Identifying Features for Ground Vehicles Software Product Lines by Means of Annotated Models

Rafael Durelli
University of São Paulo
26 PUBLICATIONS **37** CITATIONS
SEE PROFILE

Ricardo Ramos
Universidade Federal do Vale do São Francisco (UNIVASF)
47 PUBLICATIONS **126** CITATIONS
SEE PROFILE

Oscar Pastor
Universitat Politècnica de València
494 PUBLICATIONS **4,275** CITATIONS
SEE PROFILE

Valter Vieira de Camargo
Universidade Federal de São Carlos
48 PUBLICATIONS **88** CITATIONS
SEE PROFILE

Identifying Features for Ground Vehicles Software Product Lines by Means of Annotated Models

Rafael S. Durelli[1,†,‡], Daniel B. F. Conrado[1,†], Ricardo Argenton Ramos[2,§], Oscar Lopez Pastor[3,§], Valter V. de Camargo[1,†,§], and Rosângela A. D. Penteado[1,†,§]

[1] Computing Dept., Federal University of São Carlos, São Carlos – São Paulo – Brazil
{rafael_durelli,daniel_conrado,valter,rosangela}@dc.ufscar.br
[2] Collegiate of Computer Engineering, Federal University of Vale do São Francisco, Juazeiro – Bahia – Brazil
ricargentonramos@gmail.com
[3] Dept. of Computer Systems and Computation, Universidad Politécnica de Valencia, Valencia – Spain
opastor@dsic.upv.es

Abstract. An approach for the identification of features supported by class models annotated with stereotypes is shown in this paper. The models are automatically reverse engineered by a tool called Rejasp/Dmasp where attributes and methods are stereotyped if they have some relation with candidate features. The approach consists of four guidelines and focuses on identifying features in embedded systems of ground vehicles. As a preliminary evaluation, the guidelines were applied in creating a product line in the domain of ground vehicles.

Keywords: Software Product Line, Embedded Systems, Ground Vehicles

1 Introduction

Software Product Line (SPL) enables systems to be developed quickly through the composition of reusable artifacts [2], that is, in other words, the software – a product line member or product – is developed by composing features of a specific domain [7]. Features are abstractions of design and code that represent the variability of a domain and may be optional, alternative or mandatory.

In general, the development of Embedded Systems (ES) is not supported by systematic techniques of reuse, leading to bad time-to-market and low quality of products. Previous studies have explored the use of SPL techniques for developing embedded systems aiming at increasing productivity and quality of these systems [3,4,6]. However none of these papers present clear guidelines or support tools for the agile identification of features for rapid development of SPL in a fast way [5].

Most researchers in the literature do not provide explicit guidelines for the identification of features to build product lines of ES. Recent research such as Mohan et al [5]

† Financial support provided by CNPq and FAPESP under INCT-SEC processes 573963/2008-8 and 08/57870-9 respectly.
‡ Financial support provided by Capes.
§ Financial support provided by CNPq – Projeto Universal – Process n° 483106/2009-7.

recognizes the need to integrate the product line engineering with agile methods, such as XP [1]. The authors state that the time-to-market is less each day and techniques that facilitate the rapid engineering of a product line are extremely important. However, they do not present clear and systematic guidelines that can be easily replicated for identifying features from a set of products previously developed.

Kim [3], Lee et al [4] and Polzer et al [6] use techniques of SPL to aid the development of ES, however, did not have clear guidelines to identify the characteristics.

This paper presents an approach that consists of four guidelines that support the identification of features for the agile construction in of SPL for ground vehicle (GV) domain. The main contribution consists in an alternative approach for features identification based on analyzing models rather than analyzing only source code or trust in knowledge of the domain. We argue that the identification of features based on models is easier than when it is conducted only base on experience and analysis of the source code of existing systems. Although the approach is composed of four guidelines, only the second one is commented in a more detailed way due to space limitations.

2 Agile Approach for Derivation of LPS for Embedded Systems

Figure 1 depicts schematically the process of applying the four proposed guidelines for identifying features in ES domain. Each "***Gn***" acronym represents a guideline.

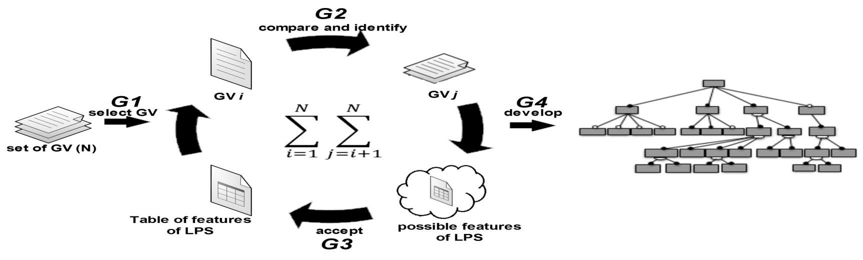


Fig. 1. Guidelines for identifying features of SPL.

The first step to start the process is to choose a particular domain in which the SPL must be built, for example, ground vehicles or unmanned aerial vehicles. Next, the "***G1-Select GV***" guideline consists of obtaining a set of GVs. At least three systems in the domain must be selected. In our case study, four systems for GVs that use many devices were obtained from an Internet Repository[4]. One of them, called BumperCar, has the responsibility to avoid collisions with obstacles, thus it uses some types of sensors such as ultrasonic and touch. The second one, called Explorer, has the capability to exploit a specific environment. The third one, called Forklift is responsible for pick up a particular object and carry it to another place, this GV is controlled by the user by means of Bluetooth protocol. The forth follow the same pattern.

When G1 is done, each of the systems must be labeled with numbers ranging from 1 to *N*. For each GV, it is performed one cycle, as shown in Figure 1. For instance,

[4] `http://www.nxtprograms.com/projects2.html`

the GVs of our case study have been enumerated from 1 to 4 so that the first GV was compared with GVs 2, 3, and 4; the second GV with the GVs 3 and 4 and the third was compared with the fourth GV.

The guideline "***G2-Compare and Identify***" is the most important of the approach. It states that GVi should have its hardware (sensors and actuator) and software compared to the others GVsj, where $i+1 \leq j \leq N$, in order to identify features in this domain. This comparison is supported by a tool called Rejasp/Dmasp, that aims to recover annotated (stereotyped) class models from source code of systems, where the stereotypes means indications of candidate features. These indications are evident in the models through stereotypes, that is, attributes and methods that have some key words representing a "concept" of the domain are stereotyped. This tool has a "Concept Manager" where the "domain concepts" which must be mined in the source code of systems can be registered. Thus, we can register the "domain concepts" that must be searched in the source code. Domain concepts are words that represent information relevant to the domain, for instance, the concepts "motor" and "sensor" are considered important terms in the field of GV and can be features of a SPL in this domain. These concepts must be obtained through experience of the developers in the domain or through existing ontologies.

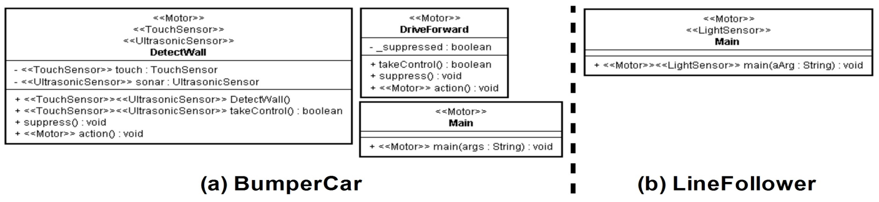
Figure 2 depicts parts of class diagrams generated by Rejasp/Dmasp based on source code of two GVs (*BumperCar* and *LineFollower*) used as our case study. Stereotypes only appear upper class names when either an attribute or method has been identified as an occurrence of the underlying Concept Domain (Candidate Feature). So if a class has the stereotype ≪*Motor*≫ it is because some attribute/method also has that stereotype.

As can be seen, the stereotype ≪*Motor*≫ is presented in all classes. Due to that, the concept "Motor" possibly will be indicated as a mandatory feature. However the stereotype ≪*TouchSensor*≫, ≪*UltraSonicSensor*≫ and ≪*LightSensor*≫ do not appear in all classes, which means that each system has different types of sensors and possibly they will be classified as alternative or optional features. We argue that identify features only based on an analysis of the source codes is an expensive and time consuming task. Thus, this tool reduces complexity and improves the productivity of the task of identifying features of SPL.

After the process of identifying features it must be created an artifact called Table of Candidate Features, as shown in Table 1, wherein, at first, all the concepts identified in the class models must be inserted. In the next guideline, these candidate features will be analyzed in order to decide if they can be considered final or relevant feature of the domain. It is worth to mention that the tool can annotate (stereotype) methods and attributes that are not features, generating false-positives. It is also important to point out that the quality of the process of identifying features is completely dependent on the quality of the concepts registered with the Concept Manager.

An important detail is that the identification coverage (how much the tool manages to identify all correct features) can be higher if we use the tool incrementaly. For example, if some features are not present in the first retrieved model, we can update the Concept Manager including new Domain Concepts and run the tool again to retrieve a new model that has a higher coverage. Then, this process can be repeated until most of the features have been stereotyped in the model.

In the "***G3-Accept***" guideline, one must analyze and classify the features of the Table of Candidate Features. The analysis consists in deciding if a feature must be considered as a "relevant feature" of the domain. This decision process must be supported by the domain engineer's knowledge and other information sources like sensor's manual and API documentation. Furthermore, the selected features must be classified in mandatories, optionals or alternatives; however, it's beyond the scope of this paper. The final step is to create a new artifact called Table of SPL Features shown in Table 2 which contains all relevant features and the type of them. Table 2 contains a subset of the relevant features for the SPL of our case study.



<<Motor>>
<<TouchSensor>>
<<UltrasonicSensor>>
DetectWall
- <<TouchSensor>> touch : TouchSensor
- <<UltrasonicSensor>> sonar : UltrasonicSensor
+ <<TouchSensor>><<UltrasonicSensor>> DetectWall()
+ <<TouchSensor>><<UltrasonicSensor>> takeControl() : boolean
+ suppress() : void
+ <<Motor>> action() : void
<<Motor>>
DriveForward
- _suppressed : boolean
+ takeControl() : boolean
+ suppress() : void
+ <<Motor>> action() : void
<<Motor>>
Main
+ <<Motor>> main(args : String) : void
(a) BumperCar
<<Motor>>
<<LightSensor>>
Main
+ <<Motor>><<LightSensor>> main(aArg : String) : void
(b) LineFollower

Fig. 2. Candidate Features

Table 1. Table of candidate features.

Systems	Stereotypes	Candidate Features
	≪Motor≫	Motor
BumperCar	≪TouchSensor≫	Touch
	≪UltraSonicSensor≫	UltraSonic
LineFollower	≪Motor≫	Motor
	≪LightSensor≫	Light

Table 2. Table of SPL Features

Candidate Concept/Feature	Description	Type
Motor	Represents the existence of motors.	Mandatory
Sensor	A physical stimuli detection device.	Optional
UltraSonic	Measures its proximity to an object	Optional
Touch	Detects collisions	Optional
Light	Measure light intensity	Optional

After guidelines G2 and G3 have been applied, the Feature Model must be created using the guideline "***G4-Develop***". The Table of SPL Features that was generated in "***G3-Accept***" supports the Feature Model creation. Figure 3 depicts the Feature Model of our case study. This guideline is also beyond the scope of this paper and will not be detailed.

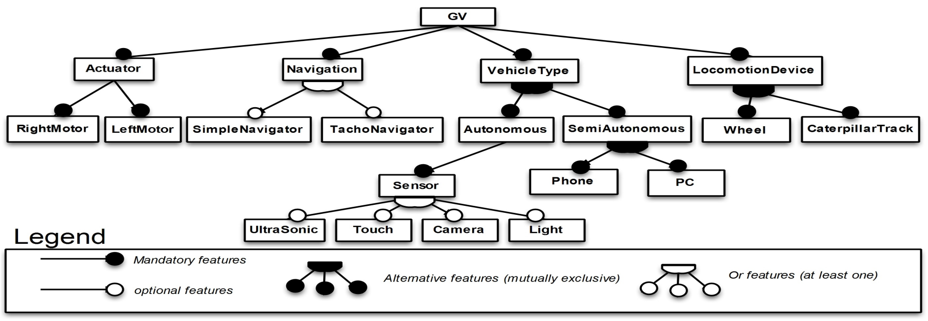


Fig. 3. The developed Feature Model.

3 Final Remarks

We argue that identify features only by means of experience and existing source code is more costly and error prone than using a model-based approach like the one presented by us. In our approach, models assist in an agile identification of domain concepts in several systems of a domain, which makes the identification of features a more controlled and productive task. The quality of the identified features depends on the set of concepts previously registered in Concept Manager of Rejasp/Dmasp. Therefore, we suggest registering a concept list or a domain's ontology in the tool.

We also argue that top-down strategies for feature identification, that is, those that identify features by analyzing a certain domain instead of systems previously developed, are not suitable when the SPL must be created in a short time. The main cause is that analyzing a domain usually takes a long time to be finished and yields a wide range of features that are not relevant or that may never be used to derive products from the PL.

As a future work, we intend to improve the proposed guidelines in order to be applied in existing agile methods like XP or SCRUM.

References

1. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change (2nd Edition). Addison-Wesley Professional (2004)
2. Clements, P., Northrop, L.: Software Product Lines. Addison-Wesley (2002)
3. Kim, H.K.: Applying product line to the embedded systems. In: Computational Science and Its Applications - ICCSA 2006. Springer (May 2006)
4. Lee, J., Cho, J.H., Ham, D.H., Kim, J.S.: Methodology for embedded system development based on product line. vol. 2, pp. 920 –923 (2005)
5. Mohan, K., Ramesh, B., Sugumaran, V.: Integrating software product line engineering and agile development. Software, IEEE 27(3), 48 –55 (may 2010)
6. Polzer, A., Kowalewski, S., Botterweck, G.: Applying software product line techniques in model-based embedded systems engineering. In: MOMPES '09: Proceedings of the 2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software. pp. 2–10. IEEE Computer Society, Washington, DC, USA (2009)
7. Weiss, D.M., Chi: Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley Professional; Har/Cdr edition (1999)