Extractive SPL Adoption Applied Into a Small Software Company

Luisa Rincón *, Edwin Muñoz †, Juan-C Martinez *, Marie-C Pabon * and Gloria Álvarez *
*Departamento de Electrónica y Ciencias de la Computación, Pontificia Universidad Javeriana
Cali, Colombia
†Acople Tecnológico SAS - Cali - Colombia
‡Centre de Recherche en Informatique, Université Panthéon Sorbonne
Paris, France

Abstract—Acople Tecnológico SAS is a Colombian enterprise specialized in software development whose products use devices for biometric facial and fingerprint recognition. The functional overlap of such products is an opportunity for systematic reuse to reduce software development and maintenance costs. In this sense, in this paper we report our experiences, lessons learned and future work from applying concepts of software product lines engineering to evolve three of its different software applications into a software product line. Based on our experience, we explain how we have tailored existing concepts of product line engineering, specifically how we have followed so far the process, which includes the variability analysis that refers to the study of commonalities and variabilities among a set of software applications, the development of reusable software assets, and the derivation of individual products from the product line.

Keywords—Software Product Lines, Domain Engineering, Feature Models, Core Assets, Industrial Case, Reusable artifacts, Extractive adoption

I. INTRODUCTION

Software Product Line Engineering (SPL) is a software development paradigm that seeks to create product families rather than creating individual products. These families group a set of products that share common characteristics while having some variations from each other. The software product lines (SPL) allow capitalize the development effort in many products while that efficiently manage their differences. The SPL promise benefits such as reuse and reduction in development time and costs [1].

The domain engineering and the application engineering are the two main process of the SPL [1]. Domain engineering consists in studying the commonalities and variabilities among a set of software applications and developing the reusable artifacts and resources that form the basis for the software product line (core assets) 1. These core assets will enable the rapid construction of particular software system belonging to a same family of products. Application engineering, in turn, consists of analyzing, designing, building, customizing, or testing the final products, using the core assets and the specific requirements expressed by the customers. The domain engineering process is executed once in order to construct the product-line infrastructure, whereas the application engineering phase is executed each time a new product is derived from the product line.

SPLE has been applied in several domains, especially in large-scale software development [1]–[6], but it also has increasingly garnered interest from small to medium-sized companies. Motivated by those experiences our general goal is to apply the related concepts of software product line engineering in Acople Tecnológico SAS, a small Colombian enterprise specialized in software development.

In this paper, we propose a step to achieve this goal. Specifically, we present how we have followed so far the process, which includes the variability analysis that refers to the study of commonalities and variabilities among a set of software applications, the development of reusable software assets and the derivation of individual products from the product line. We have followed an extractive adoption strategy [7], because it allows us to develop the new assets taking advantages of the software artifacts already developed by Acople Tecnológico. In fact, we used as reference three applications that Acople Tecnológico had previously launched on the market.

We believe the use of a SPL approach in Acople Tecnológico can be significantly advantageous because this company has a portfolio of software products that share a significant amount of functionality and code, but are developed, tested and maintained separately. This has lead to high costs in terms of time and technical resources which wasted technical resources and involved a high investment in time and cost.

The remaining parts of this paper is structured as follows. Section II, gives an overview of the characteristics of the company and its context (domain analysis) and presents the model as a result of analysing company’s applications. Section III explains how we defined the reusable modules that implement the features defined in the variability model that we proposed as result of analyzing the company. Section IV, presents the benefits and lessons learned in the use of a SPL approach. Section V, provides a summary of works related to our proposal, and finally Section VI presents the conclusions and suggests future research directions.

II. VARIABILITY ANALYSIS

A. Domain Analysis

Acople Tecnológico SAS is a small software company located in Cali, Colombia. It has over six years of experience and since its establishment, it has been delivering software solutions at the enterprise level using biometric devices. Along
In the years, Acople Tecnológico SAS has developed and marketed several independent applications, in fact, well-known Colombian enterprises such as Proservis Temporales SAS, Coca Cola Cali, or Servientrega SA, have successfully used Acople Tecnológico’s applications.

Based on company’s experience, we choose three applications developed by Acople Tecnológico that share many different features as a base to carry out an analysis of their variability to define the variability inherent to the product line.

Chosen applications were: Working Time Control, Access Control, and Restaurant. A brief explanation of each application is presented below:

- **Working Time Control**: this application measures how many time each employee works during his working day. Its main purpose is to identify the start and the end of the working days to prevent fraud and irregularities when working time of each employee is calculated.

- **Access Control**: it manages permissions to control access to highly secure areas. To achieve this, the application identify who is each person by mean of biometric devices.

- **Restaurant**: this application uses biometric identification to control the schedule to delivery food and to authorize people for receiving lunches or dinners when they are sponsored by an enterprise.

Several analysis techniques can be used to analyze and model the variability. In our case, we used feature modeling as the notation to represent our product line, because this notation has been widely adopted in the Software Product Lines community [8]. Under this notation, each feature is a node in a tree structure and represents a distinguishable characteristic that is relevant to our stakeholders, from a technical or functional point of view. The tree’s root of the Feature Model represents whole product line, and it is decomposed into its most prominent features. Then, these prominent features are decomposed into their prominent sub-features, and so on, until the family is entirely decomposed into features.

Among features there are several relationships to define how features are related each other, depending on, for instance, if a child is an optional or a mandatory feature. Specifically, there are three types of relations: mandatory, optional and group cardinality. Mandatory features (dark circles) are always selected, optional features (white circles), can be chosen or not, and cardinality group features (arc with UML cardinality), where the selection is realized among a limited set of alternatives according to the cardinality. In addition to that, the notation introduces two types of constraints among features: requires and excludes. The requires constraint states that for a given feature to be selected, the required feature has to be selected too. The excludes constraint states that for a given feature to be selected, the excluded feature has to be deselected [9].

Given the complexity of each application, we modeled the variability of each application through individual feature models first. Doing so, we identified features shared by the three applications such as Reports, People, Notification or Device. Furthermore, we identify features such as Calendar or Schedule that were considered only by one of the applications, but could be optionally used in other applications to extend their functionality. Moreover, we detected some features that were managed as separate concepts, but could be generalized into more general terms in order to unify them as one. For instance, we created the feature Application Type to generalize the three types of applications that we re factored to extract the product line, and we decomposed this feature into three sub-features: Working Time, Restaurant and Access.

### B. Resulting Feature Model

Figure 1 shows the feature model we constructed as a result of analyzing each application separately. Below, we briefly outline the functionality of each feature:

- **Application Type** refers to the main functionality of a product. Currently, it includes Working Time, Access, and Restaurant types. There is a mandatory relation between the features Working Time and Schedule. In fact, a product that includes Working Time should also include Schedule, to manage staff shifts.

- **People** is related, on the one hand, with the management of the people that could be identified by the biometric devices (Person Identification) and, on the other hand, with the management of the administrator users (Application Users). Any product of this product line should always identify, by mean of biometric devices, people who belong to the organization (Insiders), and it may optionally identify people not belonging to the organization (Outsiders). The application administrator users could generate Reports.

- **Records** represents the mechanism used to transfer the information captured by the biometric devices to the application. This communication could be Online when the device has Internet access or Batch when records should be manually downloaded to the system. It is noteworthy that if the Application Type is Access, then it is essential to have Internet access, and thus there is an exclusion relation between the features Access and Batch.

- **Device** refers to different kind of biometric devices that could be used by an application. Currently, the SPL considers two kinds of devices, Facial and Fingerprint, however, the SPL could be extended to incorporate iris, voice, or card recognition.

- **Number of People** represents the maximum number of people that the biometric device could identify. This feature has two sub-features \( P \) \( 0 \) to \( 1200 \), which means from zero up to one thousand two hundred people is supported by the device, and \( P \) \( 1200 \) to infinite, which means the device supports over one thousand two hundred people. Currently, the facial recognition devices incorporated in the SPL support a maximum of 1200 people. Thus, there is requires relation between the features Facial and \( P \) \( 0 \) to \( 1200 \).

- **Transaction** points out how a new event is registered. An automatic transaction takes the data directly from the biometric device. Otherwise, the data come from
a manual process. Any product derived from the SPL would have defined one transaction type. Optionally, a biometric application can manually register data. Otherwise, data should be captured by biometric devices.

- **Measure Unit** indicates what is measured, calculated, or controlled by the application. For example, the feature *Consumption* has to be used for product or service management, or to control the consumption. *Activity* has to be used in transactional applications, for example, to manage the purchasing of goods and services, and *Time* has a variety of uses, such as, working or delivery time calculation. Depending on what type of application is selected it is necessary to select a different Measure Unit. We use among those features some requires relations, as can be seen in the Figure 1, to control this dependency.

- **Notification** is about notifying to application users and stakeholders particular system conditions or events.

- **Calendar** includes the organization and management of calendar, including holidays. According to our domain analysis applications that contain the feature *Access* must also include the Calendar feature.

The final version of the feature model was validated and semantically verified using Variamos. Variamos is a tool to perform verification of feature models, then after analyzing our feature model with this tool, we made sure that it has no errors. Additionally, by using Variamos, we found the number of different products that could be obtained from our feature model. Variamos determines the number of products based on the combinations resulting from adding and subtracting features. The analysis of the proposed feature model established that 320 different products could be generated. Taking into account that the model includes the features found in three previously developed products, the possibility to obtain 320 products shows that software product lines facilitate reuse planning.

According to Acople Tecnológico’s experience, of these 320 potential applications, the feature *Application Type* determines the main commercial products and therefore six of them, resulting from combining this feature and its sub features, are very attractive from a commercial point of view, even though other possible combinations could be commercialized too. Based on this idea, Acople Tecnológico plans to define a classification of products according to the *Application Type* to manage a licensing schema. This schema could include several versions, like *Basic*, *Business*, and *Corporate* licenses. The license would be defined along with the product characteristics and functionality. For example, a *Working Time* control product could be a *Basic* application whereas a *Working time + Access + Restaurant* product could have a *Corporate* license.

Once we analyzed the common and variable elements in our domain, the next step was to design a flexible software architecture that supports the specified variations. Next section explains how this goal has been achieved in our particular case.

### III. DEVELOPMENT OF REUSABLE ASSETS

At the current level of the proposed product line, we have defined software modules as core assets of our product line, but we have not yet defined other assets such as test cases, process description, or any other element of a software production process. In this section, we explain how we have defined reusable modules that implement the features defined in our variability model, and how we used those modules when we want to assemble products from the product line.

#### A. Core Assets Definition

We group in modules features with close functionality, where each module is a distinct (physically separated) code unit. This strategy, called compositional approach [10] provides us a direct link between each feature and its implementation, giving us high degree of feature traceability and therefore is easier to maintain [11]. In total, we defined nine modules: *People, Devices, Schedule, Calendar, Records, Application Type, Reports, Notifications*, and *Common*. Calendar, Schedule, and Notifications are optional, which means they may or not may included in the applications obtained from the product line.

---

2http://variamos.com/
The People module manages operations related to the features People and Application Users. Some of those operations are to manage user accounts, permissions, logoffs, password changes, and more. On the other hand, the functionality of the Person Identification feature is controlled recording each person directly on the biometric device.

The Device module controls the communication between the software application and the biometric devices by mean of operations for downloading records, controlling the number of users, and adding new biometric devices. This module supports the feature Device. In addition, this module also supports the features, Number of People, P 0 to 1200 and P 1200 to infinite which are useful to define how many people will be supported by the software application.

The Schedule module is in charge of managing aspects related to staff scheduling and the Calendar module manages information about holidays required by the applications. The Records module handles the registry of data delivered by the biometric devices as well as data required by each application type. This module also enables the addition of attributes to the Records table according to the particular requirements of each application. The attributes are added trough SQL statements, like ALTER TABLE, and the portions of code that manipulates the database information manage those schema differences, making them transparent to the final user. This module implements the features Records and Transactions, and their sub-features.

The module Application Type supports the three types considered in the SPL and the measure units associated with each type. The implementation of this module consist of two parts: an interface that exposes services shared by all the application types, and a particular java class for each application type that includes its own functions exposed as services. The Notifications module has the functionality for composing and sending messages and supports the feature Notification. Finally, the Common module supports functions shared among all the modules such as exceptions and dates handling, validations, currency format, and internationalization.

B. Core Assets Design and Implementation

With the aim of using, as much as possible, previous assets developed by Acople Tecnológico, to quickly changing from conventional software development to a software product line engineering approach, we decide to develop assets by using Java EE technologies, because previous assets were developed on Java EE technologies too. In addition, at coding level, we use traditional design concepts of software engineering to implement variability such as polymorphism, inheritance and reflection [12].

As each module was considered as an independent part of code, we follow the model–view–controller (MVC) pattern to design and implement each one of them. We choose this pattern because it gives us independence between the user interface of each module and its business logic.

Model and controller layers were implemented as a Java SE project, which were packaged as a .jar files, whereas the view layer was separated into two different parts, a view controller (managed bean) and a folder with .xhtml files that represents the module screens. As a result we had for each module three assets: a .jar file, a folder with .xhtml files and a .class file.

Model layer was implemented using the Data Access Object Pattern (DAO), to separate low-level data accessing operations from high-level business services. Common to all modules we defined and implemented a generic interface IDAO<T> where we declare standard database operations. Then, we defined the model entities that each module needed and we implemented a specific DAO for each entity. We reuse the most of the model entities previously developed by Acople Tecnológico, although we created custom annotations to add meta-data into the Java source code to facilitate data validations using reflection techniques.

For the controller layer, we defined a public java interface with the available services of each module and at least one Java class for implementing those services. Outside of those modules only the public operations were accessible, obtaining as a result high cohesion and low coupling among modules.

Finally, the view layer was implemented with the Java Server Faces - JSF- technology. This framework provides the file faces-config.xml which is useful to define the managed bean associated to each view and the navigation rules among the different views of each module. For instance, according to the business logic in the module Devices it is possible to access to the view createDevice.xhtml only from the view listDevices.xhtml. For each module we defined those rules to incorporate them into a general faces-config file when a new application is derived. We also created a general template file template.xhtml which defines the appearance and distribution (header, content and footer) for all the final user screens.

C. Generate product line members

In an effort to take advantage of the Acople Tecnológico’s experience all applications obtained from the product line will follow the rules defined by the Java Servlet Specification, and the general architecture presented in Figure 2. To achieve this, configuration and derivation tasks have to be done manually by following these steps:

1) Create a new Web Project by using Eclipse JEE IDE\(^3\). This project must respect same structure presented in Figure 2.
2) Create a new database in the database engine installed on the application server.
3) Select from the feature model those choices that defines client requirements and satisfy the model restrictions
4) Identify the modules that have to be included according to the features previously selected.
5) Execute the SQL statements of each selected module to create tables and populate them with their basic data when is required.
6) Move into the LIB folder the .jar files of each module previously identified and other third-party libraries required by the application.

---

\(^3\)https://eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/mars2
IV. Discussion

The development of the LPS for applications based on biometric devices could bring several benefits to Acople Tecnológico, besides several lessons have been learned from the process of designing and development the SPL. In this section both of them are discussed.

A. Benefits for the Business

Acople Tecnológico changed its methods to build applications by applying a development paradigm with better results in development time and quality. The following aspects show evidence of this improvement.

The models’ characteristics and functionality are not limited to a particular application. As examples, the component Device could be tailored to many types of biometric devices such as facial, fingerprint, iris, or voice recognition. The component Reports implements a Report Factory. As a result, each derived application can add as many reports as needed. In the same way, a generic database design allows the use of the same schema by several modules. Thus, to derive specific applications, minor changes are required. As an example, the table Records is included in Working Time, Restaurant, and Access applications to store the information about events, avoiding inconsistencies in each application type. However, Records’ attributes are identified as mandatory for some application types but optional for others. Therefore, an ALTER TABLE is applied to eliminate the NOT NULL restriction for not required attributes, according to the particular application type selected.

Furthermore, the SPL could be used to extend the portfolio of Acople Tecnológico’s applications in the future, allowing them to envision new applications since different contexts could require the same functionality. Taking advantage of this opportunity, Acople Tecnológico already generated a new product targeted to preschool institutions, allowing them to register persons authorized to pick up each child and to identify who picked them each day. In this way, the school increases the security in the process of releasing the children to their families. This new product has many similarities with the Working Time one, and its set of features is one of the 320 combinations produced by the Variamos analysis.

In the same way, the SPL eases the creation of any of the three products including particular functionality required by a specific client. This was the case of a new variation of the Working Time application. Before the implementation of SPL methodologies, a particular request was attended by duplicating the application code—in this case, Working Time—and making further modifications over it. With the SPL, the basic application functionality is coded in libraries that are added to a new project, then specific functionalities are treated as new features, generating new libraries to implement them. Eventually, those new features could be included in the features model being available to use on new types of products.

Moreover, with the aim to facilitate the inclusion of new features or application types, the modules design, and implementation involved extension points. As an example, the component Marks has attributes to identify the application type and extra attributes to manage diverse data types (numeric, alphanumeric, or boolean).

Thus far, the SPL also facilitated the generation of a third new product that integrates the functionality of Working Time, Restaurant, and Access applications. With this product Acople Tecnológico served one client that wanted to use the three products. Before the SPL implementation, three different applications would have been sold and installed.

B. Lessons Learned

Several lessons have been learned during the analysis, design, and implementation of the SPL, among them, are the...
The features model allowed us to abstract the problems and to focus development efforts on reusing artifacts, leading to high quality and functionality, reducing the development time and giving quicker customer response. This improves the company’s business opportunity.

The variability model was useful to identify applications’ modules. It includes business rules that guarantee high cohesion and low coupling, easing the reuse of components.

The modules’ dependencies definition was crucial to avoid circular references and to achieve low coupling. Besides dependencies point out the components’ implementation order. Moreover, as far as possible, a product derived from the SPL will contain only the logic associated with the features the product comprises.

Even tough in our particular case the 320 feature combinations could be analyzed to select a set of new products that could bring the Acople Tecnol´ogico’s market attention, generally that is not a viable task due to the vast number of possible combinations. However, having the combinations is valuable since it allows to verify if a particular set of product requirements is valid in the LPS. In fact, when a new client wants a product from the company, Acople Tecnológi­co can easily identify what requirements are covered by the product line and therefore could better estimate the cost and the development time required to satisfy them. If there are requirements that cannot be satisfied through the reuse of assets previously developed into the product line, the company negotiates and develops those requirements like a product-specific addition.

The development of an LPS in an SME could bring some advantages, among them, the products are small like the organization itself, in the sense that they are composed of few components. This facilitates the analysis, design, and development stages, as well as the transition from the methodologies applied before to the new ones. However, there are also some disadvantages; the main one is that the organization have few human resources. Thus, maintaining the operation along with the development of the LSP is a demanding task.

Finally, the most complex task was to design the modules, maintaining the abstraction while increasing reusing. The experience of Acople Tecnológi­co in attending the biometric applications market was a key aspect in the definition of generic modules. The design facilitated the codification task since it provides a better comprehension of components’ functionality. Thus, since benefits are greater than the cost of design and implementation of the LPS, Acople Tecnológi­co is planning to develop, in the near time, an LPS for the second set of products in their portfolio, related with auto service portals directed to employees, clients, and providers, among others.

V. RELATED WORKS

In recent years the industry has reported several experiences regarding the implementation of Software Product Lines. The Product Line Hall of Fame lists the best SPL industrial applications presented in the Software Product Line Conference over the years. Among them, Siemens Healthcare [4] combined SPL with agile development to overcome some challenges detected in the development of imaging products. Those products, ranging from medical scanners to visualization systems, share medical imaging functionality, including different image modalities, data types, analysis algorithms and manipulation concepts. The U.S. Army Live Training Transformation and the U.S. Navy command and control systems of Naval surface combatants [3] are two SPLs that bring improvements in quality, time to deployment, cost, and engineering productivity.

Linden et al. [13] collect the experiences of eight companies, including Nokia, Bosch, Philips and Siemens, describing the main effects of the LPS adoption. Additionally, Koziolek et al. [5] report the experiences from executing domain and economic analysis in more than 20 software systems belonging to four application cases in the ABB engineering company and concluded that SPL should be applied in only one of the four cases. They identified feature domains, subdomains and components with the potential to constitute a SPL. Pohl et al. [1] present another fifteen successful implementations. The authors highlight the costs and development time reduction, as well as the improvement of product quality. Birk et al. [2] analyzed the SPL practices used by five organizations, among them Hewlett-Packard and Bosch. Laguna y Hernández [14] report the application of methods and techniques of the Product Line Engineering in the domain of e-commerce software and describe the modeling, configuration, design and implementation stages. Sanchez et al. [15] report the development of an SPL to generate e-learning web mining products that include variations of three major features: e-learning platform, queries, and data mining suite. The authors describe the analysis of the domain and the application engineering processes. Parra et al. [6] describe the implementation of SPLs in Heinsohn Business Technology. They developed a SOA modular architecture to support the adoption of a SPL over a set of JEE artifacts already developed for different domain applications.

Even though SPL has been applied in several domains, most studies reported in literature refers to projects carried out in large companies. In contrast, in this paper, we report the experience and lessons learned from the SPL implementation in a small organization. Besides, different to other cases, in this work, we adopted an extractive approach that took advantage of the Acople Tecnológi­co experience in the market. Moreover, even though biometric recognition is included as a feature in many SLP examples [16], [17], to the best of our knowledge, there is no report of an SPL including biometric devices for authentication with an aim different to software functionality usage control. The software products obtained by the SLP presented in this paper, use the authentication for other purposes such as working time or restaurant consumption control.

VI. CONCLUSIONS AND FUTURE WORK

This paper has presented how a set of products that use biometric devices have been re-factored, using an extractive approach, into a Software Product Line. The crucial aim of introducing the product line approach in Acople Tecnológi­co was to improve reuse in a planned way, which means a

---

4http://splc.net/fame.html
reduction in the time and cost of developing new products related.

The approach developed in this paper presents the experiences and lessons learned from applying SPL in a particular industrial setting. Doing so, we define a variability model expressed as a feature model that was inspired by a real case, and therefore may be used as a reference for future researches. Furthermore, we explain our general strategy to develop reusable components based on a compositional approach and the benefits and lessons learned from applying software product lines into an small enterprise.

So far we have come a long way in the implementation of our product line, but we are aware that there are other steps to follow in order to finish our goal. As a next step, we will continue extending our proposal defining a strategy for configuring and assembling as automatic as we can the modules that we have developed to create new products from the product line. We have considered tools as Gradle or Maven to turn automatic the assembly process. Moreover, we are interested on incorporate non-functional requirements such as security and availability into the SPL development to improve the current software assets.

ACKNOWLEDGMENTS

This work has been partially supported by the Pontificia Universidad Javeriana Cali, Colombia, under the program COFINPRO2014 which sponsors the Project No. 4127.

REFERENCES


