Aspect oriented configuration of software product line features

Jakub Perdek

Inštitút informatiky, informačných systémov a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave

20. Október, 2021

Abstrakt

Generovanie variantov produktov pomáha vytvoriť produkty z danej domény efektívnejšie a lacnejšie. Nevzniká tak iba jeden produkt, ale ľahko adaptovateľné produkty pre rôzne prostredia, jazyky a ďalšie požiadavky (Beuche a Dalgarno 2006). Derivovať produkty prispôsobené požadovanej variabilite možno za pomoci aspektovo orientovaného prístupu, hlavne jazyka AspectJ [2] (Young a Math, 1999), ktorého možnosti by sme chceli pri tejto úlohe preskúmať. Vyberáme použitie jazyka AspectJ, pretože umožňuje oddeliť záležitosti ako je logovanie, kešovanie, poolovanie, ale hlavne je možné vyradiť z vykonávania vybrané aspekty, respektíve isté vlastnosti aplikácie, prípadne aj celkovú závislosť na tomto jazyku [1] (Laddad, 2003).

Zaujíma nás preto uplatnenie jazyka pri umožnení konfigurovateľnosti konkrétnych vlastností v kóde, a samotné generovanie konkrétneho variantu produktu s prípadnou požiadavkou nezávislosti implementácie od tohto použitého jazyka. Zhodnotíme prínosy a kvality aspektovo orientovaného prístupu na konkrétnej implementácii, a porovnáme ich s identifikovanými problémami a benefitmi spomenutými v rôznych reimplementáciách už existujúcich softvérových systémov ako je vedecká kalkulačka (Botterweck, 2009) alebo databázový systém (Kastner, 2007). Samotné analýzy naznačujú význam voľby domény pri zostrojovaní radu softvérových produktov, preto by sme v neposlednom rade chceli porovna vplyv domény pri ich implementácii.

Plánujeme prispôsobiť existujúcu hru Battleship tak, aby umožňovala manažovať vlastnosti na základe už vytvoreného modelu vlastností, väčšina ktorých nie je vôbec implementovaná. Cieľom je aj analyzovať spôsoby odvodenia konkrétneho produktu, a mieru jeho závislostí od jazyka AspectJ.

Ďalšou aplikáciou pre analýzu je odvodzovanie rôznych fraktálov (ukážky typov nájdete v Pelánek, 2012) z pôvodného algoritmu, ktoré

by sme realizovali zásahom aspektov do jeho vykonávania. Predpokladáme, že úloha samotného návrhu vlastností pre uvedenú aplikáciu je určená hodnotou vzhľadu fraktálu, pričom zhotovenie radu produktov preň môže vyžadovať iné nároky na model vlastností, ako napríklad generovanie všetkých možných derivácií pre následné overenie estetiky a identifikovanie najvhodnejších kandidátov. Zamýšľame preto zhodnotiť význam prípadného potenciálu aspektovo orientovaného riešenia pre derivovanie konkrétnych fraktálov.

1 Introduction

Creation of software products needs to create project and develop given solution according requirements. This is often expensive and needs analysis for every such project. One possible way is to implement software product line to make possible derive such resulting products or components and evolve then on basis of domain knowledge. Some features can be too expensive for customer to pay for them and because of it they are ommited. Others collide with necessary ones and needs to be ommited too.

Appropriate domain analysis is required to design mandatory and voluntary features of the given system to know which products should be possible to generate and if it brings value to customer and his changing requirements. For this purpose we used feature models. In reality many of these models can exist, but only few them can support these changing needs. We think that few relations in feature model can be expressed and solved by aspect and classes almost similarly for each group of them. But sometimes features can collide and some features can't be used together. Because of mentioned problem, its necessary to think about derivation of given product and developing certain feature without colliding ones.

In our work we try to focus on applying aspect oriented language to develop program on feature model basis to serve as software product line. During our work we analyze applying ways how to do it in AspectJ language and its consequences on code quality, encapsulation, modularity benefits and dependencies which arise with using AspectJ language. We used hints and advices from papers which provided refactoring on many systems using aspect orientated languages, especially previously mentioned AspectJ.

Provided software product line and our development techniques can be later applied on fractal construction. New shapes and patterns applied at given positions during recursion drawing of given fractal are voluntary features of given model. To perceive potential aesthetic feeling from some of them is necessary to derive such products. Having appropriate product line enabling generate full range of products for their next validation is the main requirement for this specific task.

2 Overview of sections

Section 3 describes software product line and domain analysis applied to its creation. Actual state and design of Battleship game is described in section 4. Applied techniques for solution refactoring are presented in section 5. In section 6 we present a game derivation based on actual configuration settings and introducing annotations and expression which helps to choose content to copy in result project. Section 7 describes how is possible to create all possible derivations based on feature model of given solution. We are using Battleship game again with appropriate feature model for this purpose.

We mentioned possible applications for deriving fractals in section 8. Making the best design of software product line which enables derivation in all possible cases to enrich our perception and produce patterns for next validation is main goal here.

In section 9 we conclude impacts on modularity. We also analyze these impacts using aspect oriented programming and a given case of software product line approach.

Work which is done in given area is described in section 10. We describe some existing approaches solving similar problems that we have.

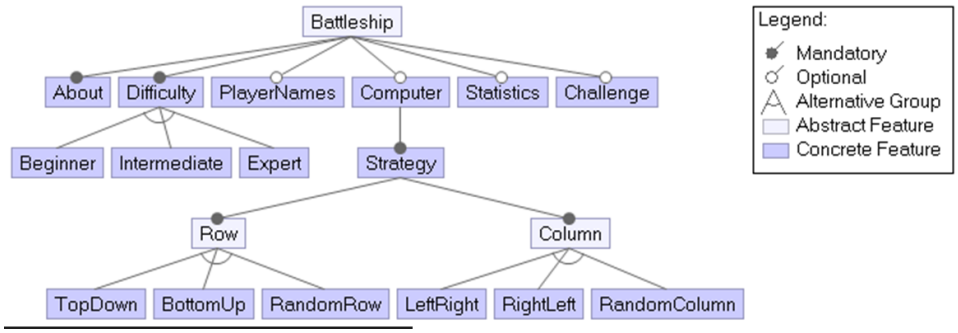
Our conclusions and next work we mentioned in section 11.

3 Software product line and domain analysis

4 Actual state and design of solution for Battleship game

To provide the most appropriate feature model which can capture potential needs of customer and domain knowledge to serve as basis for our later analysis we choose already provided one. Mentioned feature model is a part of the Battleship game in one repository for which depicts its features. Battleship game is simple game consisting of two players trying to destroy ships to each other. Lot of improvements are possible to include as additional features to make the game more interesting. We think that given feature model is based of performed domain analysis and identified many features with decision of their optionality.

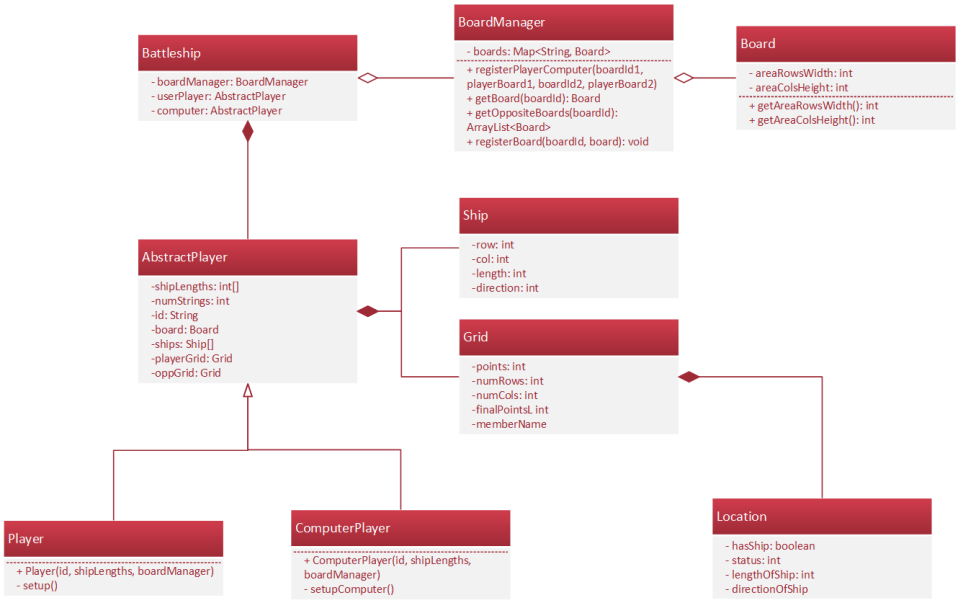
Firstly, we need to mention some problems of provided implementation of Java Battleship game. Only some of mandatory features were provided in unconfigurable way. From feature model we can name features like player names, collecting statistics, computer strategy subtree and challenge as unimplemented. Many features are hardcoded and can not be easily enhanced without code refactoring. Other aspect of such implementation is that user cant choose one option from given options without modifying code. For example choosing difficulty option in the game. The same code is repeated



Obr. 1: Feature diagram for Java Battleship game

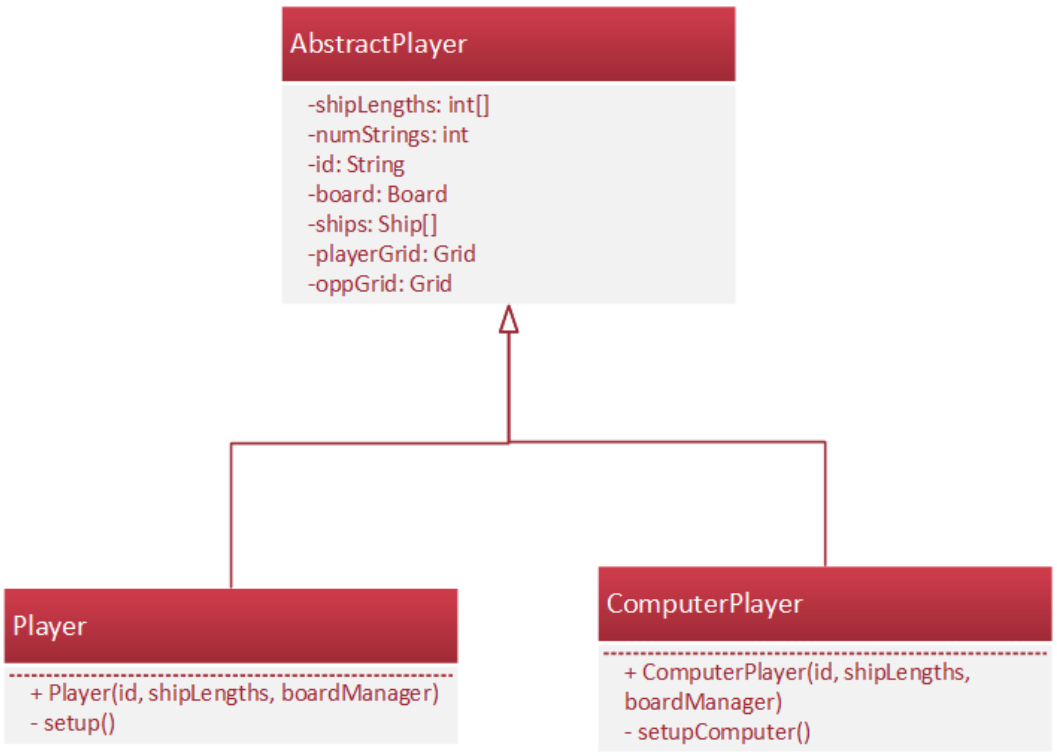
in many classes and methods. It lacks of some object oriented programming practises. No encapsulation resulting in free access to public variables from any class. Another identified problem was that concerns were not fully separated. For example setup of player was not included in player class. Some static method should be converted to dynamic ones to be associated to given instance. This game use console only and not have any graphic user interface. We let console interface as is. Implementation not provide any possibility of product derivation at all. Potential refactoring was required then.

Mentioned Battleship game is not large and not have many features, but can serve as potential board game domain application. We can evaluate its benefits on using aspects and creation of software product line with final product derivation. Separation of concerns is the main benefit observed from development enabling divide components as separated classes and aspects during product derivation into separated files for each feature and making development easier. Because of aspects, new features often do not need to change existing code and require only to specify new functionality with pointcuts and managing collisions. This functionality can be easily managed using configuration file with static variables which are used in if condition of given advice. If condition is true, then functionality will be applied otherwise not. Not only extending existing or adding new features are possible using aspects. But configuring existing classes changing their parameters is possible too. These aspects can be also replaced by other aspects which use different strategy. Solution should be more configurable, modular and easily can respond to changes. Not all changes are possible, but with modularity enabling easy product derivation these can be developed separately.



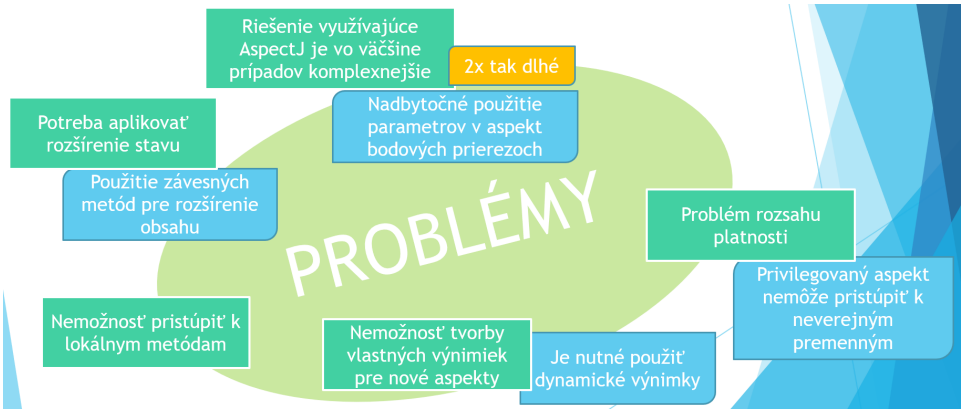
Obr. 2: Class Diagram of modified solution

5 Refactoring of Battleship game with using AspectJ language



Obr. 3: Player abstraction

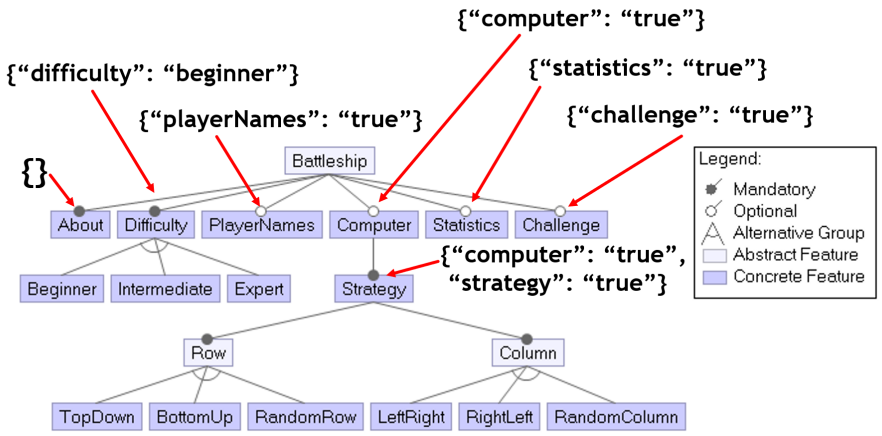
Po prvej iterácii je diagram balíkov zobrazený na obrázku 4.



Obr. 4: Problémy pouitia jazyka AspectJ

6 Derivation of given Battleship game

If classes and aspects are available there should be question how to separate given features from others except only configure aspects. We come with idea, that we can copy files or part of them which represents required features. It will be based on feature models as previous part. Mandatory features from the top of the three always should be copied. Other features deeper in feature tree often require from their parents to be included. This creates new problem how to copy such files. We needed rules which can be mappped to features. We again come with idea of creating annotations with expression enabling specify condition when such feature should be copied and when not. Mapping of some features with given values to feature tree can be seen on picture 5. For example if variable playerNames is true, then it all necessary classes for this feature will be copied in final solution. If rule will be empty then will be evaluated as true. If any file will be without annotation or all rules of its annotations will be false, then file will not be copied into final solution.



Obr. 5: Derivation rules

We specified three types of annotations:

1. The first annotation is //@{}. This annotation can be used on classes, interfaces or aspects to copy whole file. Additional check can be added to program, that one of these keywords should be included after annotation.

2. The second annotation is //#{}. This annotation is suitable for include or exclude given methods. If file should be copied then at least one condition of annotation should be true. It should not be mixed with first one.

3. The third annotation is //%{}. Using this annotation some imports can be included or excluded according including or excluding given method from file. It can be used with second type of annotation.

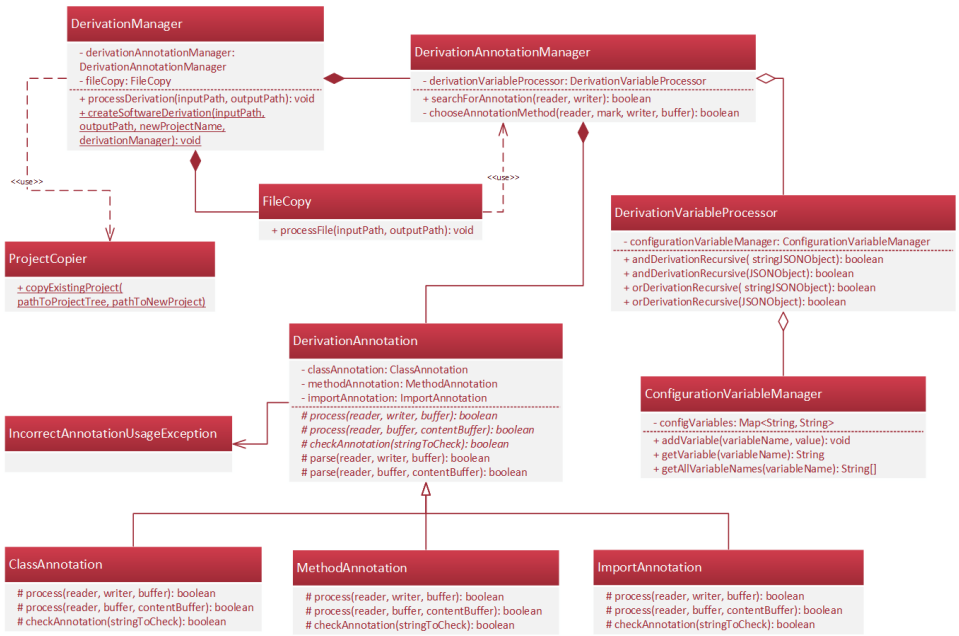
We mainly use first type of annotation, but in some cases remaining two should be used. Because that solution is modular it is possible to copy all files for each feature and make derived product functional. We can say so in situation where splitting some files with many features occurring at one place can be solution, but destroing such entities and their responsibilities at all. Because of it remaining two annotations are used. Content can be managed effectively then. Because we include or exclude methods and they can import classes which in some cases will not be part of final solution its necessary to annotate them using third annotation to be included or excluded. Typical example is functionality for setting names. Two methods are annotated with

Obr. 6: Zloený derivaný výraz

second annotation. Both of them expects that variable setNames will be true if they should be included. Only second method for setting computer name has another condition in annotation that computerOpponent variable should be set to true, otherwise method will not be copied.

For making more complex condition we enable to use AND and OR operators to specify relation between variables as is described on picture 6. We use recursion to evaluate them. If no operators are used expression is evaluated as AND statement. For AND operator all children should be true. For OR operator at least one should be true.

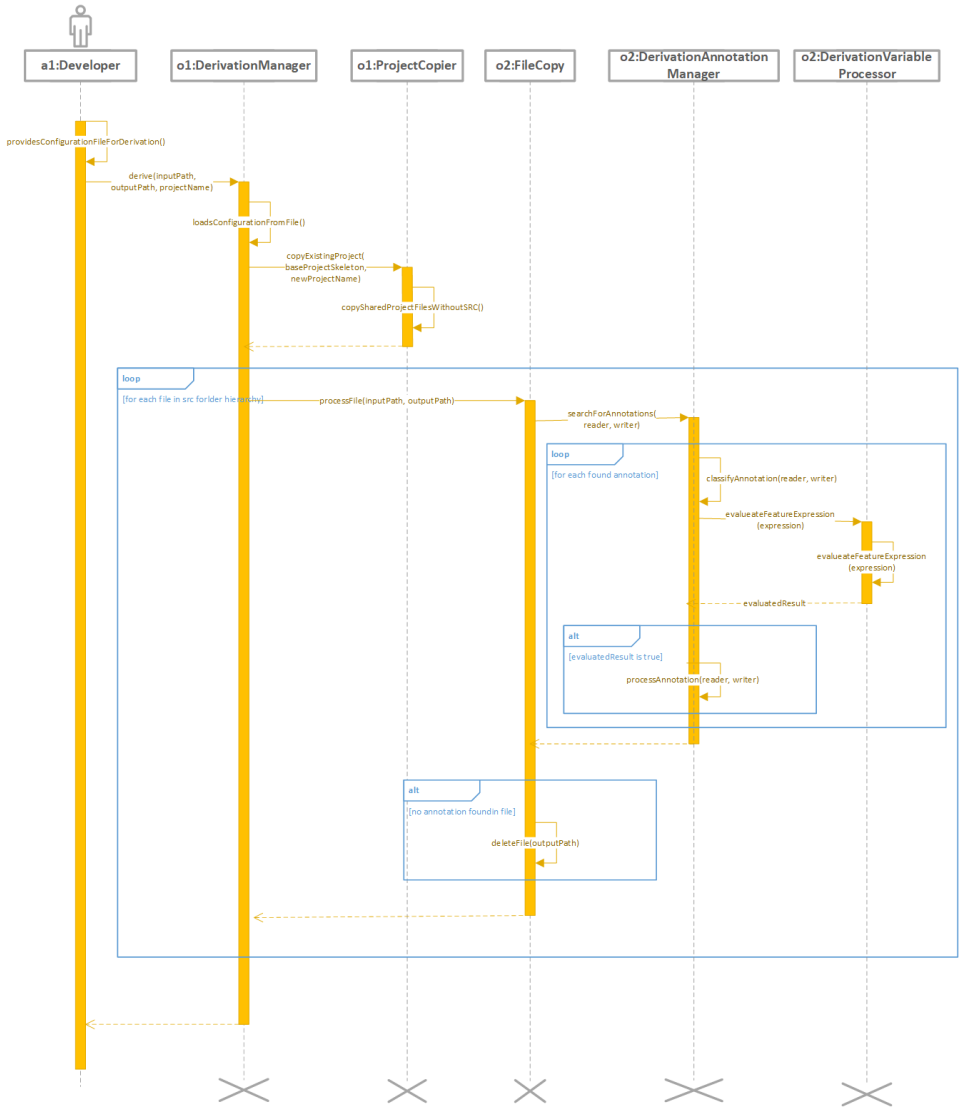
Class diagram for given solution is depicted on picture 7. DerivationManager class manages derivation and contain all necessary tools for this operations. For example project copier to copy empty project without classes or FileCopy class to manage reading and writing of given file as stream. During copying is necessary to find if file should be coped or not. For this purpose program needs to find annotation and evaluate its expressions. Searching is done by DerivationAnnotationManager which is using abstract class DerivationAnnotation with all of possible derivation annotation classes which extends this mentioned abstract class and deals with its own type



Obr. 7: Class diagram for product derivation

of processing annotation. Available are methods working with StringBuilder to save string for later or Writer to write values immediately. Each annotation should check if some keywords are included. For example aspect, interface or class for first type of annotation. DerivationVariableProcessor class helps DerivationAnnotationManager to evaluate expression associated with annotation. Provided implementation should evaluate expressions with AND and OR operators too. It needs to know used variables and their values to decide mentioned expression correctly. Mapping is provided by ConfigurationVariableManager.

We also model dynamic behavior for this generator as is depicted on picture 8. At the beginning developer chooses features setting values to configuration file. After method for derivation is called and configuration file is applied then DerivationManager class firstly copy empty project to specified result place. After it FileCopy class is used. For each file in source src directory tries to find annotation using DerivationAnnotationManager. If annotation is found then DerivationVariableProcessor class evaluates condition. If condition is true, annotation is processed using given implementation for given type of annotation. If file not have annotations or all expressions results in false result then the file will be deleted.

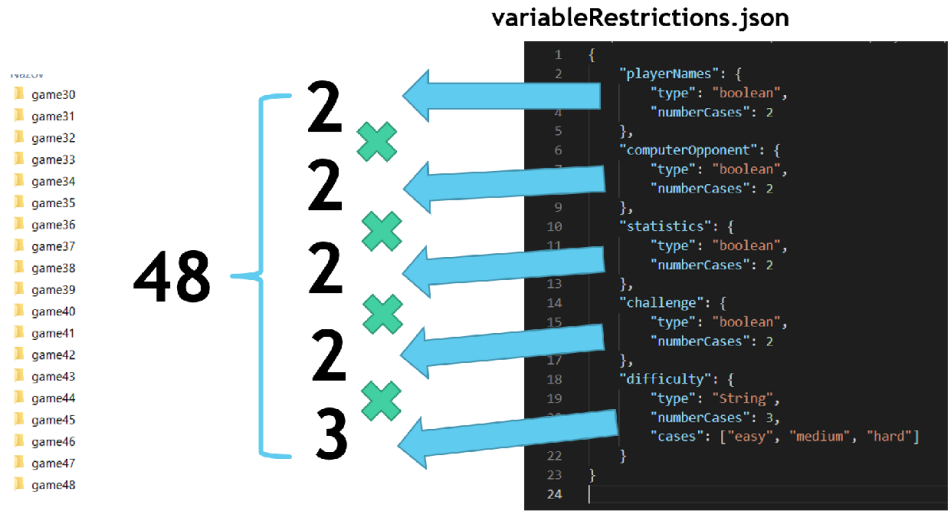


Obr. 8: Sekvenný diagram pre deriváciu produktu

7 Derivation of all types of Battleship games

Feature models helps to determine variability in solution. According them it is possible to apply given derivation. We used variables from configuration too, but also other classes that generate all possible derivations. Changing configuration variables was necessary. We need to create another configuration file where variable names are specified with other information about them. For example variable type and number of possible cases. Variable can have boolean type and possible cases are two, true and false. if variable have

string type, cases should be named explicitly in array. For example if we can include difficulty setting and include it in derivation products then possible cases will be three. Namely easy, medium and hard. We named configuration file as variableRestrictions.json. Generator will generate all cofiguration for all cases of given variables. For each configuration described derivation method will be called. In our project we created 2 * 2 * 2 * 2 * 3 = 48 derivations as depicts picture 9.



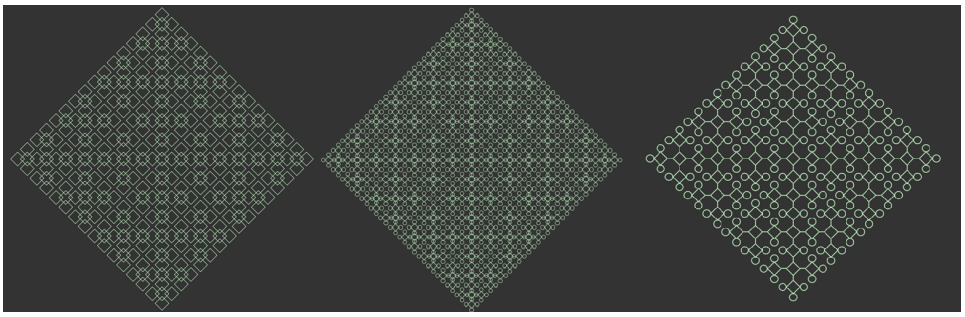
Obr. 9: Derivation of all Battleship game types

8 Software product line for fractals

Generating fractals should be interesting application in software product line design. Not only mathematical approach, but also user requirement should be applied during development. Feature models will then vary according these requirements. To evaluate resulting shapes and use their variable basis is necessary to derive such software enabling to draw fractals. All possible derivations should be created for their aesthetic evaluation to see which of them provides value to our perception. After this phase its possible to continue with development using certain derivations and enhance their structure.

During fractals analysis we found many types of fractals. Some of them are created dividing certain shapes to some extent like Sierpinskeho fractal, others extends to space using given iteration parameter. The length of line not changes during iterations. Examples are w-curves or Krishna anklet. Some for other types are created using rotation of lines in space. For example Koch snowflake or hexagram. Mentioned algorithm type used to generate fractal can be enhanced by adding more shapes on user request. Not only

parameters of algorithm will apply there. Lot of parameters are possible there. For example radius of circles, or different length of given shapes, width of the line, ommiting certain recursion calls or drawing unfilled lines as exchange for given line. With unfilled lines its possible to manage diameter between these lines. We applied given approach drawing w-curves and results shows interesting shapes apart from w-curve shape using high value for this diameter. Software product lines can help in such situations to generate all types easily and only evaluate given result when launching application is only remaining requirement.



Obr. 10: Different anklet fractal shapes

9 Resulting modularity and derived products

Aspect helps to produce modular solutions and enhance functionality without modifying of existing code. In case of separating concerns using them can prevent code scattering. Large parts of interacting code for logging, exception handling, third party functionality which is used because of code improvements can be moved to separated classes and let main logic for each concern remain in given place without such code directly included. Mentioned parts of such concerns are included during weaving. In case of product line approach we found that is possible to introduce mechanism to easy copy certain parts of given project and let project to be functional and possibly prepared for next development phase.

9.1 Modification of functionality and use of AspectJ language

Developed game using AspectJ shows these mentioned benefits as crucial for purpose of this application. Other benefits using AspectJ enabling easier configuration provided by changing method calls with values provided by chosen approach. These approach can be changed only by adding or removing given aspects from execution. Developing such aspect requires only

changing core of few methods of one type of given aspect copy to reach this goal. Aspects well suits for required features too. They well cover variable features. Adding such feature needs to prepare given aspects and join points for them. Application should be developed in respect to changing requirements. Provided code then should be parameterized because of enabling use of many features and possibility to configure application.

Using aspect not brings only benefits to development and refactoring of existing solution. AspectJ requires to include certain refactoring and code modifications to hangs certain aspects on these modified parts. Aspects pointcuts are twice as long and lot of provided parameters are repeated many times. If these pointucts are modified only in few places their copy is unreadable and is hard to distinguish amongst them. Other crucial problem is need to create empty methods called hooks to hang up certain aspect on them, because aspect oriented methods for changing behaviour of code inside method are missing. In many cases aspects are applied only on one method, not groups of methods. In larger applications it can happen that private variables are exposed to priviledged aspects and all encapsulation benefits are lost. AspectJ is not only language witch supports aspect oriented programming, but in my opinion its most powerful way for software product line development including product derivation.

9.2 Generating derivation of Battleship game

During our work we created annotations which enable selection of given classes, interfaces and aspects in final derivation. We prepared and connected these classes with configuration file to make such selection to be based on actual configuration. Changing one variable results in applied derivation and creation of different result. We let to generate all possible derivations for all implemented variable features from feature model to prove its functionality. We need only three type of basic annotations to mark classes to separate them. We also developed algorithm to evaluate expression which enables copy given files to result project. Expression consists of variables which identifies features of given feature model which serves as basis for software product line application.

10 Recent work

Kód kt

11 Conclusions and next work

Analyzo

Literatúra

[1] R. Laddad. *AspectJ in action: practical aspect-oriented programming.* Manning, Greenwich, CT, 2003. OCLC: ocm53049913.

[2] T. J. Young and B. Math. Using AspectJ to Build a Software Product Line for Mobile Devices. page 73, 1999.