C# Partial Classes as a Mechanisms to Implement Feature-Oriented Designs: An Exploratory Study*

Lidia Fuentes
Dpto. Lenguajes y Ciencias de la Computación
Universidad de Málaga
(Spain)
lff@lcc.uma.es

Elio López
Dpto. Lenguajes y Ciencias de la Computación
Universidad de Málaga
(Spain)
ealopezs@gmail.com

Pablo Sánchez
Dpto. Matemáticas, Estadística y Computación
Universidad de Cantabria
(Spain)
p.sanchez@unican.es

ABSTRACT

C# partial classes allows developers to divide the implementation of a class into several slices where each slice contains an increment of functionality as compared to the other slices. Thus, combining different set of slices, we can get classes with a variable range of functionality. With this description, C# partial classes seems to be, as also pointed out by other authors, a suitable mechanism for implementing feature-oriented designs. This paper explores this idea, by systematically applying C# to a feature-oriented decomposition based on an industrial case study and comparing the results we previously obtained using the feature-oriented language CaesarJ. As main contributions, (1) we identify benefits and pitfalls of C# partial classes for implementing feature-oriented decompositions; and (2) we outline potential solutions to alleviate these pitfalls.

Categories and Subject Descriptors

D2.3 [**Software-Engineering**]: Coding Tools and Techniques

General Terms

Experimentation, Languages

Keywords

Partial Classes, Feature-Oriented Programming, Software-Product Line, C#

1. INTRODUCTION

Feature-Oriented Programming (FOP) [8] is a relatively recent paradigm for programming software systems by composing software modules, which are called *features*. A feature is considered as an increment on functionality, usually with a coherent purpose, of a software system [3, 10].

C# partial classes [1] allows developers to split the implementation of a class into a set of of files, each one containing a slice of, or an increment on, the class functionality. Therefore, as already identified by other authors [6], C# partial classes seems a suitable mechanism to implement feature-oriented designs keeping feature implementations well-modularized and appropriately separated. Partial classes can also be found in other modern programming languages, such as Ruby.

Nevertheless, although this idea seems initially promising, it has not been explored in depth. As a consequence, the community lacks of empirical evidence about the strengthens and weaknesses of partial classes as a mechanism to achieve feature-orientation at the code level.

This paper provides an exploratory study on this topic, where C# partial classes are applied to the implementation of a feature-oriented design of an industrial case study, more specifically, to the design of a Smart Home Software Product Line [5, 4, 9, 7] [1].

We have had used this case study during 3 years in the context of the AMPLE project and we have already produced a feature-oriented design, using UML packages and merge relationships [7, 4] and a feature-oriented implementation in CaesarJ [2]. This case study covers different kinds of variability [9].

In our experiment, we have tried to derive a feature-oriented implementation from the same feature-oriented design we have used previously. Then, we have compared the obtained result with the feature-oriented implementation in CaesarJ we already had. Finally, by comparing both implementations, we have identified strengthens and weaknesses of C# partial classes as mechanism to implement feature-oriented designs.

To check the results obtained in this experiment are meaningful and valid, we have also used C# partial classes to a second industrial case study, called Sales Scenario [2], which is in charge of sales processing in a Customer Relationship Management (CRM) system. We have checked the strengthens and weaknesses identified for the SmartHome case study also appear in the Sales Scenario case study.

After this introduction, this paper is structured as fol-

*This work has been supported by the Spanish Ministry Project TIN2008-01942/TIN, the EC STREP Project AMPLE IST-033710, and the Junta de Andalucía regional project FarmWare TIC-5231.

FOSD'10, October 13, 2010, Eindhoven (The Netherlands)

[1] `http://personales.unican.es/sanchezbp/CaseStrudies/SmartHome`
[2] `http://www.feasiple.de/description/bsp_ss_en.html`

lows: Section 2 introduces briefly the Smart Home case study. Section 3 identified desirable language facilities which have found useful when implementing feature.oriented designs. Section 4 describes the results obtained from our exploratory study using C# partial classes. Finally, Section 6 outlines some conclusions and future work.

2. CASE STUDY: A SMART HOME SOFTWARE PRODUCT LINE

Copia y pega de la descripción de la Smart Home más breve explicación de los modelos arquitectónicos. La figura del modelo hay que ponerla después de escribir el artículo, para que contenga todos los elementos que nos hacen falta para explicar los fallos de C#.

3. WHAT WE WANT TO FIND IN A FEATURE-ORIENTED PROGRAMMING LANGUAGE

Básicamente, características que tienen los lenguajes como CaesarJ que nos facilitan la vida para implementar modelos de SPL, tales como lo de reescritura automática de las dependencias entre clases.

4. IMPLEMENTING THE SMART HOME MODELS USING C# PARTIAL CLASSES

Describir como se implementan diferentes situaciones en el modelo de la SmartHome.

4.1 Scenarios 1: ...

Descripción de como implementar algo tal como una feature que extiende de otra.

4.2 Scenarios 2: ...

Descripción de como implementar otra cosa, tal como una feature que extiende de dos.

4.3 Scenarios 3: ...

Otro caso diferente ...

4.4 Conclusions

Tabla o bullets comparando lo que querríamos tener y lo que tenemos con las clases parciales.

5. SKETCH OF SOLUTIONS FOR C# PARTIAL CLASSES PITFALLS

Breves comentarios sobre como solucionar estos problemas. (NOTA: No estoy muy seguro acerca de si poner o no esta sección).

6. SUMMARY AND FUTURE WORK

7. REFERENCES

[1] Joseph Albahari and Ben Albahari. *C# 3.0 in a Nutshell*. O'Reilly, 4 edition, January 2010.

[2] Ivica Aracic, Vaidas Gasiunas, Mira Mezini, and Klaus Ostermann. An Overview of CaesarJ. 3880:135–173, 2006.

[3] Don S. Batory. Feature Models, Grammars, and Propositional Formulas. In J. Henk Obbink and Klaus Pohl, editors, *Proc. of the 9th Int. Conference on Software Product Lines (SPLC)*, volume 3714 of *LNCS*, pages 7–20, Rennes (France), September 2005.

[4] Lidia Fuentes, Carlos Nebrera, and Pablo Sánchez. Feature-Oriented Model-Driven Software Product Lines: The TENTE Approach. In Eric Yu, Johann Eder, and Colette Rolland, editors, *Proc. of the Forum of the 21st Int. Conference on Advanced Information Systems (CAiSE)*, volume 453 of *CEURS Workshops*, pages 67–72, Amsterdam (The Netherlands), June 2009.

[5] Iris Groher and Markus Völter. Aspect-Oriented Model-Driven Software Product Line Engineering. *Transacations on Aspect-Oriented Software Development (Special Issue on Aspects and Model-Driven Engineering)*, 6:111–152, 2009.

[6] Miguel A. Laguna, Bruno González-Baixauli, and José M. Marqués. Seamless Development of Software Product Lines. In *Proc. of the 6th Int. Conference on Generative Programming and Component Engineering (GPCE)*, pages 85–94, Salzburg (Austria), October 2007.

[7] Carlos Nebrera, Pablo Sánchez, Lidia Fuentes, Christa Schwanninger, Ludger Fiege, and Michael Jäger. Description of Model Transformations from Architecture to Design. Deliverable D2.3, AMPLE Project, September 2008.

[8] Christian Prehofer. Feature-Oriented Programming: A Fresh Look at Objects.

[9] Pablo Sánchez, Nadia Gámez, Lidia Fuentes, Neil Loughran, and Alessandro Garcia. A Metamodel for Designing Software Architectures of Aspect-Oriented Software Product Lines. Deliverable D2.2, AMPLE Project, September 2007.

[10] Pamela Zave. FAQ Sheet on Feature Interaction. AT&T, `http://www2.research.att.com/~pamela/faq.html`, 1999.