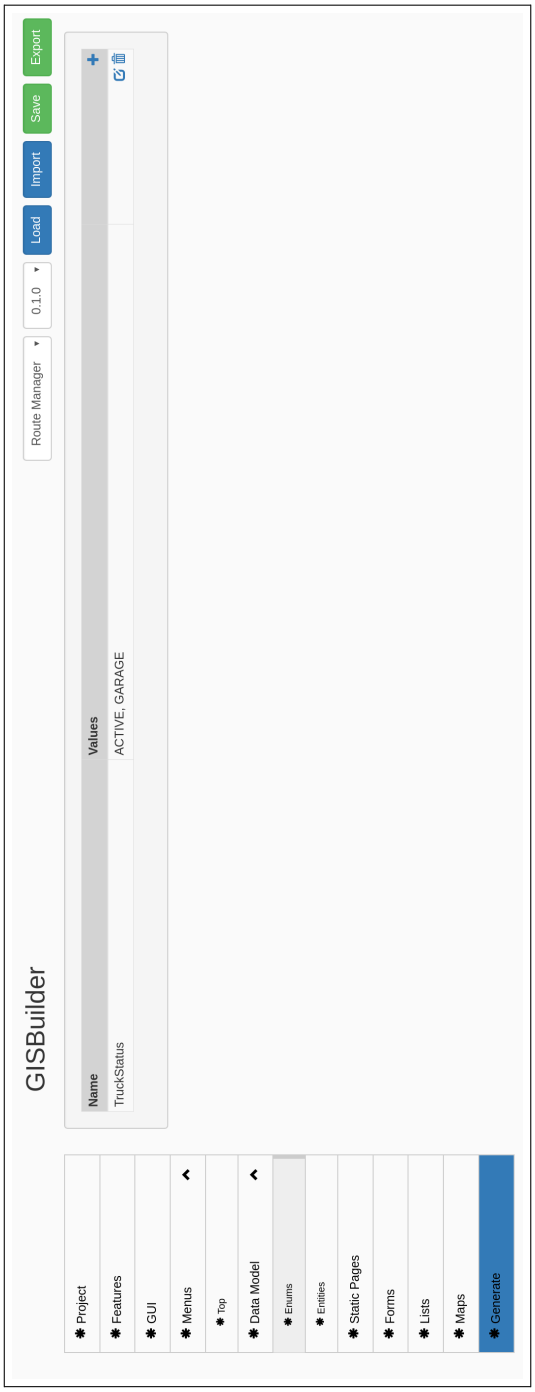


Figure B.11: Data model: section to define the enums of the application

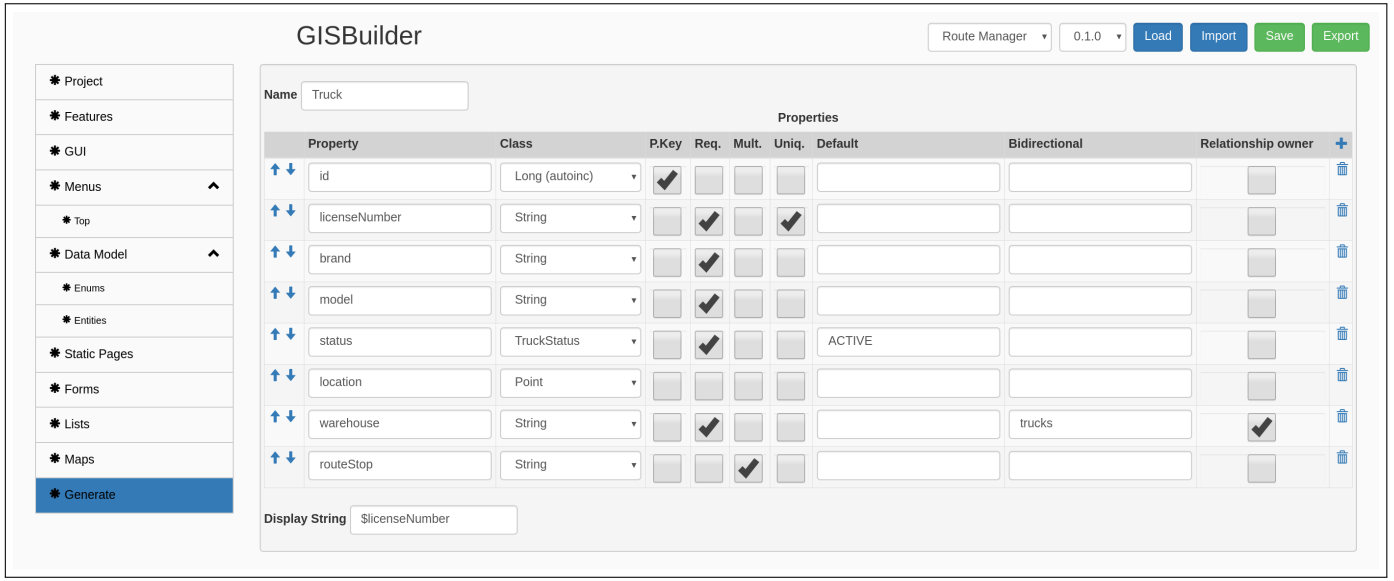


Figure B.12: Data model: *Truck* entity definition

Figure B.13: Data model: *PickUpLocation* entity definition

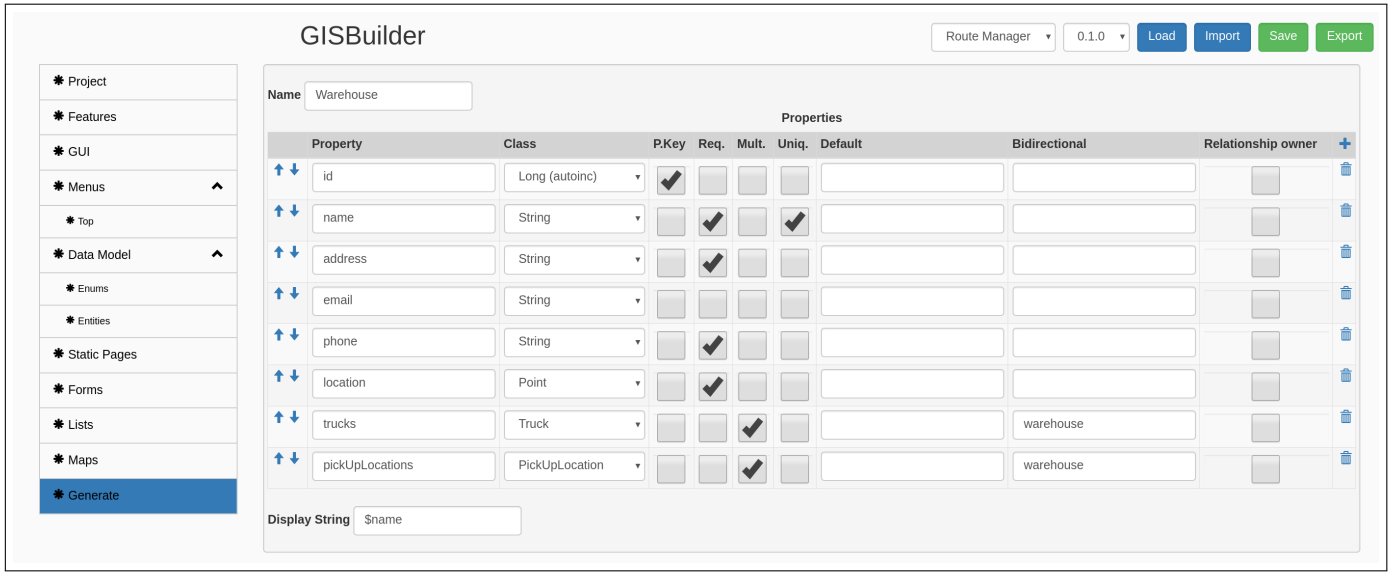


Figure B.14: Data model: *Warehouse* entity definition

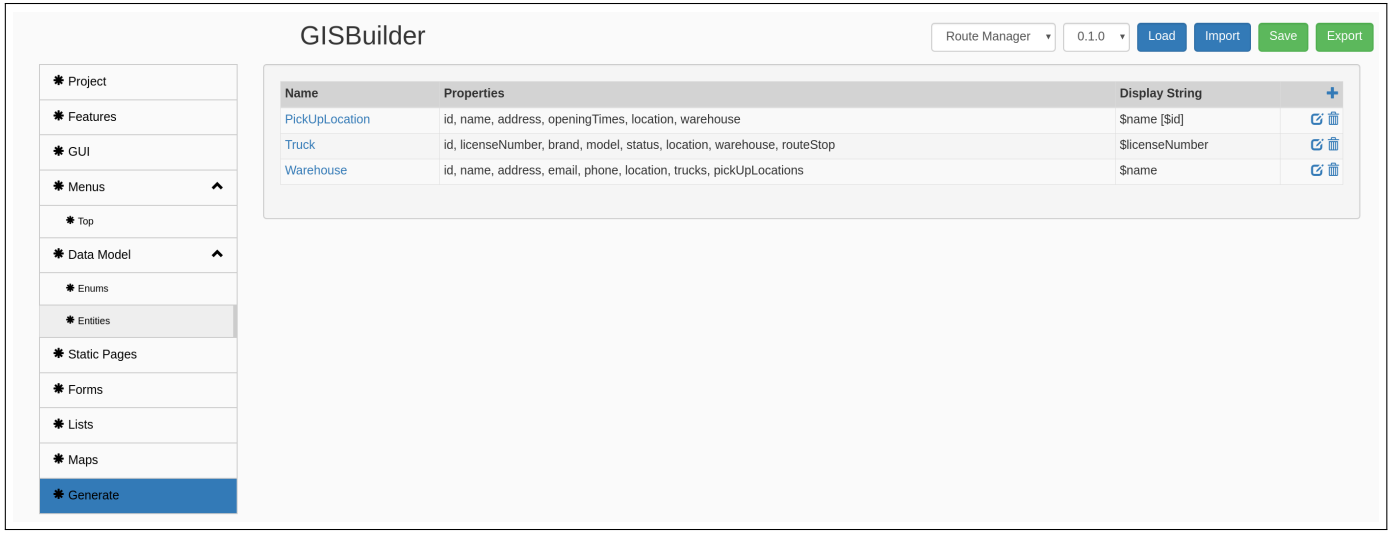


Figure B.15: Data model: a list with the three entities previously defined

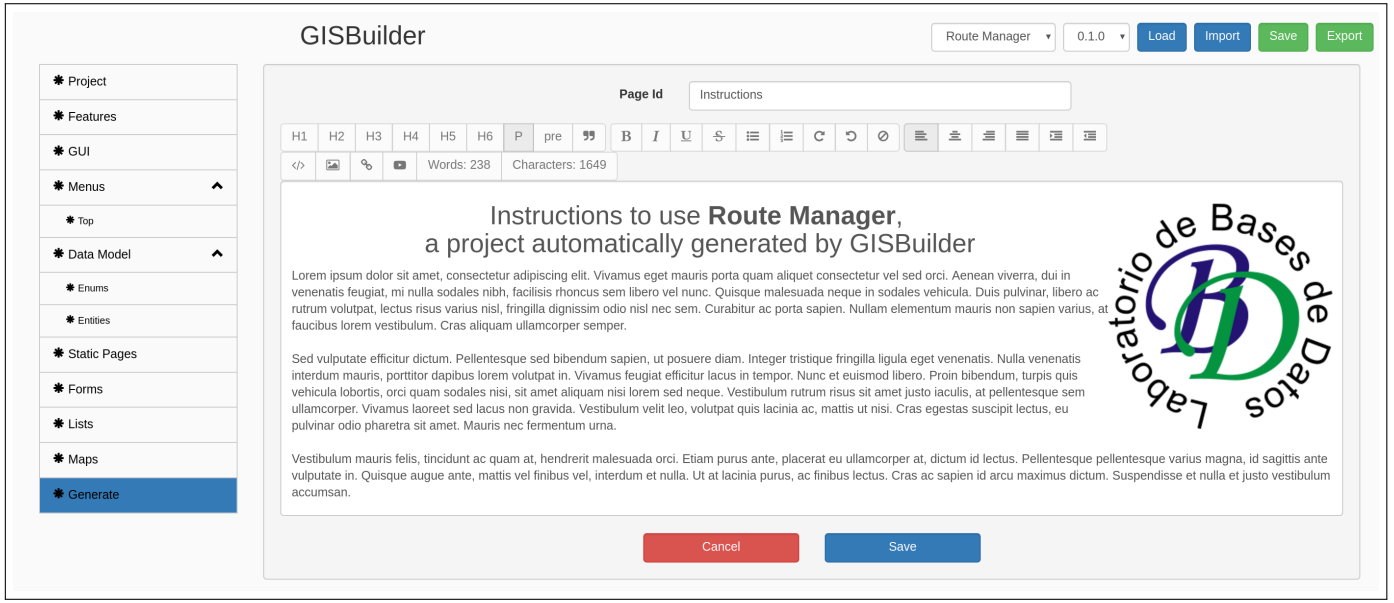


Figure B.16: Static pages WYSIWYG editor

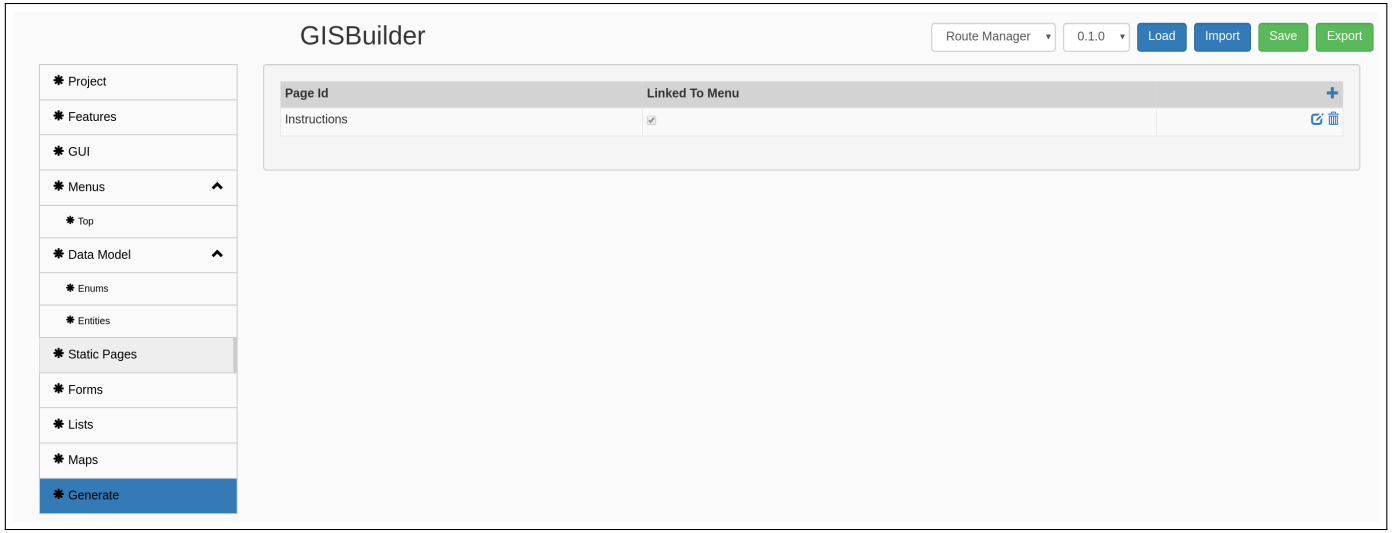


Figure B.17: List of the static pages to be created for the application

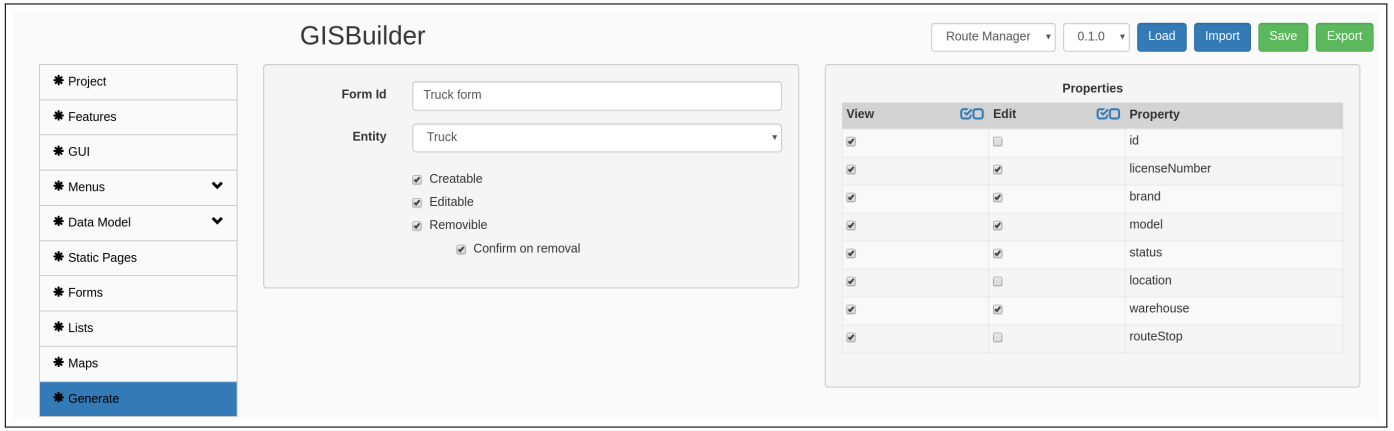


Figure B.18: Defining a form to create, edit and remove trucks

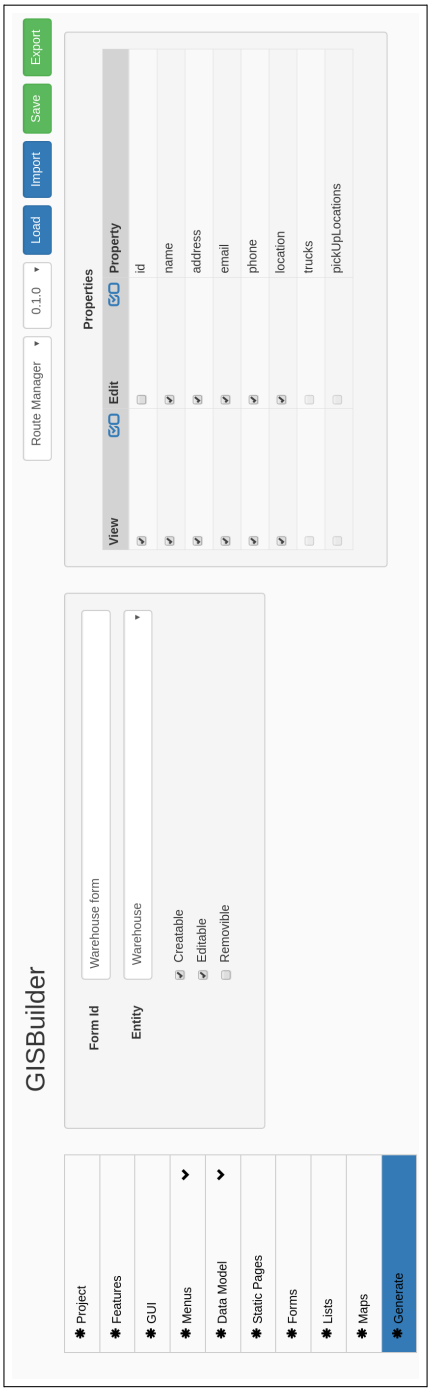


Figure B.19: Defining a form to create and edit warehouses

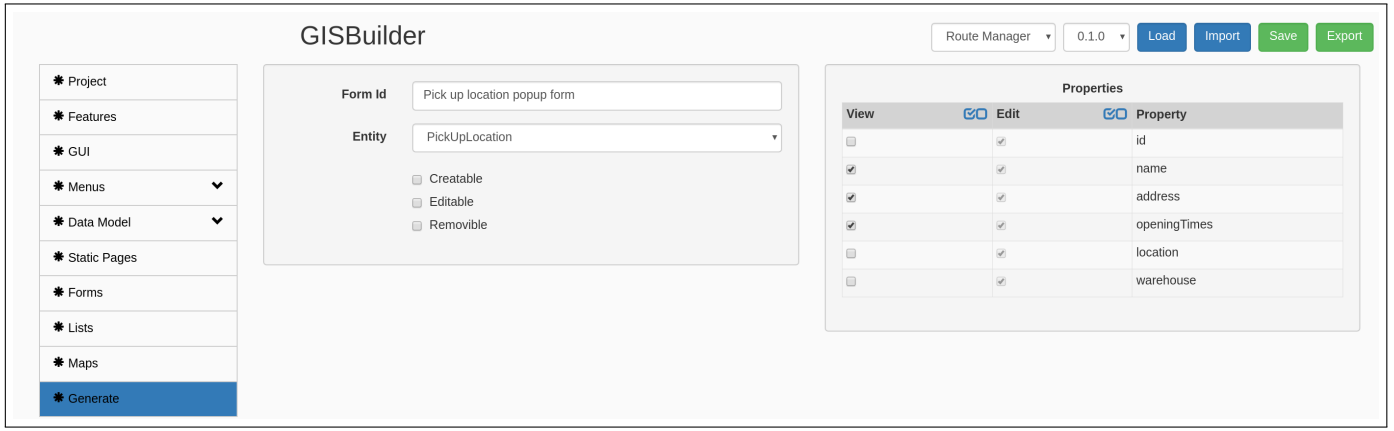


Figure B.20: Defining a simple form that only shows three properties of the pick up locations

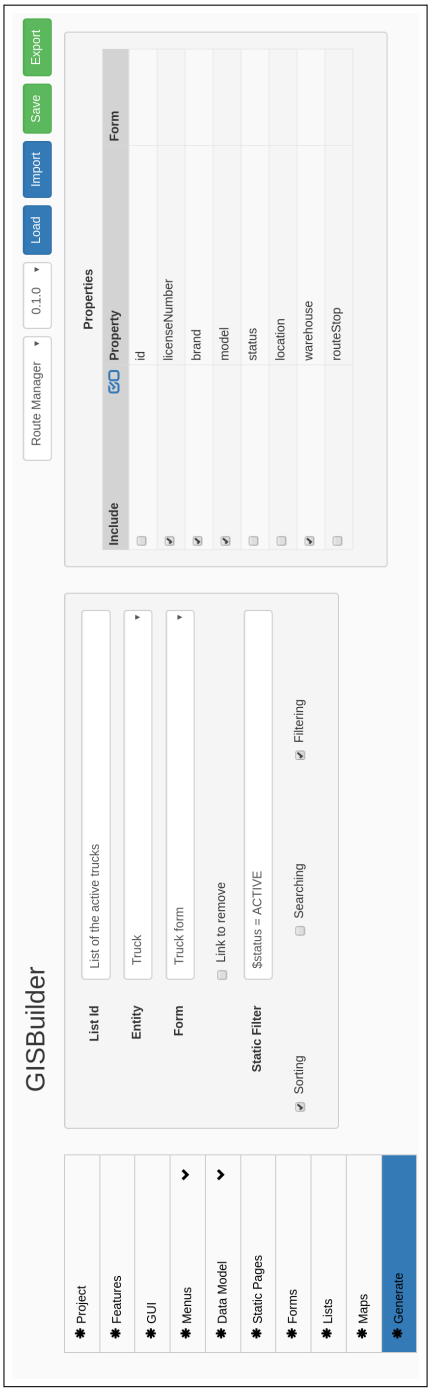


Figure B.21: Defining a list to show only the active trucks

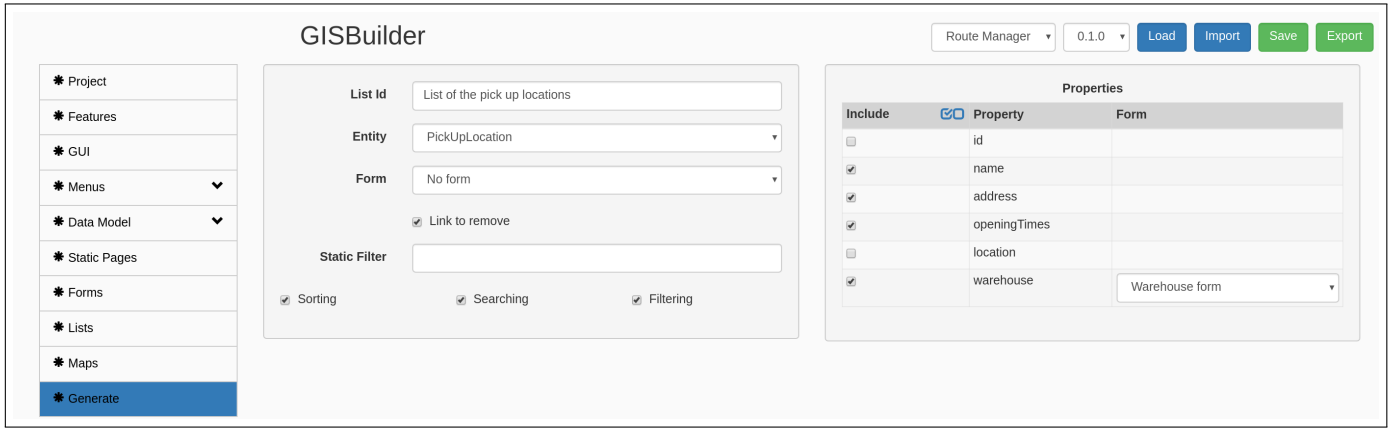


Figure B.22: Defining a list to show the pick up locations

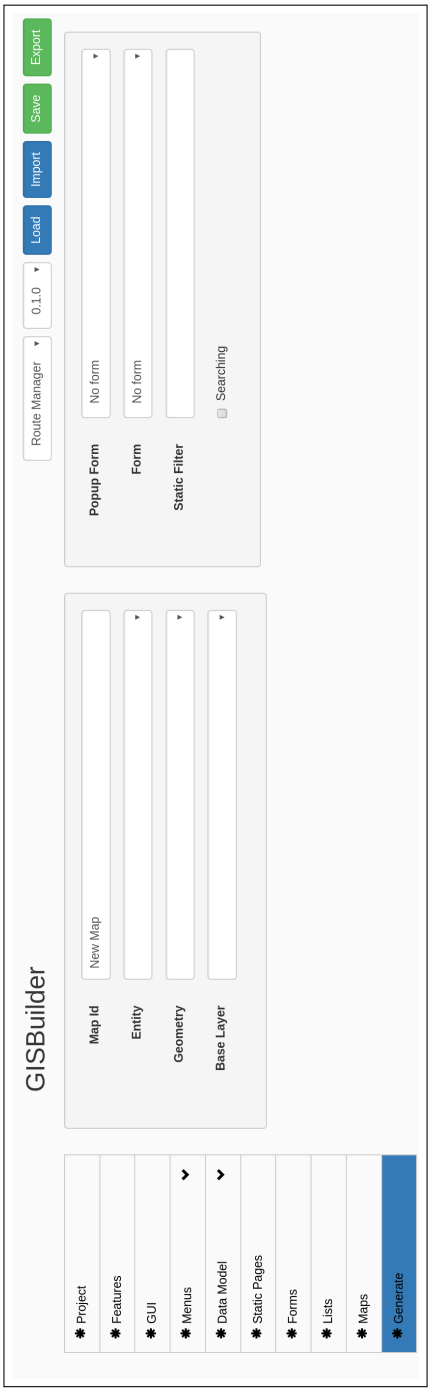


Figure B.23: Creating a new map for the application

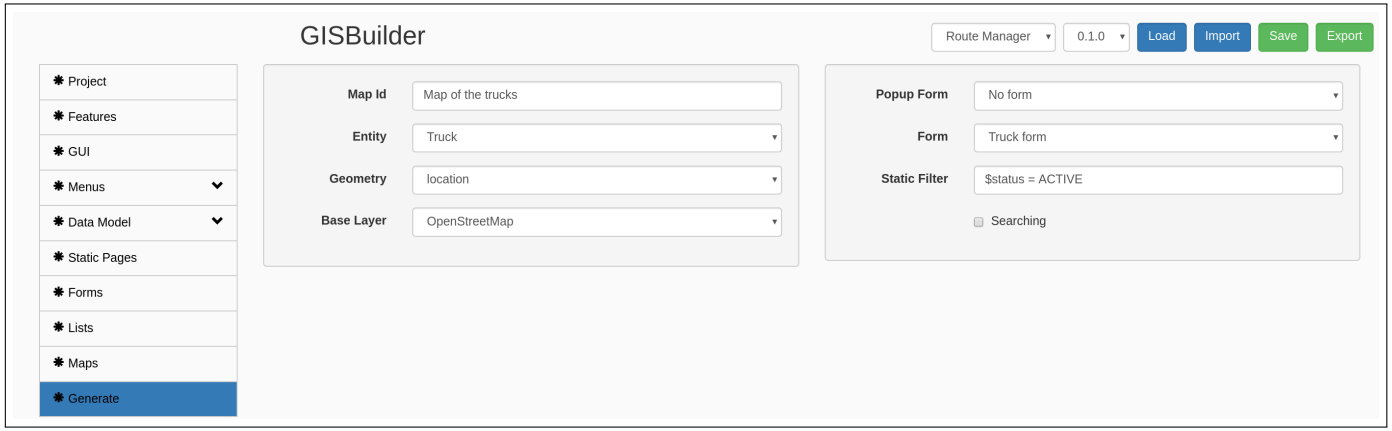


Figure B.24: Specifying a map to visualize the trucks managed by the application

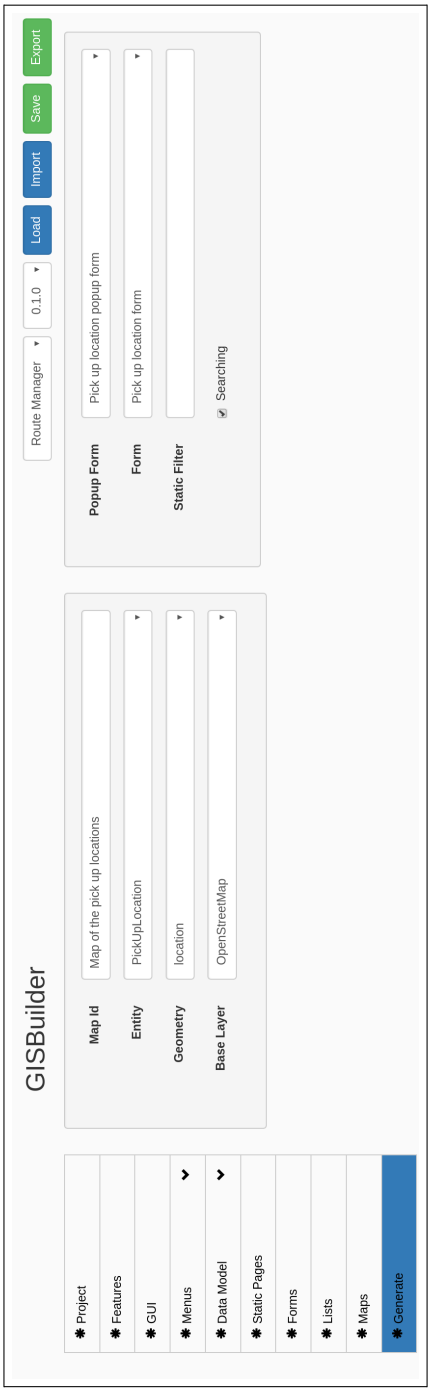


Figure B.25: Specifying a map to visualize the pick up locations managed by the application

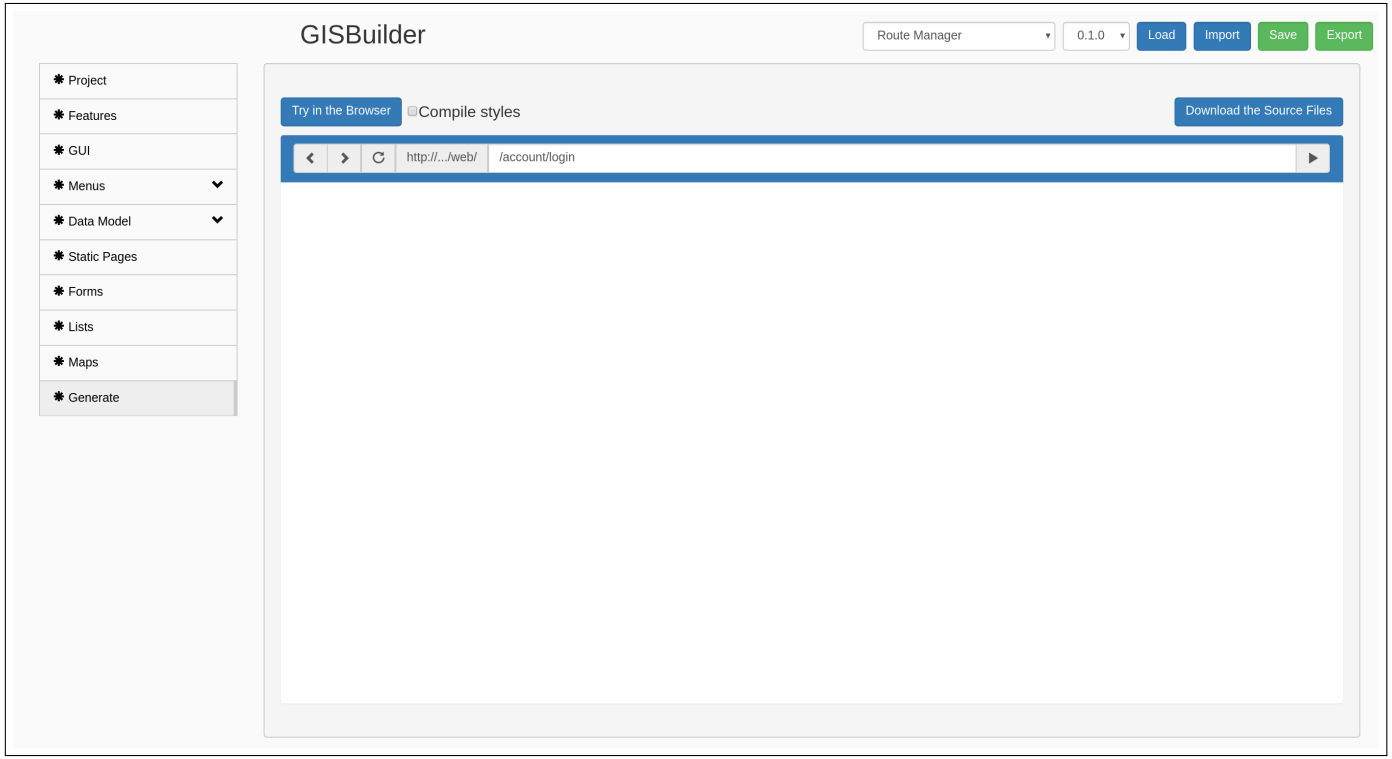


Figure B.26: Section to preview and generate the products

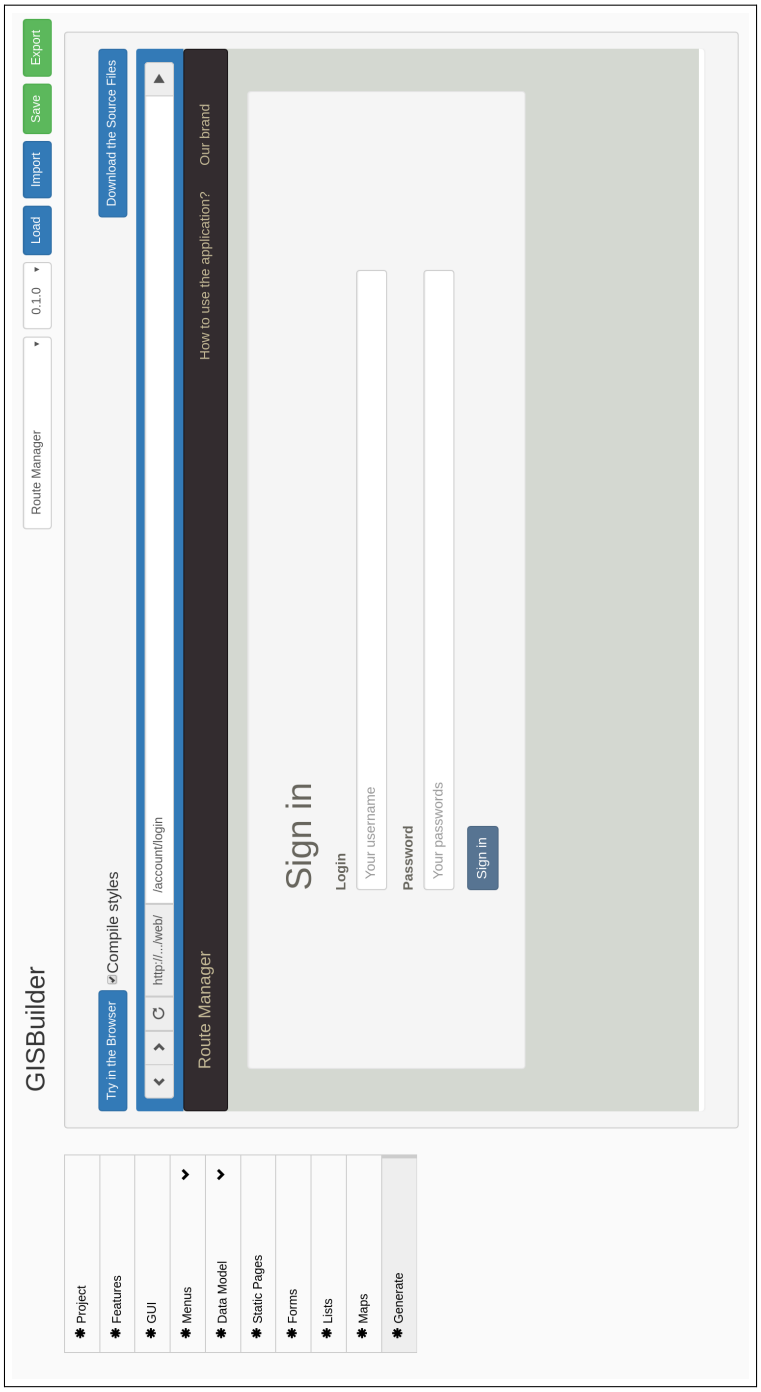


Figure B.27: Previewing a product, showing the authentication page

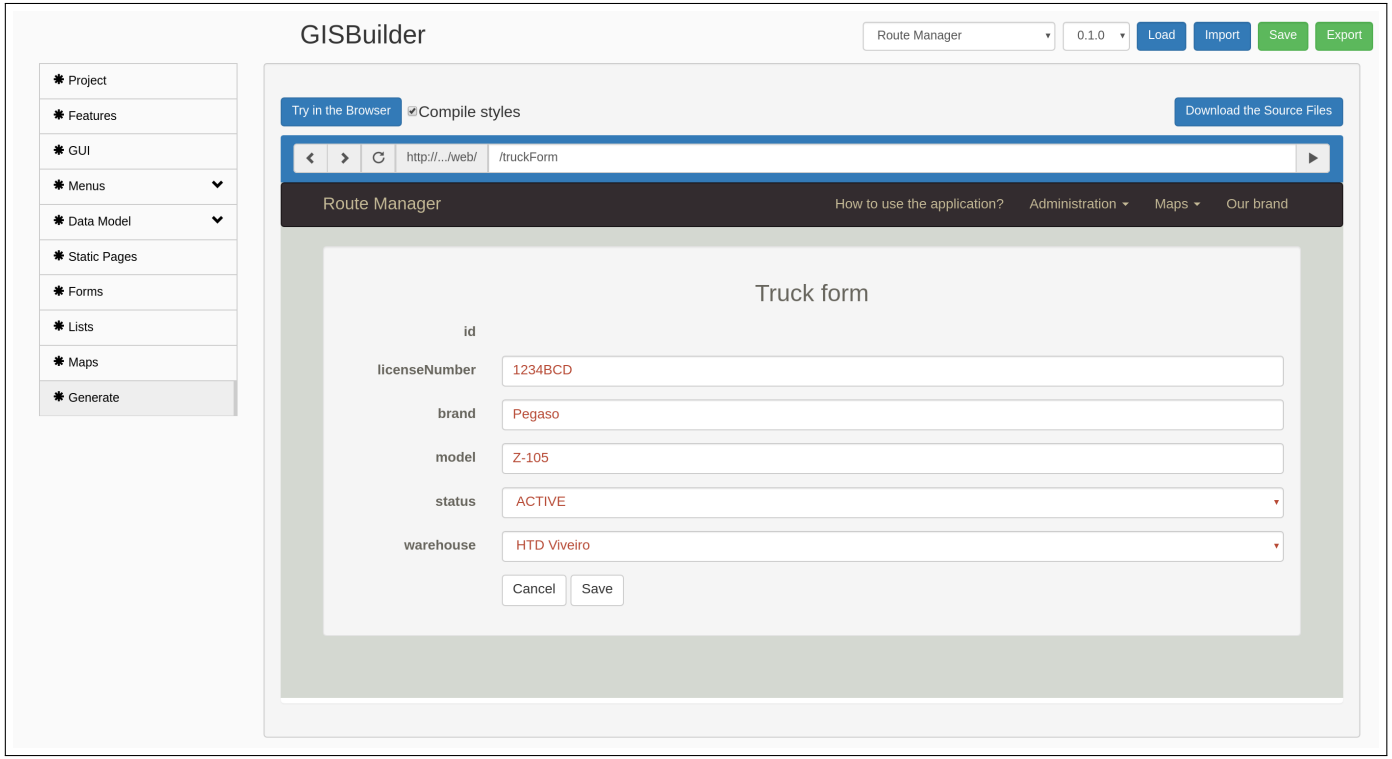


Figure B.28: Previewing a product, trying to create a new truck

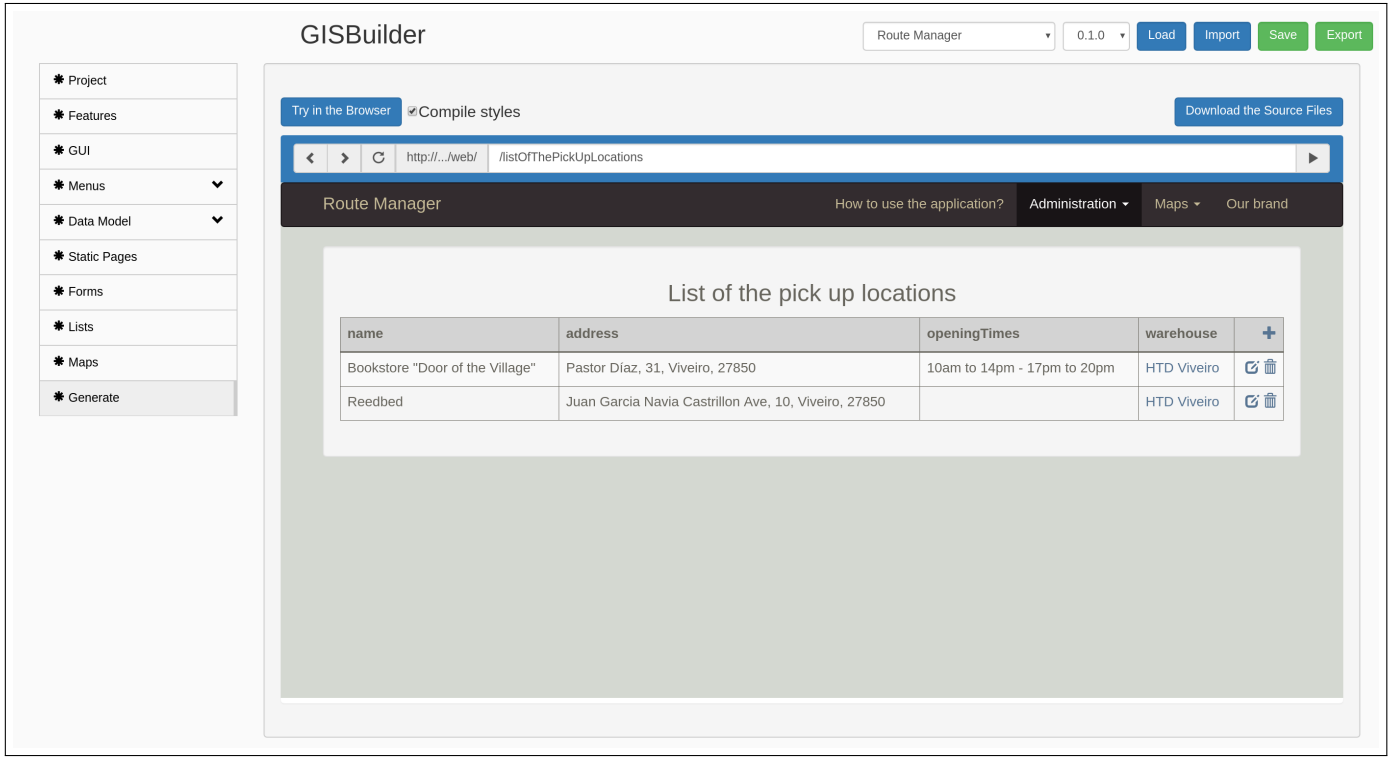


Figure B.29: Previewing a product, listing the pick up locations

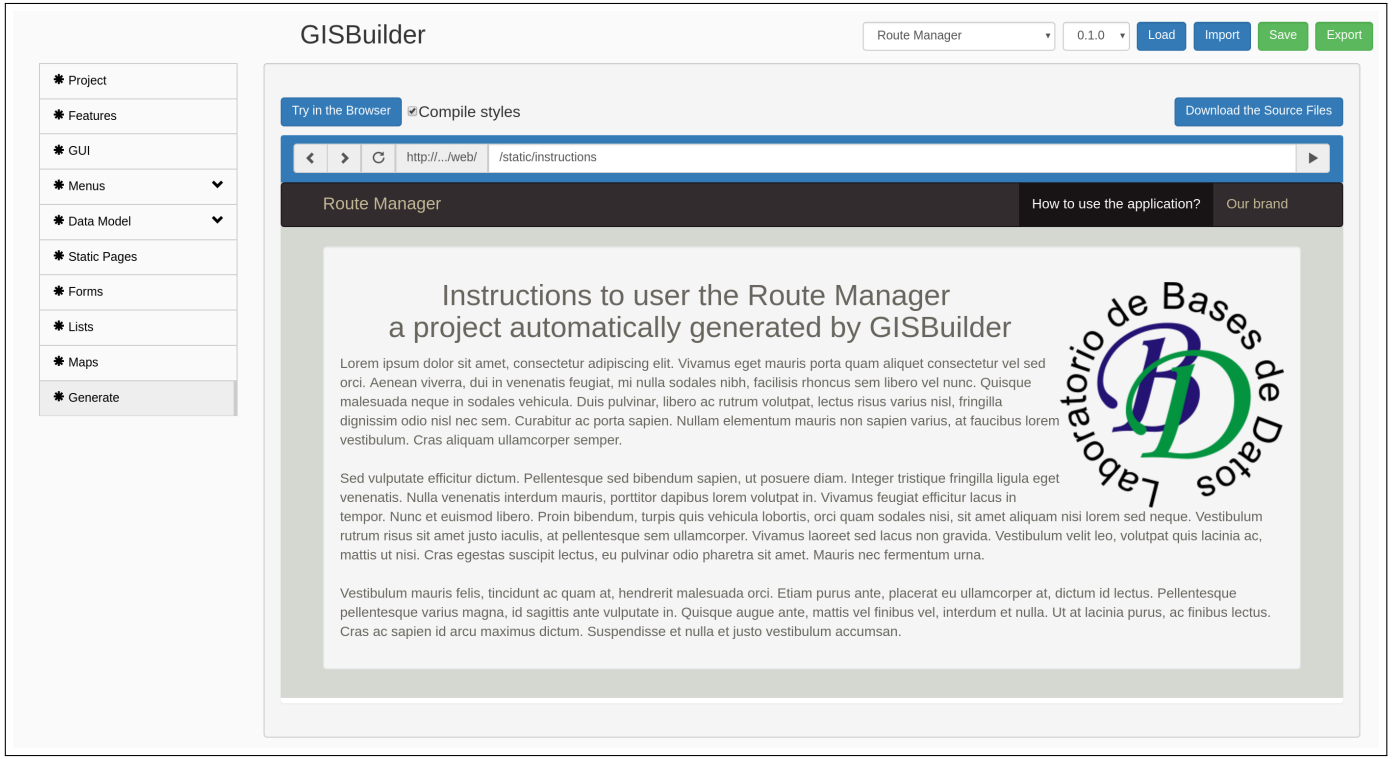


Figure B.30: Previewing a product, showing a static page

C

Publications and Other Research Results Related to the Thesis

Publications

International Conferences

- Alejandro Cortiñas, Miguel R. Luaces, Oscar Pedreira, Ángeles S. Places, and Jennifer Pérez: *Web-based Geographic Information Systems SPLE: Domain Analysis and Experience Report.* In Proceedings of the 25th International Systems and Software Product Line Conference (SPLC 2017), ACM, Sevilla (Spain), 2017. Pending publication.

- Alejandro Cortiñas, Miguel R. Luaces, Oscar Pedreira, and Ángeles S. Places: *Scaffolding and in-browser generation of web-based GIS applications in a SPL tool.* In Proceedings of the 25th International Systems and Software Product Line Conference (SPLC 2017), ACM, Sevilla (Spain), 2017. Pending publication.

- Alejandro Cortiñas, Carlo Bernaschina, Miguel R. Luaces and Piero Fraternali: *Enabling Agile Web Development through In-Browser Code Generation and*

Evaluation. In Proceedings of the 7th International Conference on Model and Data Engineering (MEDI 2017), Springer, Barcelona (Spain), 2017. Pending publication.

- Alejandro Cortiñas, Carlo Bernaschina, Miguel R. Luaces and Piero Fraternali: *Improving GISBuilder with Runtime Product Preview.* In Proceedings of the 17th International Conference on Web Engineering (ICWE 2017), LNCS 10360, Springer, Rome (Italy), 2017, pp. 549-553.

- Nieves R. Brisaboa, Alejandro Cortiñas, Miguel R. Luaces, and Oscar Pedreira: *Creating web-based GIS applications using automatic code generation techniques.* In Proceedings of the 15th International Symposium on Web and Wireless Geographical Information Systems (W2GIS 2017), LNCS 10181, Springer, Shangai (China), 2017, pp. 19-34.

- Nieves R. Brisaboa, Alejandro Cortiñas, Miguel R. Luaces, and Oscar Pedreira: *GISBuilder: A framework for the semi-automatic generation of web-based geographic information systems.* In Proceedings of the 20th Pacific Asia Conference on Information Systems (PACIS 2016), AIS Electronic Library (AISeL), Chiayi (Taiwán), 2016.

- Nieves R. Brisaboa, Alejandro Cortiñas, Miguel R. Luaces, and Matías Pol'la: *A Reusable Software Architecture for Geographic Information Systems based on Software Product Line Engineering.* In Proceedings of the 5th International Conference on Model & Data Engineering (MEDI 2015), LNCS 9344, Springer, Rodas (Grecia), 2015, pp. 320-331.

National Conferences

- Nieves R. Brisaboa, Alejandro Cortiñas, Miguel R. Luaces, and Oscar Pedreira: *Aplicando scaffolding en el desarrollo de Líneas de Producto Software.* In Proceedings of the XXI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2016), Ediciones Universidad de Salamanca, Salamanca (España), 2016, pp. 23-36.

- Alejandro Cortiñas, and Miguel R. Luaces: *Generación, almacenamiento y consulta de datos espaciales masivos.* In Proceedings of the XXII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2017), Biblioteca Digital de Sistedes, La Laguna (España), 2017.

Journals and Book Chapters

- Miguel R. Luaces, Alejandro Cortiñas, and Guillermo de Bernardo: *Fundamentos de SIG: Tecnologías básicas.* Reprografía Noroeste, S. L., 2016.

International Research Stays

- *January, 2017 - April, 2017.* Research stay at Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (Como, Italy).

D

Resumen del Trabajo Realizado

D.1 Introducción

El enfoque tradicional para el desarrollo de software se compone de una serie de pasos que deben repetirse para cada producto nuevo. Análisis de requisitos, diseño de la solución, implementación, testeo y mantenimiento son realizados una y otra vez incluso cuando se desarrollan productos similares. Todavía es común que se desarrolle el software de manera artesanal, siguiendo este proceso. El problema de este enfoque es que tanto el desarrollo de software en sí como el mantenimiento de los productos son dos procesos lentos y altamente costosos, si queremos realizar software de calidad. Por esta razón, se han llevado a cabo muchos esfuerzos buscando la industrialización del desarrollo de software. La ingeniería de líneas de productos software (LPS) y el desarrollo dirigido por modelos (DDM) son dos de los principales campos de investigación que trabajan en esta dirección.

La ingeniería de líneas de producto software usa estrategias como la producción en masa, la personalización en masa o la reutilización de artefactos de software para automatizar el desarrollo de familias de productos de software. Es decir, esta disciplina se utiliza en sistemas de software similares, diferentes sólo en ciertas características [ABKS13]. Una línea de productos software se define como "un conjunto de sistemas software que comparten un conjunto común de características

(*features*), las cuales satisfacen las necesidades específicas de un dominio o segmento particular del mercado, y que se desarrollan a partir de un sistema común de activos base (core assets) de una manera preestablecida" [CN02]. En resumen, una línea de producto software nos permite generar una familia de productos software semejantes construidos mediante el montaje e integración automática de un conjunto de componentes reutilizables comunes. El conjunto de características (*features*) proporcionadas por una LPS se suele organizar en lo que denominamos *feature model*, y cada uno de los productos a generar se define como el conjunto de características proporcionadas por el mismo.

El desarrollo dirigido por modelos es un paradigma para aplicar las ventajas del modelado a las actividades de ingeniería de software [BCW12]. Los dos conceptos principales en DDM son modelos, que son representaciones simplificadas de la realidad centrada en un dominio concreto, y transformaciones, que son operaciones manipuladoras sobre estos modelos que permiten transformarlas en modelos nuevos, más refinados, o incluso en código fuente del sistema final. La implantación real de DDM en la industria es muy baja [BCW12], pero sin embargo, sí se usa una técnica que es, de alguna manera, una aplicación informal de algunos principios de DDM: el *scaffolding*. Esta técnica fue popularizada en el 2005 por Ruby on Rails[1], y permite acelerar el desarrollo de software mediante la generación de código fuente. Suele ser utilizado por los programadores en las primeras etapas del desarrollo software, donde a partir de una especificación una herramienta permite generar código repetitivo y fácilmente abstraible. Otras plataformas actuales utilizando esta técnica son Grails[2], Spring Roo[3] o Yeoman[4], por ejemplo.

El objetivo de LPS y DDM no es sólo mejorar la eficiencia en la producción de software, sino también la calidad de los sistemas de software producidos. Ambos tratan de cambiar el paradigma del desarrollo de software de la producción artesanal a la industrial. La diferencia es que las LPS se centran en la construcción de familias de productos que comparten componentes idénticos y reutilizables, que sólo difieren en algunas características, mientras que DDM genera código específico de plataforma a partir de un modelo más abstracto y flexible. Como consecuencia, la gama de productos diferentes que se pueden crear con DDM es más amplia que la proporcionada por LPS, pero los productos generados por un LPS son más fáciles de especificar y normalmente están listos para la producción cuando se generan [PPP09, CAK+05]. Es decir, ambos enfoques buscan automatizar e industrializar el desarrollo de sistemas de software complejos, pero las herramientas

[1] http://rubyonrails.org/
[2] https://grails.org/
[3] http://projects.spring.io/spring-roo/
[4] http://yeoman.io/

y características proporcionadas por cada enfoque son de naturaleza muy diferente. Investigaciones anteriores han resuelto que algunos dominios de aplicación se beneficiarían de una combinación de LPS y DDM [TBD07, CFP08]. Los sistemas de información geográfica (SIG) son uno de esos dominios.

Un SIG es un sistema de información con características y capacidades geoespaciales [WD04]. Dentro de un sistema de información geográfica, datos geográficos como la forma de un río o el área de un edificio se pueden manejar y representar en un visor de mapas. Los SIG son ampliamente utilizados en varias aplicaciones de uso general (por ejemplo, motores de búsqueda web, redes sociales, etc.) y en varios campos de investigación y producción (por ejemplo, ingeniería, gestión de recursos, biología, ecología, logística, etc.). El impulso que se ha producido en las tecnologías de las comunicaciones y el acceso a Internet permiten el uso actualmente de los SIG en multitud de dispositivos móviles para visualizar y manejar datos geográficos almacenados en ordenadores a través de Internet. Es más, el avance en las capacidades de geoposicionamiento permite que actualmente podamos conocer nuestra posición utilizando cualquier móvil de uso común, lo que ha propiciado la aparición de nuevas funcionalidades de carácter geográfico en aplicaciones ya existentes y la incorporación de la posición de los usuarios en muchos flujos de trabajo. Ejemplos de esto son todas funcionalidades de posición de aplicaciones como Facebook o Twitter. Otro cambio en el software SIG provocado por estos avances es el incremento de las funcionalidades y uso de los sistemas de información geográfica basados en web. Tradicionalmente, estos SIG basados en web han proporcionado un pequeño conjunto de características pero, con la tecnología actual, pueden ser tan potentes como las aplicaciones SIG de escritorio, pero contando con todas las ventajas de ser una aplicación web.

Los sistemas de información geográfica siempre han compartido una enorme cantidad de funcionalidades y características entre ellos, independientemente del contexto de aplicación del SIG. Ciertos requisitos, como almacenar e indexar datos georreferenciados, realizar consultas basadas en la ubicación, mostrar información como un conjunto de capas o agrupar capas en mapas diferentes, son compartido por una buena parte de los SIG existentes. Sin embargo, los primeros componentes y software SIG solían ser implementados siguiendo diferentes e incompatibles modelos conceptuales, lógicos y físicos. Por ejemplo, dependiendo del software utilizado el tipo de dato *polygon* tenía una definición diferente, o el predicado *overlaps* contaba con un significado semático particular en cada caso. Debido a esto, era muy complejo desarrollar aplicaciones, software o componentes interoperables entre sí porque ni siquiera se podían migrar los datos de una aplicación a otra sin implementar un proceso *ad-hoc*. Para solucionar este problema, se llevó a cabo un esfuerzo conjunto y colaborativo por parte de dos organizaciones, la

International Organization for Standardization[5] (ISO) a través de la ISO/TC 211 [fSn] que define el conjunto de estándares ISO 19100, y el Open Geospatial Consortium[6] (OGC). Actualmente la mayor parte de software SIG los cumple, por lo que la interoperatibilidad entre los distintos componentes es sencilla y se pueden reemplazar componentes equivalentes sin problema.

Dado que las funcionalidades de los SIG son muy similares entre las distintas aplicaciones, éstas por lo general comparten también un conjunto común de componentes, como el visor de mapas o la biblioteca de importación de datos geográficos, siendo la mayor diferencia entre las distintas aplicaciones el dominio concreto al que se está enfocando el SIG, lo que repercute directamente en el conjunto de datos que se maneja en la aplicación. Es decir, no es lo mismo hacer una aplicación SIG que mejore el flujo de trabajo de una empresa de transportes, donde podemos ver la posición de los camiones, que una aplicación para la identificación de parcelas agrícolas. Pero incluso aquellos módulos que dependen de los datos a manejar o modelo de datos, son muy parecidos entre sí y su código se puede abstraer y generalizar en función de especificaciones como el modelo de datos. Por lo tanto, aunque cada aplicación puede tener un propósito diferente, son todas muy similares desde el punto de vista de la arquitectura, los componentes y la tecnología.

En consecuencia, encontramos que las aplicaciones SIG basados en web tienen muchos módulos que se pueden producir ensamblando componentes del estilo más clásico de las LPS (es decir, una biblioteca de visor de mapas con sus sub-características, un importador de cartografía, etc.), pero hay otros módulos que necesitan generarse específicamente para cada producto en función de alguna especificación (por ejemplo, todos los módulos relacionados con el modelo de datos, como entidades georreferenciadas, propiedades, relaciones, capas, mapas, etc., o la estructura del menú, etc. .). Las técnicas actuales de implementación LPS, la mayoría de ellas mostradas en [ABKS13, MTS+14], no son adecuadas para la generación de código dinámico a partir de las especificaciones de los productos. Encontramos que la técnica del *scaffolding*, que podemos considerar un subcojunto de los conceptos del DDM, se puede aplicar para extender las funcionalidades clásicas de las LPS y automatizar el desarrollo de esas partes del sistema. Por lo tanto, esta tesis parte de la idea de que la combinación de LPS y DDM puede ayudar a crear una herramienta completa para el desarrollo automatizado de SIG.

Aunque la idea de combinar LPS y DDM ya ha aparecido en trabajos de investigación anteriores [CAK+05, VG07], su combinación práctica está lejos de ser fácil, que es precisamente nuestro enfoque en este trabajo. En primer lugar, la mayoría de las herramientas existentes para LPS o DDM han surgido de proyectos de

[5] https://www.iso.org/home.html
[6] http://www.opengeospatial.org/

investigación y, algunos de ellos pueden tener dificultades para apoyar la variedad de tecnologías aplicadas en las aplicaciones web, en los sistemas de información geográfica y en ambos tipos de sistemas combinados. Por otro lado, las herramientas de las LPS y del DDM no son las mismas, y no encontramos ninguna plataforma capaz de combinar estos dos enfoques. Por lo tanto, el primer objetivo de esta tesis es diseñar e implementar un motor de derivación para LPS que pueda ensamblar componentes en función de un conjunto de características elegidas de un modelo de características, pero que también pueda generar código fuente transformando modelos siguiendo un enfoque más cercano a DDM. Además, el motor de derivación debe estar basado en tecnologías actuales del desarrollo de software para poder usarlo directamente en la industria.

A la hora de validar el motor de derivación y la LPS para aplicaciones SIG basadas en web, es necesario aplicarlos en un contexto industrial real. Nosotros podemos conseguir esto uniendo esfuerzos con Enxenio, una PYME (pequeña y mediana empresa) española con experiencia en sistemas de información geográfica. De hecho, esta empresa es un proveedor líder de sistemas de información geográfica basados en la web en la región de Galicia, con muchos proyectos previos para la administración pública y clientes privados. Enxenio ha colaborado con el Laboratorio de Bases de Datos de la Universidad de A Coruña desde hace mucho tiempo, y varios trabajos como [LPFCP09, BCLF+07, PBF+07] son algunos de resultados de esta relación. Para nuestro trabajo actual, Enxenio ha estado ayudando con su experiencia y sus aplicaciones web existentes basadas en SIG. Esta colaboración no es altruista ya que Enxenio se beneficiaría enormemente de los resultados de esta tesis, ya que la aplicación de SPLE para el desarrollo de futuros SIG daría a la empresa una ventaja estratégica.

Para sacar el máximo provecho de esta colaboración, los conocimientos de los expertos en arquitecturas, requisitos y tecnologías SIG debe ser tenido en cuenta de manera explícita a lo largo del proceso que defina la LPS. También tenemos acceso a código de productos existentes, otra ventaja que debemos aprovechar. Además, los sistemas de información geográfica son un campo con una fuerte estandarización, incluyendo definiciones de servicios y arquitecturas que se deben tener en cuenta. Por último, si la línea de productos de software se va a utilizar en la industria real, la evolución debe mantenerse tanto para la plataforma como para los productos generados. Por lo tanto, la metodología seguida debe facilitar el manejo de esta evolución. Como no hemos encontrado una metodología que cubriera todos todo lo mencionado, el segundo objetivo de esta tesis es la definición de una metodología para la creación de líneas de producto software que tenga en cuenta todos estos aspectos.

Dado que la LPS está diseñada para ser usada dentro de una PYME, Enxenio, no

podemos asumir que los analistas que definirán y generarán los distintos proyectos son expertos ingenieros del dominio con conocimientos en LPS. Por lo tanto, el proceso de definición y derivación de los productos debe ser simple y no debe requerir demasiado conocimientos sobre LPS o sobre ninguna tecnología específica más allá de las que ya están presentes en los procesos industriales. Por esta razón, el tercer objetivo de esta tesis es desarrollar una herramienta para la definición y derivación de los productos de manera que su uso no sea intrusivo con los procesos actuales de la compañía. Para cumplir esto, la herramienta debe ser desarrollada usando tecnologías web de forma que pueda utilizarse desde cualquier dispositivo de la compañía sin requerir instalación de ningún software adicional.

Para resumir, el objetivo principal de esta tesis es demostrar que técnicas de LPS y de DDM pueden combinarse para producir aplicaciones SIG basadas en web más eficientemente y con mayor calidad. Para llevarlo a cabo, hemos dividido este objetivo en tres objetivos específicos:

1. Diseñar e implementar un montón de derivación que pueda ensamblar componentes usando técnicas propias de LPS pero que también sea capaz de generar código fuente mediante un enfoque DDM.

2. Definir una metodología para crear LPS confeccionada para aprovechar la aportación de una PYME en cuanto a adquisición de conocimientos y a la evolución de la LPS y sus productos.

3. Desarrollar una herramienta para la especificación y derivación de los productos de la LPS de modo que su uso no sea intrusivo con los flujos de trabajo de una compañía de desarrollo de software.

D.2 Estructura de la tesis

Esta tesis está compuesta por tres partes. La primera parte está dedicada a la metodología y consta de tres capítulos. El primero introduce los conceptos detrás de las líneas de productos de software y describe el estado del arte del campo, incluyendo las ventajas del SPL y los problemas no resueltos. El segundo muestra la metodología que hemos definido para la aplicación de la ingeniería de línea de productos software en cualquier dominio.

La segunda parte cubre la aplicación de la metodología previamente definida dentro de nuestro concurso, los sistemas de información geográfica basados en la web. En el primer capítulo de esta parte, hacemos un breve resumen de los conceptos para los sistemas de información geográfica, y describimos algunos programas relacionados con SIG. En el resto de los capítulos de esta parte, cada

paso de nuestra metodología se aborda: análisis de requisitos, extracción de requisitos y características de productos existentes; Diseño de arquitectura, donde estudiamos las arquitecturas de referencia para SIG y seleccionamos una como propia; Evaluación y derivación de los productos, mostrando la trazabilidad entre las características y la arquitectura y los detalles relativos a la derivación de productos específicos.

La última parte es sobre GISBuilder, una herramienta que implementa las especificaciones extraídas de nuestro proceso. En el primer capítulo de esta parte se muestra el estado de la técnica en las tecnologías de líneas de productos de software y en la generación industrial de técnicas de software. A continuación describimos nuestra herramienta y cada parte que la compone. Finalmente validamos nuestra herramienta y proponemos algunos casos de uso. El último capítulo es conclusiones y trabajo futuro.

D.3 Contribuciones y Conclusiones

El objetivo principal de nuestro trabajo es el diseño de una línea de productos software para sistemas de información geográfica. Más específicamente, nos centramos en aplicaciones SIG en web. No circunscribimos el dominio del SIG generado, por lo que el LPS debe ser capaz de crear aplicaciones genéricas. Para lograr este objetivo, seguimos una estrategia de tres pasos: primero abordamos la decisión sobre la metodología a aplicar, luego definimos nuestro LPS a través de la aplicación de la metodología elegida y finalmente diseñamos e implementamos nuestra herramienta siguiendo los requerimientos de Nuestro LPS.

Debido al contexto de nuestro trabajo, decidimos desarrollar una nueva metodología que pueda aprovechar nuestras ventajas, como contar con una empresa de apoyo (Enxenio) con expertos SIG y productos existentes. Al mismo tiempo, también buscamos metodologías existentes que se ajusten a nuestro problema, y hemos encontrado tres metodologías diferentes que combinadas trabajan en sinergia para complementarse unas con otras. Por lo tanto, nuestra primera contribución es el diseño de una metodología para la definición de líneas de productos software, basada en otras tres metodologías y mejorada por nosotros mismos teniendo en cuenta la entrada de expertos de la industria.

Nuestra segunda y principal contribución es el diseño de una línea de productos de software para sistemas de información geográfica basados en la web. Este diseño es el resultado de la aplicación de nuestra metodología. Dos contribuciones derivadas son:

- Una lista exhaustiva de características para la información geográfica genérica

basada en la web teniendo en cuenta los diferentes productos existentes.

- Una arquitectura para sistemas de información geográfica basados en arquitectura de referencia, especificada por estándares, y mejorada con características Desde arquitecturas de productos existentes.

Nuestra última contribución es una herramienta para la definición y generación de sistemas de información geográfica basados en la web. Esta herramienta sigue las especificaciones determinadas en el diseño de nuestro LPS, e incluye varias bibliotecas que son contribuciones propias.

D.4 Trabajo Futuro

Los siguientes pasos de nuestra investigación son:

- Implementación de las características aún no implementadas, así como evolución y mantenimiento de los componentes actuales teniendo en cuenta los productos que los utilizan.
- Diseño de una metodología para enfrentar la evolución de los productos y la plataforma de forma sincronizada. Un enfoque en esta dirección es el uso de un sistema de control de versiones, como git, para apoyar esta metodología.
- Diseñar un sistema integrado basado en la integración continua para que la herramienta sea totalmente independiente del despliegue, incluso para la implementación de pila completa. Este sistema debe trabajar en sintonía con la metodología de evolución para que un equipo de desarrollo pueda trabajar en un producto, mover las actualizaciones realizadas al código de la plataforma y reubicar fácilmente el producto nuevamente con los nuevos activos.
- Actualizar el controlador de variabilidad con más operaciones desde [BSRC10], ya que no hay ninguna herramienta que proporciona todas ellas.
- Aplicar la misma metodología en el contexto de una familia de productos diferente.
- En el contexto de la ingeniería web, sería interesante ir más allá de la capacidad de vista previa en tiempo de ejecución y proporcionar un marco para la programación en vivo de los productos, mostrando los productos en ejecución en una ventana en el navegador y la interfaz de especificación en otra diferente.

Bibliography

[ABKS13] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines*. Springer, 2013.

[ADT07] Felipe Anfurrutia, Oscar Díaz, and Salvador Trujillo. On Refining XML Artifacts. *Web Engineering*, 4607:473–478, 2007.

[AK04] S.A. Ajila and A.B. Kaba. Using traceability mechanisms to support software product line evolution. In *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, 2004. IRI 2004.*, pages 157–162. IEEE, 2004.

[AKL09] Sven Apel, Christian Kästner, and Christian Lengauer. FEATURE HOUSE: Language-Independent, Automated Software Composition. In *Proceedings of the 31st International Conference on Software Engineering*, pages 221–231, 2009.

[Bat05] Don Batory. Feature models, grammars, and propositional formulas. In *International Conference on Software Product Lines*, volume 3714, pages 7–20. Springer, 2005.

[BCA+13] Agustina Buccella, Alejandra Cechich, Maximiliano Arias, Matías Pol'la, Maria Del Socorro Doldan, and Enrique Morsan. Towards systematic software reuse of GIS: Insights from a case study. *Computers and Geosciences*, 54:9–20, 2013.

[BCLF+07] N R Brisaboa, J A Cotelo-Lema, A Fariña, M R Luaces, J R Parama, and J R R Viqueira. Collecting and publishing large multiscale geographic datasets. *Software: Practice and Experience*, 37(12):1319–1348, oct 2007.

[BCLP16] Nieves R Brisaboa, Alejandro Cortiñas, Miguel R Luaces, and Oscar Pedreira. Aplicando scaffolding en el desarrollo de Líneas de Producto

Software. In *Proceedings of the XXI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2016)*, 2016.

[BCP+14] Agustina Buccella, Alejandra Cechich, Matías Pol'la, Maximiliano Arias, Maria Del Socorro Doldan, Enrique Morsan, Maria del Socorro Doldan, and Enrique Morsan. Marine ecology service reuse through taxonomy-oriented SPL development. *Computers and Geosciences*, 73:108–121, 2014.

[BCPA14] Agustina Buccella, Alejandra Cechich, Matías Pol'la, and Maximiliano Arias. Un Modelo de Metadatos para la Gestión de la Variabilidad en Líneas de Productos de Software. *15th Argentine Symposium on Software Engineering, ASSE 2014*, pages 158–172, 2014.

[BCPA16] Agustina Buccella, Alejandra Cechich, Matias Pol, and Maximiliano Arias. Software Product Line Reengineering : A Case Study on the Geographic Domain. *Journal of Computer Science & Technology*, 16(1):14–28, 2016.

[BCW12] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*, volume 1. Morgan & Claypool Publishers, sep 2012.

[BMML15] Peter A Burrough, Rachael McDonnell, Rachael A McDonnell, and Christopher D Lloyd. *Principles of geographical information systems*. Oxford University Press, 2015.

[Bos00] Jan. Bosch. *Design and use of software architectures: adopting and evolving a product-line approach*. Addison-Wesley, 2000.

[BPCA16] Agustina Buccella, Matias Pol'La, Alejandra Cechich, and Maximiliano Arias. A Variability Representation Approach Based on Domain Service Taxonomies and Their Dependencies. In *Proceedings - International Conference of the Chilean Computer Science Society, SCCC*, volume 2016-Septe, pages 116–119, 2016.

[Bra04] Gilad Bracha. Generics in the java programming language. http://www.oracle.com/technetwork/java/javase/generics-tutorial-159168.pdf, July 2004. (Accessed: 24-may-2017).

[BRN+13] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. A survey of variability modeling in industrial practice. *Proceedings of the Seventh*

International Workshop on Variability Modelling of Software-intensive Systems - VaMoS '13, page 1, 2013.

[BRPD05] Luca Balzerani, D Di Ruscio, Alfonso Pierantonio, and Guglielmo De Angelis. A product line architecture for web applications. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 1689–1693. ACM, 2005.

[BSR04] Don Batory, Jacob Neal Sarvela, and Axel Rauschmayer. Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, 30(6):355–371, 2004.

[BSRC10] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, sep 2010.

[CAK+05] Krzysztof Czarnecki, Michał Michal Antkiewicz, Chang Hwan Peter Chp Kim, Sean Lau, and Krzysztof Pietroszek. Model-driven software product lines. *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 126–127, 2005.

[CBLF17] Alejandro Cortiñas, Carlo Bernaschina, Miguel R Luaces, and Piero Fraternali. Enabling Agile Web Development through In-Browser Code Generation and Evaluation. *Proceedings of the 7th International Conference on Model and Data Engineering (MEDI 2017)*, 2017. Pending publication.

[CD03] Rafael Capilla and Juan C Dueñas. Light-weight product-lines for evolution and maintenance of web sites. In *Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on*, pages 53–62. IEEE, 2003.

[CFP08] Carlos Cetina, Joan Fons, and Vicente Pelechano. Applying software product lines to build autonomic pervasive systems. *Proceedings - 12th International Software Product Line Conference, SPLC 2008*, 2:117–126, 2008.

[CGR+12] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. Cool features and tough decisions: A comparison of variability modeling approaches. *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '12*, pages 173–182, 2012.

[CHE05] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged Configuration Through Specialization and Multi-Level Configuration of Feature Models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[Che06] Nicholas Chen. Convention over configuration, November 2006. (Accessed: 24-may-2017).

[CN02] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[Com] Federal Geographic Data Committee. Geospatial standards. https://www.fgdc.gov/standards. (Accessed: 22-may-2017).

[CSA15] Gerry Gerard Claps, Richard Berntsson Svensson, and Aybüke Aurum. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology*, 57:21–31, 2015.

[Dat] Databases and Software Engineering Workgroup University of Magdeburg. Tools for feature-oriented software development. http://wwwiti.cs.uni-magdeburg.de/iti_db/research/fosd-tools/. (Accessed: 28-mar-2017).

[DMG13] Paule-Annick Devoine, Bogdan Moisuc, and Jerome Gensel. GENGHIS: an Environment for the Generation of Spatiotemporal Visualization Interfaces. In *Innovative Software Development in GIS*, pages 121–150. John Wiley & Sons, Inc., 2013.

[DPG14] Jessica Díaz, Jennifer Pérez, and Juan Garbajosa. A model for tracing variability from features to product-line architectures: A case study in smart grids. *Requirements Engineering*, 20(3):323–343, 2014.

[DW99] Desmond Francis D'Souza and Alan Cameron Wills. *Objects, components, and frameworks with UML : the catalysis approach*. Addison-Wesley, 1999.

[EIA+11] KARAGIOZI Eleni, FOUNTOULIS Ioannis, KONSTANTINIDIS Alexandros, ANDREADAKIS Emmanouil, and NTOUROS Konstantinos. Flood hazard assessment based on geomorphological analysis with GIS tools-The case of Laconia (Peloponnesus, Greece). In *Proceedings, Symposium GIS Ostrava*, volume 2011, page 11p, 2011.

[ESR] ESRI ESRI. Shapefile technical description, jul. 1998. http://support.esri.com/white-paper/279. (Accessed: 19-may-2017).

[ESR17] ESRI. Architecting the ArcGIS Platform: Best Practices, 2017.

[FAB+13] Marianela Ciolfi Felice, Mathieu Acher, Arnaud Blouin, Olivier Barais, and Others. Interactive visualisation of products in online configurators: a case study for variability modelling technologies. In *Proceedings of the 17th International Software Product Line Conference co-located workshops*, pages 82–85. ACM, 2013.

[FKA+13] Janet Feigenspan, Christian Kästner, Sven Apel, Jörg Liebig, Michael Schulze, Raimund Dachselt, Maria Papendieck, Thomas Leich, and Gunter Saake. Do background colors improve program comprehension in the #ifdef hell? *Empirical Software Engineering*, 18(4):699–745, 2013.

[FPK+11] Janet Feigenspan, Maria Papendieck, Christian Kästner, Mathias Frisch, and Raimund Dachselt. FeatureCommander: Colorful #ifdef World. In *Proceedings of the 15th International Software Product Line Conference on - SPLC '11*, page 1, New York, New York, USA, aug 2011. ACM Press.

[fSa] International Organization for Standardization. Iso 19107:2003 - geographic information: Spatial schema. https://www.iso.org/standard/26012.html. (Accessed: 22-may-2017).

[fSb] International Organization for Standardization. Iso 19110:2016 - geographic information: Methodology for feature cataloguing. https://www.iso.org/standard/57303.html. (Accessed: 22-may-2017).

[fSc] International Organization for Standardization. Iso 19111:2007 - geographic information: Spatial referencing by coordinates. https://www.iso.org/standard/41126.html. (Accessed: 22-may-2017).

[fSd] International Organization for Standardization. Iso 19112:2003 - geographic information: Spatial referencing by geographic identifiers. https://www.iso.org/standard/26017.html. (Accessed: 22-may-2017).

[fSe] International Organization for Standardization. Iso 19115-1:2014 - geographic information: Metadata - part 1: Fundamentals. https://www.iso.org/standard/53798.html. (Accessed: 22-may-2017).

[fSf] International Organization for Standardization. Iso 19116:2004 - geographic information: Positioning services. https://www.iso.org/standard/37805.html. (Accessed: 22-may-2017).

[fSg] International Organization for Standardization. Iso 19119:2005 - geographic information: Services. https://www.iso.org/standard/39890.html. (Accessed: 22-may-2017).

[fSh] International Organization for Standardization. Iso 19119:2016 - geographic information: Services. https://www.iso.org/standard/59221.html. (Accessed: 22-may-2017).

[fSi] International Organization for Standardization. Iso 19125:2004 - geographic information: Simple feature access – part 1: Common architecture. https://www.iso.org/standard/40114.html. (Accessed: 22-may-2017).

[fSj] International Organization for Standardization. Iso 19128:2005 - geographic information: Web map server interface. https://www.iso.org/standard/32546.html. (Accessed: 22-may-2017).

[fSk] International Organization for Standardization. Iso 19139:2007 - geographic information: Metadata - xml schema implementation. https://www.iso.org/standard/32557.html. (Accessed: 22-may-2017).

[fSl] International Organization for Standardization. Iso 19142:2010 - geographic information: Web feature service. https://www.iso.org/standard/42136.html. (Accessed: 22-may-2017).

[fSm] International Organization for Standardization. Iso/iec 13249-3:2016 - information technology: Database languages - sql multimedia and application packages - part 3: Spatial. https://www.iso.org/standard/60343.html. (Accessed: 22-may-2017).

[fSn] International Organization for Standardization. Iso/tc 211 standards catalogue - geographic information/geomatics. https://www.iso.org/committee/54904/x/catalogue/. (Accessed: 22-may-2017).

[GG02] Hassan Gomaa and Mark Gianturco. Domain modeling for World Wide Web based software product lines with UML. In *International Conference on Software Reuse*, pages 78–92. Springer, 2002.

[GLA+09] Dharmalingam Ganesan, Mikael Lindvall, Chris Ackermann, David McComas, and Maureen Bartholomew. Verifying Architectural Design Rules of the Flight Software Product Line. In *Proceedings of the 13th International Software Product Line Conference*, number July 2015 in SPLC '09, pages 161–170, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[HA03] John E. Harmon and Steven J. Anderson. *The Design and Implementation of Geographic Information Systems*. Wiley, 1st edition, 2003.

[HZS+15] Claus Hunsen, Bo Zhang, Janet Siegmund, Christian Kästner, Olaf Leßenich, Martin Becker, and Sven Apel. Preprocessor-based variability in open-source and industrial software systems: An empirical study. *Empirical Software Engineering*, apr 2015.

[IMY+16] Takahiro Iida, Masahiro Matsubara, Kentaro Yoshimura, Hideyuki Kojima, and Kimio Nishino. PLE for automotive braking system with management of impacts from equipment interactions. *Proceedings of the 20th International Systems and Software Product Line Conference on - SPLC '16*, pages 232–241, 2016.

[INS] INSPIRE. Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 establishing an Infrastructure for Spatial Information in the European Community.

[INS08] INSPIRE Network Services Drafting Team. INSPIRE Network Services Architecture, 2008.

[ISO09] ISO. ISO/IEC 10746-3:2009 - Information technology – Open distributed processing – Reference model: Architecture, 2009.

[JB09] Hans Peter Jepsen and Danilo Beuche. Running a software product line: standing still is going backwards. *SPLC '09 Proceedings of the 13th International Software Product Line Conference*, pages 101–110, 2009.

[KAK08] Christian Kästner, Sven Apel, and Martin Kuhlemann. Granularity in software product lines. In *Proceedings of the 30th international conference on Software engineering - ICSE '08*, page 311, New York, New York, USA, may 2008. ACM Press.

[KATS12] Christian Kästner, Sven Apel, Thomas Thüm, and Gunter Saake. Type checking annotation-based product lines. *ACM Transactions on Software Engineering and Methodology*, 21(3):1–39, 2012.

[KB07] J. Kunze and T. Baker. The Dublin Core Metadata Element Set. RFC 5013 (Informational), August 2007.

[KCH+90] Kyo C Kang, Sholom G Cohen, James a Hess, William E Novak, and a Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. *Distribution*, 17(November):161, 1990.

[KKL+98] 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. *Annals of Software Engineering*, 5(1):143–168, 1998.

[KS90] Sunder Kekre and Kannan Srinivasan. Broader Product Line: A Necessity to Achieve Success? *Management Science*, 36(10):1216–1231, 1990.

[KTS+09] Christian Kästner, Thomas Thum, Gunter Saake, Janet Feigenspan, Thomas Leich, Fabian Wielgorz, and Sven Apel. FeatureIDE: A tool framework for feature-oriented software development. In *2009 IEEE 31st International Conference on Software Engineering*, pages 611–614. IEEE, 2009.

[LAL+10] Jörg Liebig, Sven Apel, Christian Lengauer, Christian Kästner, and Michael Schulze. An analysis of the variability in forty preprocessor-based software product lines. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 105–114, 2010.

[LdS01] Carl Lagoze and Herbert de Sompel. The Open Archives Initiative: Building a Low-barrier Interoperability Framework. In *Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '01, pages 54–62, New York, NY, USA, 2001. ACM.

[LGBH09] Miguel Laguna, Bruno González-Baixauli, and Carmen Hernández. Product line development of web systems with conventional tools. *Web Engineering*, pages 205–212, 2009.

[LGM15] Paul A Longley, Michael F Goodchild, and David J Maguire. *Geographic Information Science and Systems*. Blackwell Publ, Hoboken, NJ, edición: r edition, mar 2015.

[LKA11] Jörg Liebig, Christian Kästner, and Sven Apel. Analyzing the Discipline of Preprocessor Annotations in 30 Million Lines of C Code. *Proceedings of the 10th ACM International Conference on AspectOriented Software Development AOSD*, pages 191–202, 2011.

[LPFCP09] Miguel R Luaces, David Trillo Pérez, J Ignacio Lamas Fonte, and Ana Cerdeira-Pena. An Urban Planning Web Viewer Based on AJAX. In Gottfried Vossen, Darrell D E Long, and Jeffrey Xu Yu, editors, *Web Information Systems Engineering - WISE 2009*, Lecture {Notes} in {Computer} {Science}, pages 443–453. Springer Berlin Heidelberg, oct 2009.

[LST+06] Daniel Lohmann, Fabian Scheler, Reinhard Tartler, Olaf Spinczyk, and Wolfgang Schröder-Preikschat. A quantitative analysis of aspects in the eCos kernel. *ACM SIGOPS Operating Systems Review*, 40(4):191, 2006.

[LT92] Robert Laurini and Derek Thompson. *Fundamentals of Spatial Information Systems*. Academic Press, London, 1 edition edition, apr 1992.

[MD15] Leticia Montalvillo and Oscar Díaz. Tuning GitHub for SPL development: branching models & repository operations for product engineers. In *Proceedings of the 19th International Conference on Software Product Line Pages - SPLC'15*, pages 111–120. ACM, jul 2015.

[MGP08] Belén Magro, Juan Garbajosa, and Jennifer Pérez. A software product line definition for validation environments. *Proceedings - 12th International Software Product Line Conference, SPLC 2008*, pages 45–54, 2008.

[MGP09] Belén Magro, Juan Garbajosa, and Jennifer Pérez. The Development of A Software Product Line for Validation Environments. In *Applied Software Product Line Engineering*, pages 173–200. Taylor and Francis, 2009.

[MMYA02] Hafedh. Mili, Ali Mili, Sherif Yacoub, and Edward Addy. *Reuse based software engineering : techniques, organization and measurement*. Wiley, 2002.

[MP13] Michael S. Mikowski and Josh C. Powell. *Single Page Web Applications: JavaScript end-to-end*. Manning, 2013.

[MP14] Andreas Metzger and Klaus Pohl. Software product line engineering and variability management: achievements and challenges. In *Proceedings of the on Future of Software Engineering*, pages 70–84. ACM, 2014.

[MPV05] Yannis Manolopoulos, Apostolos N Papadopoulos, and Michael Gr Vassilakopoulos. *Spatial databases: technologies, techniques and trends*. IGI Global, 2005.

[MSC+14] Ivan Do Carmo Machado, Alcemir Rodrigues Santos, Yguarata Cerqueira Cavalcanti, Eduardo Gomes Trzan, Marcio Magalhaes de Souza, and Eduardo Santana de Almeida. Low-level variability support for web-based software product lines. *Variability Modelling of Software intensive Systems (VaMoS)*, 2014.

[MTS+14] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, and Gunter Saake. An overview on analysis tools for software product lines. In *Proceedings of the 18th International Software Product Line Conference on Companion Volume for Workshops, Demonstrations and Tools - SPLC '14*, pages 94–101, New York, New York, USA, sep 2014. ACM Press.

[NBM13] Elisa Yumi Nakagawa, Martin Becker, and Jose Carlos Maldonado. Towards a Process to Design Product Line Architectures Based on Reference Architectures. *Proceedings of the 17th International Software Product Line Conference*, pages 157–161, 2013.

[PBC+12] Patricia Pernich, Agustina Buccella, Alejandra Cechich, Maximiliano Arias, Matías Pol'la, María del Socorro Doldan, and Enrique Morsan. Product-Line Instantiation Guided By Subdomain Characterization : A Case Study. *Journal of Computer Science & Technology*, 12(3):116–122, 2012.

[PBF+07] Ángeles S. Places, Nieves R. Brisaboa, Antonio Fariña, Miguel R. Luaces, José R. Paramá, and Miguel R. Penabad. The Galician virtual library. *Online Information Review*, 31(3):333–352, jun 2007.

[PBL05] Klaus Pohl, Günter Böckle, and Frank Van Der Linden. *Software Product Line Engineering: foundations, principles and techniques*, volume 49. Springer Science & Business Media, 2005.

[Per02] George Percivall. OpenGIS Service Architecture, 2002.

[PJ05] Ulf Pettersson and Stan Jarzabek. Industrial experience with building a web portal product line using a lightweight, reactive approach. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 326–335. ACM, 2005.

[PO97] T.T. Pearse and P.W. Oman. Experiences developing and maintaining software in a multi-platform environment. In *Proceedings International Conference on Software Maintenance*, pages 270–277. IEEE Comput. Soc, 1997.

[PPP09] B Pérez, M Polo, and M Piatini. Software Product Line Testing-A Systematic Review. In *4th International Conference on Software and Data Technologies (ICSoft 2009), Sofia, Bulgaria*, 2009.

[RSV01] Philippe Rigaux, Michel Scholl, and Agnes Voisard. *Spatial databases: with application to GIS*. Morgan Kaufmann, 2001.

[SC92] Henry Spencer and Zoology Computer. # ifdef Considered Harmful , or Portability Experience With C News. *Usenix*, pages 185–198, 1992.

[SC03] Shashi Shekhar and Sanjay Chawla. *Spatial databases: a tour*, volume 2003. prentice hall Upper Saddle River, NJ, 2003.

[SLB+10] Steven She, Rafael Lotufo, Thorsten Berger, Andrezj Wasowski, and Krzysztof Czarnecki. The variability model of the linux kernel. *Fourth International Workshop on Variability Modeling of Software-intensive Systems (VaMoS 2010)*, page 108, 2010.

[SLSA13] Sandro Schulze, Jörg Liebig, Janet Siegmund, and Sven Apel. Does the Discipline of Preprocessor Annotations Matter?: A Controlled Experiment. In *Proceedings of the 12th International Conference on Generative Programming: Concepts and Experiences*, pages 65–74, 2013.

[SRG11] Klaus Schmid, Rick Rabiser, and Paul Grünbacher. A comparison of decision modeling approaches in product lines. *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '11*, pages 119–126, 2011.

[Sta87] Davis M Stanley. *Future perfect*. Addison-Wesley, 1987.

[TBD07] Salvador Trujillo, Don Batory, and Oscar Diaz. Feature Oriented Model Driven Development: A Case Study for Portlets. In *Software*

Engineering, 2007. ICSE 2007. 29th International Conference on, pages 44–53. IEEE, may 2007.

[TSSPL09] Reinhard Tartler, Julio Sincero, Wolfgang Schröder-Preikschat, and Daniel Lohmann. Dead or Alive. In *Proceedings of the First International Workshop on Feature-Oriented Software Development - FOSD '09*, page 81, New York, New York, USA, oct 2009. ACM Press.

[UBFC14] S. Urli, M. Blay-Fornarino, and P. Collet. Handling complex configurations in software product lines: A tooled approach. In *Proceedings of the 18th International Conference on Software Product Line Pages - SPLC'14*, volume 1, pages 112–121, 2014.

[Van02] Frank Van der Linden. Software product families in Europe: The Esaps & Café projects. *IEEE Software*, 19(4):41–49, 2002.

[vdLSR+07] Frank van der Linden, Klaus Schmid, Eelco Rommes, Frank van der Linden, Klaus Schmid, and Eelco Rommes. *Software product lines in action: the best industrial practice in product line engineering.* Springer, 2007.

[VG07] Markus Voelter and Iris Groher. Product line implementation using aspect-oriented and model-driven software development. *Proceedings - 11th International Software Product Line Conference, SPLC 2007*, pages 233–242, 2007.

[WA17] Austin Wright and Henry Andrews. JSON Schema: A Media Type for Describing JSON Documents. Internet-Draft draft-wright-json-schema-01, Internet Engineering Task Force, April 2017.

[WAB+14] Greg Wilson, D A Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven H D Haddock, Kathryn D Huff, Ian M Mitchell, Mark D Plumbley, and Others. Best practices for scientific computing. *PLoS Biol*, 12(1):e1001745, 2014.

[WCK06] David M Weiss, Paul Clements, and Charles W Krueger. Software Product Line Hall of Fame. *SPLC 2006: Proceedings of the 10th International Software Product Line Conference*, page 237, 2006.

[WD04] Michael F Worboys and Matt Duckham. *GIS: a computing perspective.* CRC Press, Boca Raton, Fla, 2 edition edition, may 2004.

[YG05] José Manuel Cotos Yáñez and José Ángel Taboada González. *Sistemas de información medioambiental.* Netbiblo, 2005.