JOURNAL OF OBJECT TECHNOLOGY
Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering. ©JOT, 2005

Vol. 4, No. 3
Special issue: 6th GPCE Young Researchers Workshop 2004

Identifying Variations in Mobile Devices

Vander Alves, Informatics Center, Federal University of Pernambuco, Brazil

Product lines promise to improve software quality and development productivity. A central issue to meet this is systematically dealing with variations within products in a certain domain. Although various techniques are available, their use is constrained by the specific application domain. This paper presents an extractive method for handling variation in the mobile device application domain. Being based on Aspect-Oriented Programming and general program transformation, it offers enhanced configurability and composability. We illustrate the approach with a game product line built with J2ME.

1 INTRODUCTION

As software demands grow in various domains with ever-increasing quality standards and shorter development cycles, software product lines have emerged as a promise to not only improve quality and lower development costs, but also to reduce time-to-market. A software product line consists of a set of products developed from the same set of artifacts and targeted at a specific domain [2]. Indeed, by moving to the product line approach, an organization may increase its competitiveness in a specific domain of software development. At the same time, adopting this approach involves both organizational and technical commitments.

At the technical level, the product line approach to software development encompasses the following activities: domain analysis, domain design, domain implementation, and application engineering [4]. In the first three, a set of reusable artifacts is developed; in the fourth activity, these artifacts are potentially adapted and composed with custom-specific artifacts in building applications.

A central issue in this process is managing the communalities and the variabilities among the products [9]. It is a non-trivial task, due to the potentially high number of product composition instances. Indeed, to accomplish this task various techniques at the implementation level exist [1]. Their use, however, is constrained by the application domain.

In this context, this paper reports on-going work to define an extractive approach for managing variability at the implementation level. It relies on a combination of Aspect-Oriented Programming (AOP) [7] and general program transformation in order to structure and instantiate product lines, and it aims at providing enhanced configurability and composability in the mobile device domain.

The remainder of this paper is organized as follows. Section 2 describes the

Cite this article as follows: Vander Alves: "Identifying Variations in Mobile Devices", in *Journal of Object Technology*, vol. 4, no. 3, April 2005, Special issue: 6th GPCE Young Researchers Workshop 2004, pp. 51–56, http://www.jot.fm/issues/issue_2005_04/article7

application domain and the specific product line which is the subject of our approach; in Section 3, we explain our method, showing how it combines AOP and general program transformation; Section 4 evaluates the approach and discusses related work; we conclude in Section 5.

2 DOMAIN DESCRIPTION

Game development for mobile devices is an example of a domain with business and technical constraints for which the product line approach is likely to be suitable. Numerous functional variations are possible on a single game type, and each game may have to be deployed in a dozen of platforms. In addition, the development cycle must be tuned so that short time-to-market and high quality standards are met, since these games may be delivered in millions of devices.

We consider game development for mobile phones using J2ME's MIDP 1.0 profile, which is targeted at mobile devices with constrained resources, including reduced memory and computing power, and intermittent low-bandwidth connectivity [11]. Although MIDP 1.0 is supported by a number of devices, some still make extensive use of proprietary Application Programming Interface (API), which explores devices enhancements, such as advanced graphics manipulation, for instance.

In our scope, we explore the platform variation arising due to use of proprietary API. In particular, there are three platforms (P_A, P_B, and P_C) on which the same game GM is run (the actual names are not relevant here). P_A relies solely on MIDP 1.0, whereas P_B and P_C rely on MIDP 1.0 and proprietary API. GM is an actual game delivered by service carries in South America and Asia. Figure 1 illustrates its main screen.

Figure 1: Platform variation of the GM game

Negative		Positive	
Scattered	Local	Scattered	Local
pointcut and advice with empty code	inter-type declaration defining empty method	pointcut and advice defining variation code	inter-type declaration defining constant or variant method

Table 1: Mapping variation type to aspect constructs

3 APPROACH

The goal is to structure a product line around GM so that it can be easily configured for any of the platforms P_A, P_B, or P_C. Indeed, as we discuss in Section 4, such task could be accomplished solely with object-oriented constructs or other techniques, but these lack enhanced composability of relevant features in this domain. In order to accomplish this task, we rely instead on a combination of AOP and general program transformation.

The outline of our approach is as follows: given GM in P_A, we identify variation points, optionally refactor code to encapsulate these variation points, and extract the specialized behavior into aspects in AspectJ. The outcome is an aspect for introducing the specifics of each platform and an abstract GM, which we refer to as GM_{Abs}. Figure 2 illustrates our approach.

The identification of specific variation points related to platform variation is carried out manually, but this can be improved as Section 4 explains. When refactoring the original code to extract variation, extensive use is made of Extract Method. However, no refactoring is performed when the variation is scattered. At this point, we create an empty aspect to which we map the extracted variation. The precise mapping depends on the variation type and whether the variation is scattered or not. Table 1 defines this mapping.

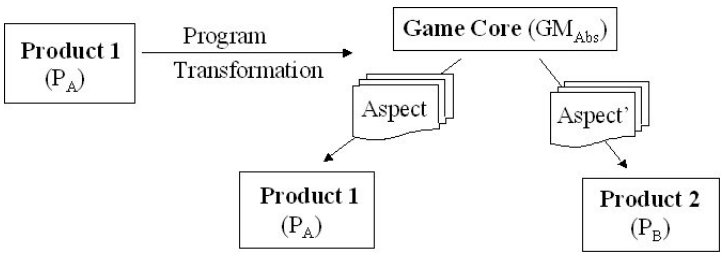

Figure 2: Approach outline

Essentially, if the variation is scattered, we rely on dynamic crosscutting by using an expressive pointcut and a piece of advice with either some code (positive variation) or none (negative variation [3]). If the variation is not scattered, we rely on static crosscutting by using an intertype declaration defining the method or the constant for the variation (posivite) or defining an empty method (negative variation).

The approach is reactive and incremental: from one game in one platform, we arrive at a product line infrastructure containing the abstract game GM_{Abs} and two aspects, one for customizing GM_{Abs} back to the original platform and another for customizing GM_{Abs} to the new platform. From there, we apply the method once again to add an aspect for another platform.

4 EVALUATION AND RELATED WORK

The previously described approach was applied in structuring a product line of game GM in three platforms. We started from P_A and applied the method twice in order to develop the product line infrastructure. The motivation for applying this specific sequence was that the development team used to develop the products in this order, for the reason that P_A uses no proprietary API, whereas the other platforms do. However, the method can be used for other sequences as well.

Following the process in this domain, we noticed that most variation points were considerably fine-grained and consisted of scattered method calls to proprietary API and of different arguments being specified for drawing method calls. Indeed, only a few variation points were coarse-grained, such as changing the type hierarchy of the GM's main screen class. In addition, the task of finding specific platform variation points was performed manually. This, however, could be automated partially with the help of aspect mining tools [6].

When extracting variant code into an aspect, either dynamically or statically according to Table 1, we noticed that some replicated code remain in the aspects. This is not surprising because the platforms are similar, despite their API differences. On the other hand, this suggests that some generic mechanism should be incorporated into the approach. We might extend ours to use generic aspects [8] or hybrid frames [10].

Furthermore, the special case of mapping scattered variation into pointcut and advices may lead to noticeable increase in bytecode size. Since the game products run on devices with constrained resources, this is an issue which must be addressed. One way would be to identify more optimized pointcuts. Another possibility would be to rely on more static approaches to handle crosscutting. To this end, we are considering integrating the use of more general program transformation engines, such as JaTS [12].

The reactive and incremental approach implies that the abstract product line artifact, GM_{Abs}, evolves whenever a new game is added into the infrastructure, since GM_{Abs} may have to be refactored to expose some customized behavior for the new product. Because this happens, there might be a chance that one aspect, customizing GM_{Abs} for another previously incorporated game, need to be adapted. This interference can be noticed by tools which show aspects customizing GM_{Abs} and could be handled semi-automatically by aspect-aware refactorings [5].

Indeed, the product line infrastructure could be developed solely with object-oriented constructs or with other techniques such as multi-paradigm design [3] or generative programming [4], but we realized that some extensively used platform related features, such as flipping, were considerably scattered or tangled with other concerns in the implementation. To achieve enhanced configurability and composability, aspect-orientation was more appropriate for modeling them in the solution space. Nevertheless, as noted previously, aspects need to be combined with parameterization mechanisms in order to be generic and thus foster reuse. In fact, such combination is an instance of multi-paradigm design.

5 CONCLUSION

Structuring a product line requires more than relying on a set of techniques. A method should be employed for this task and this depends not only on the techniques, but also on the application domain. In this context, this paper has presented on-going work to define an extractive approach for structuring a product line in the mobile device domain. By relying on AOP and program transformation, it has been possible to effectively and modularly factor out platform variation into aspects and later compose them with the application core. In addition, the approach has been applied to non-trivial real applications and suggested some points for further improvement.

REFERENCES

[1] Michalis Anastasopoulos and Cristina Gacek. Implementing product line variabilities. In *Symposium on Software Reusability*. ACM Press, May 2001.

[2] Paul Clements and Linda M. Northrop. *Software Product Lines : Practices and Patterns*. Addison-Wesley, 2001.

[3] James O. Coplien. *Multiparadigm Design For C++*. Addison-Wesley, 1998.

[4] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[5] Oberschulte C. Hanenberg S. and Unland R. Refactoring of aspect-oriented software. In *4th Annual International Conference on Object-Oriented and Internet-based Technologies,Concepts, and Applications for a Networked World (Net.ObjectDays)*, Erfurt, Germany, September 2003.

[6] Jan Hannemann and Gregor Kiczales. Overcoming the prevalent decomposition in legacy code. In *Workshop on Advanced Separation of Concerns in Software Engineering at ICSE 2001*, Toronto, Canada, May 2001.

[7] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect–Oriented Programming. In *European Conference on Object–Oriented Programming, ECOOP'97*, LNCS 1241, pages 220–242, Finland, June 1997. Springer–Verlag.

[8] Günter Kniesel and Tobias Rho. Evolvable pattern implementations need generic aspects. In *ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution*, Oslo, Norway, June 2004.

[9] Charles Krueger. Variation management for software production lines. In *Proceedings of the 2nd International Software Product Line Conference*, pages 37–48, San Diego, California, August 2002.

[10] Neil Loughran and Awais Rashid. Framed aspects : Supporting variability and configurability for aop. In *International Conference on Software Reuse (ICSR-8)*, Madrid, Spain, July 2004.

[11] Sun Microsystems. *JSR-000037 Mobile Information Device Profile (MIDP)*. World Wide Web, http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html, 2000.

[12] Federal University of Pernambuco. *JaTS - Java Transformation System*. World Wide Web, http://www.cin.ufpe.br/ jats/, 2001.

ABOUT THE AUTHORS

Vander Alves is a PhD student in Software Engineering at Federal University of Pernambuco, Brazil. He can be reached at vra@cin.ufpe.br. See also http://www.cin.ufpe.br/ vra.