An Incremental Aspect-Oriented Product Line Method for J2ME Game Development

Position Paper

Vander Alves
vra@cin.ufpe.br

Pedro Matos Jr.
poamj@cin.ufpe.br

Paulo Borba
phmb@cin.ufpe.br

Informatics Center
Federal University of Pernambuco
Recife, Pernambuco, Brazil

1. INTRODUCTION

Game development for mobile devices must address a significantly high number of variations. These arise due to variable device capability, which implies constraining application features to meet available resources in specific devices. Managing such variation thus play a key role in the development process, for which the product line approach is suitable. However, in practice, organizations will not shift to this approach from scratch; there are frequently some existing products which should be integrated into the approach. Nevertheless, guidance in this direction is still rare.

In this context, this paper describes an incremental process for structuring a product line from existing mobile device game applications. The process relies on continuous refactoring and Aspect-Oriented Programming to isolate communality from variability latent within the products. We applied this process to an initial set of three industrial-strength games and evolved them into a single product line. Thereafter, evolution was performed on the single product line rather than on the individual products.

2. DOMAIN DESCRIPTION

Game development for mobile devices is a domain with business and technical constraints for which the product line approach is likely to be suitable. Numerous functional variations are possible for a single game type, and each game may have to be deployed in a dozen of devices. Additionally, device selection may constrain application features due to limited resources.

We address 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 [4]. Although MIDP 1.0 is supported by a number of devices, a considerable number of these still make extensive use of proprietary Application Programming Interface (API), exploring device enhancements, such as advanced graphics and sound manipulation. Moreover, there is still variation related to memory and computing power, which impacts on application features. The resulting software is thus highly variant.

Within our scope, we explore the platform variation arising due to use of proprietary API and limited memory. In particular, we consider three platforms (P_A, P_B, and P_C) on which the same game GM is run. P_A relies solely on MIDP 1.0, whereas P_B and P_C rely on MIDP 1.0 and proprietary API. Further, P_C constrains bytecode size to half of the other platforms. GM is an industrial-strength game, whose main screen is illustrated by Figure 2.

Figure 1: Platform variation of the GM game

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. The outline of our approach is as follows. First, given GM in P_A and P_B, we identify variation points, refactor

code to encapsulate these points, and extract the specialized behavior into AspectJ aspects. The outcome is an aspect for introducing the specifics of each of these two platforms and an abstract GM, which we refer to as GM_{Abs}. Figure 2 illustrates the first step of our approach.

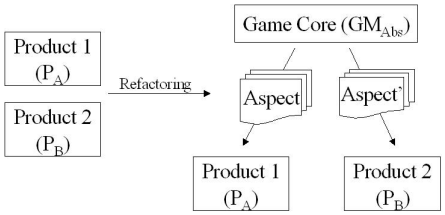

Figure 2: Approach outline

Next, the resulting product line is considered with the remaining product, and we reapply the previously described procedure. The result is a new product line encompassing all three products.

The identification of specific variation points related to platform variation is carried out with help of a diff-like tool, but this can be improved as Section 4 explains. When refactoring the original code to extract variation, extensive use is made of the Extract Method refactoring. However, in some cases we directly apply object-to-aspect refactorings: with fine-grained variations such as constant and attribute definitions, and with scattered method calls. In either case, the variation is mapped directly into an aspect construct, such as inter-type declaration or pointcut-advice.

The approach is incremental: from a game independently developed in two platforms, 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. At this stage, we apply the method once again to incorporate another platform into the product line infrastructure, which also results in a new customizing aspect for this platform.

4. EVALUATION

The previously described approach was applied in structuring a product line of game GM in three platforms. We started from GM independently deployed in P_A, P_B , and P_C, and applied the method twice in order to develop the product line infrastructure.

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 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 with help of diff-like tools. This, however, could be automated partially with the help of aspect mining tools [2].

When extracting variant code into an aspect, we noticed that some replicated code still remained 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 [3].

Furthermore, the special case of mapping scattered variation into pointcut and advices may lead to noticeable increase in bytecode size. In fact, this could double bytecode size of a single class. Since the games run on devices with constrained resources, this is an issue which must be addressed. We addressed this by identifying more optimized pointcuts, which prompted only a 8% bytecode size increase, and in this case the application could still fit into the device. 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 [5].

The 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 [1].

5. CONCLUSION

In practice, moving to the product line approach means dealing with existing products, which in the game development domain, can be highly variant due to platform variation, functional variation, and the synergy between both. This paper presents an effective migration method to the product line approach. By relying on AOP and refactoring, 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 industrial-strength applications and points for further improvement have been identified.

6. REFERENCES

[1] O. C. Hanenberg S. and U. R. Refactoring of aspect-oriented software. In *Net.ObjectDays*, Erfurt, Germany, September 2003.

[2] J. Hannemann and G. 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.

[3] G. Kniesel and T. Rho. Evolvable pattern implementations need generic aspects. In *ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution*, Oslo, Norway, June 2004.

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

[5] F. U. of Pernambuco. *JaTS - Java Transformation System*. World Wide Web, http://www.cin.ufpe.br/~jats/, 2001.