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 Prehľad obsahu sekcií

Sekcia 3 poskytuje charakteristiku jadra herného engínu a popisuje rozdiely medzi ním a samostatnou hrou. Sekcia 5 sa zameriava na analýzu architektúry herného engínu na základe kódu, ktorý obsahuje. Triedy a vzťahy medzi nimi znázorňujeme pomocou nami vytvoreného diagramu tried. Vytvorený diagram balíkov by mal znázorňovať vzťahy medzi balíkmi. V sekcii 6 pri analýze štruktúry rozdeľujeme monolyt na jednotlivé vrstvy. V tejto časti sa snažíme o realizáciu komunikácie len v rámci susedných vrstiev. Dosiahnuť komunikáciu len v jednom smere je vítané. V sekcii 7 sa snažíme oddeliť herný engín od hry už v samotnom návrhu tým, že z engínu vytvárame predpripravený subsystém, ktorý môže byť v prípade potreby nakonfigurovaný podľa potrieb hry. Tvorba hier by preto mala byť jednoduchšia a prepojenia s engínom by mali byť v menšom počte. Sekcia 8 reprezentuje návrh distributívneho riešenia pre lepšiu rozšíriteľnosť a tvorbu modulov. SOA by mala by univerzálnou voľbou kvôli princípu voľnej väzby pri skladaní služieb, ale distributívna architektúra založená na vzdialenom volaní procedúry môže byť rýchlejšia. V tejto časti sa zameriavame hlavne na zmenšenie závislostí medzi 3D matematikou. Sekcia **??** je venovaná realizácii engínu v jazyku Javascript. Snažíme sa poskytnúť informácie o použití podobnej technológie založenej na funkcionalite plátna. Zaoberáme sa komplikáciami a implikáciami preneseného riešenia v súvislosti s architektúrou. Sekcia 9 zhodnocuje vykonanú prácu spolu s opisom hlavných prínosov jednotlivých častí riešenia. Sekcia 10 sumarizuje prehľad ďalších publikácií a vývoja podobných herných engínov. Cieľom je poskytnúť aj informácie o ďalších komponentoch a technológiách pri tvorbe jadra herného engínu softvérového riešenia. Sekcia 11 obsahuje zhodnotenie práce a ďalších možností rozšírenia tejto práce.

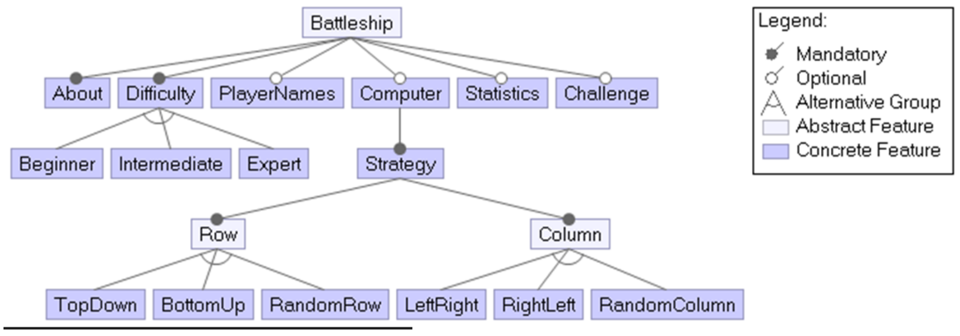
3 Software product line and domain analysis

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.

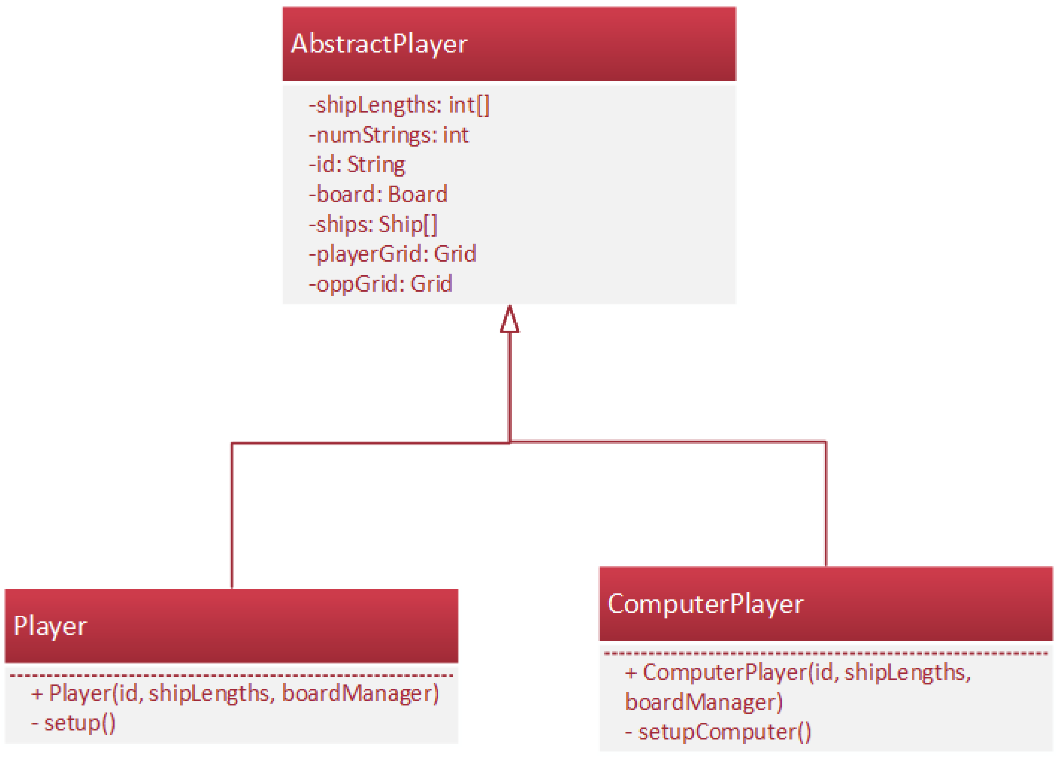
Firstly, we need to mention some problems of provided implementation of Java Battleship game. Only some of mandatory features were provided in unconfigurable way. From feature model we can name features like player names, collecting statistics, computer strategy subtree and challenge as unimplemented. 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 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.



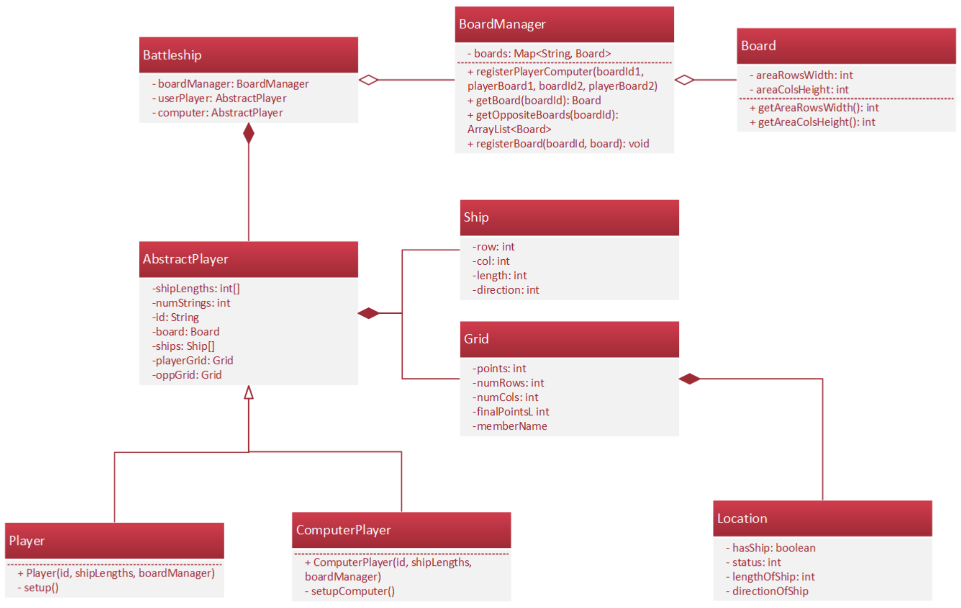
Obr. 1: Feature diagram for Java Battleship game

5 Refactoring of Battleship game with using AspectJ language

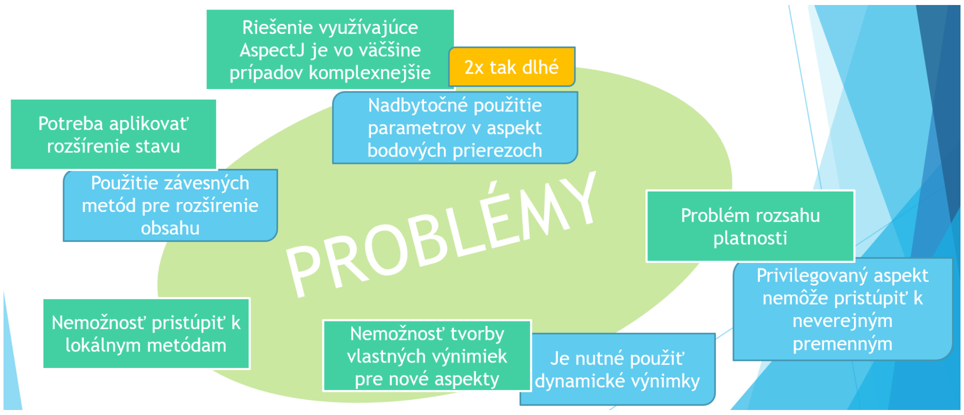


Obr. 3: Player abstraction

Po prvej iterácii je diagram balíkov zobrazený na obrázku 4.


BoardManager
- boards: Map<String, Board>
+ registerPlayerComputer(boardId1, playerBoard1, boardId2, playerBoard2)
+ getBoard(boardId): Board
+ getOppositeBoards(boardId): ArrayList<Board>
+ registerBoard(boardId, board): void
Battleship
- boardManager: BoardManager
- userPlayer: AbstractPlayer
- computer: AbstractPlayer
Board
- areaRowsWidth: int
- areaColsHeight: int
+ getAreaRowsWidth(): int
+ getAreaColsHeight(): int
Ship
-row: int
-col: int
-length: int
-direction: int
AbstractPlayer
-shipLengths: int[]
-numStrings: int
-id: String
-board: Board
-ships: Ship[]
-playerGrid: Grid
-oppGrid: Grid
Grid
-points: int
-numRows: int
-numCols: int
-finalPointsL int
-memberName
Player
+ Player(id, shipLengths, boardManager)
- setup()
ComputerPlayer
+ ComputerPlayer(id, shipLengths, boardManager)
- setupComputer()
Location
- hasShip: boolean
- status: int
- lengthOfShip: int
- directionOfShip

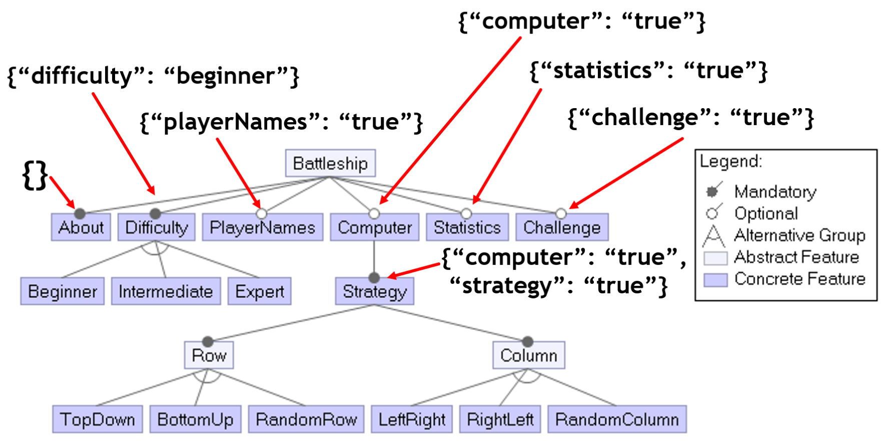
Obr. 2: Class Diagram of modified solution


Riešenie využívajúce AspectJ je vo väčšine prípadov komplexnejšie
2x tak dlhé
Potreba aplikovať rozšírenie stavu
Nadbytočné použitie parametrov v aspekt bodových prierezoch
Použitie závesných metód pre rozšírenie obsahu
PROBLÉMY
Problém rozsahu platnosti
Privilegovaný aspekt nemôže pristúpiť k neverejným premenným
Nemožnosť pristúpiť k lokálnym metódam
Nemožnosť tvorby vlastných výnimiek pre nové aspekty
Je nutné použiť dynamické výnimky

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.

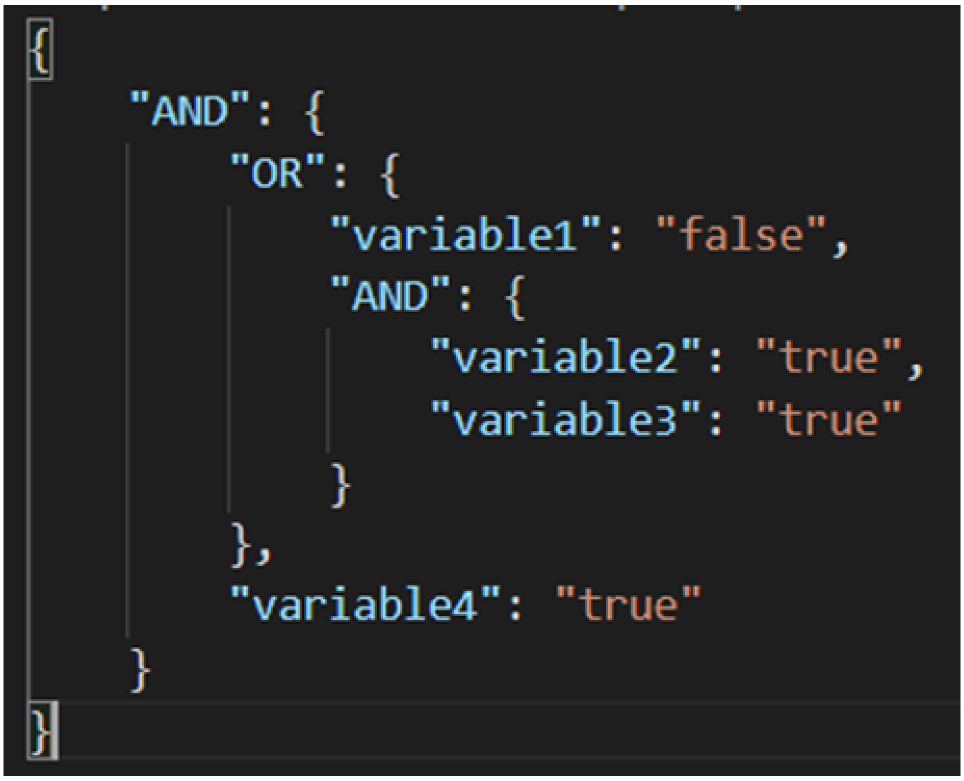
3. The third annotation is //%{}. Using this annotation some imports can be included or excluded according including or excluding given method from file. It can be used with second type of annotation.

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 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. Derivation-Manager 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 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 condi-



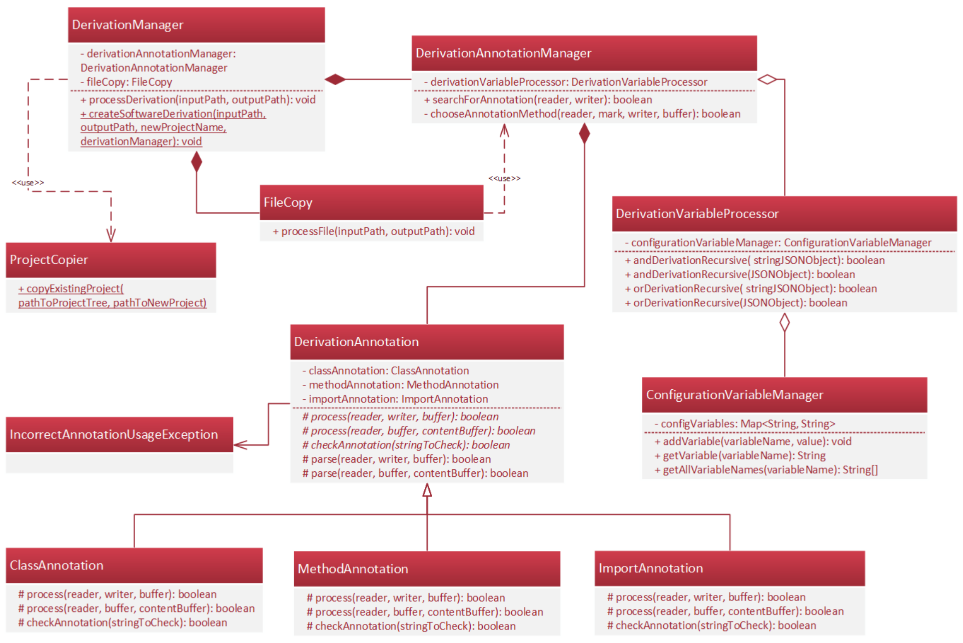
```
{
    "AND": {
        "OR": {
            "variable1": "false",
            "AND": {
                "variable2": "true",
                "variable3": "true"
            }
        },
        "variable4": "true"
    }
}
```

Obr. 6: Zloený derivaný výraz

tion. 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.

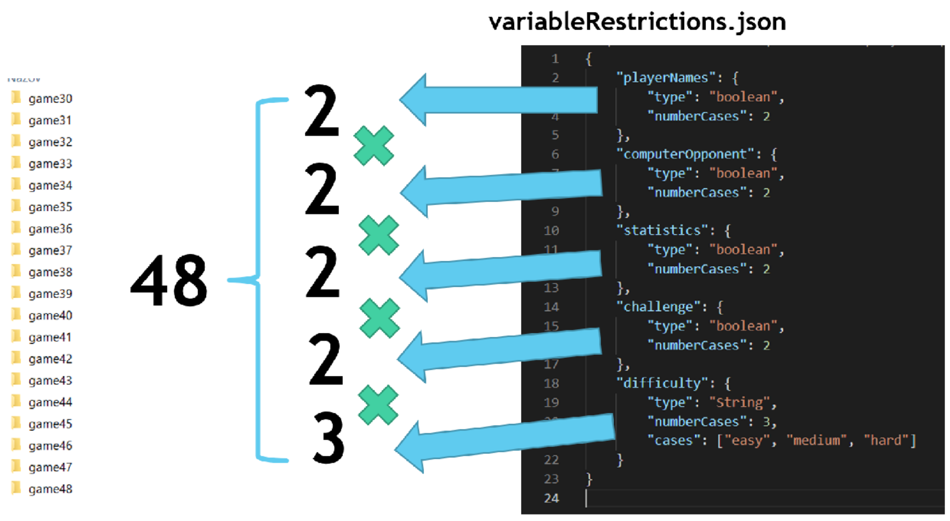
7 Derivation of all types of Battleship games

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


DerivationManager
- derivationAnnotationManager: DerivationAnnotationManager
- fileCopy: FileCopy
+ processDerivation(inputPath, outputPath): void
+ createSoftwareDerivation(inputPath, outputPath, newProjectName, derivationManager): void
DerivationAnnotationManager
- derivationVariableProcessor: DerivationVariableProcessor
+ searchForAnnotation(reader, writer): boolean
- chooseAnnotationMethod(reader, mark, writer, buffer): boolean
<<use>>
<<use>>
FileCopy
+ processFile(inputPath, outputPath): void
DerivationVariableProcessor
- configurationVariableManager: ConfigurationVariableManager
+ andDerivationRecursive(stringJSONObject): boolean
+ andDerivationRecursive(JSONObject): boolean
+ orDerivationRecursive(stringJSONObject): boolean
+ orDerivationRecursive(JSONObject): boolean
ProjectCopier
+ copyExistingProject(pathToProjectTree, pathToNewProject)
DerivationAnnotation
- classAnnotation: ClassAnnotation
- methodAnnotation: MethodAnnotation
- importAnnotation: ImportAnnotation
# process(reader, writer, buffer): boolean
# process(reader, buffer, contentBuffer): boolean
# checkAnnotation(stringToCheck): boolean
# parse(reader, writer, buffer): boolean
# parse(reader, buffer, contentBuffer): boolean
ConfigurationVariableManager
- configVariables: Map<String, String>
+ addVariable(variableName, value): void
+ getVariable(variableName): String
+ getAllVariableNames(variableName): String[]
IncorrectAnnotationUsageException
ClassAnnotation
# process(reader, writer, buffer): boolean
# process(reader, buffer, contentBuffer): boolean
# checkAnnotation(stringToCheck): boolean
MethodAnnotation
# process(reader, writer, buffer): boolean
# process(reader, buffer, contentBuffer): boolean
# checkAnnotation(stringToCheck): boolean
ImportAnnotation
# process(reader, writer, buffer): boolean
# process(reader, buffer, contentBuffer): boolean
# checkAnnotation(stringToCheck): boolean

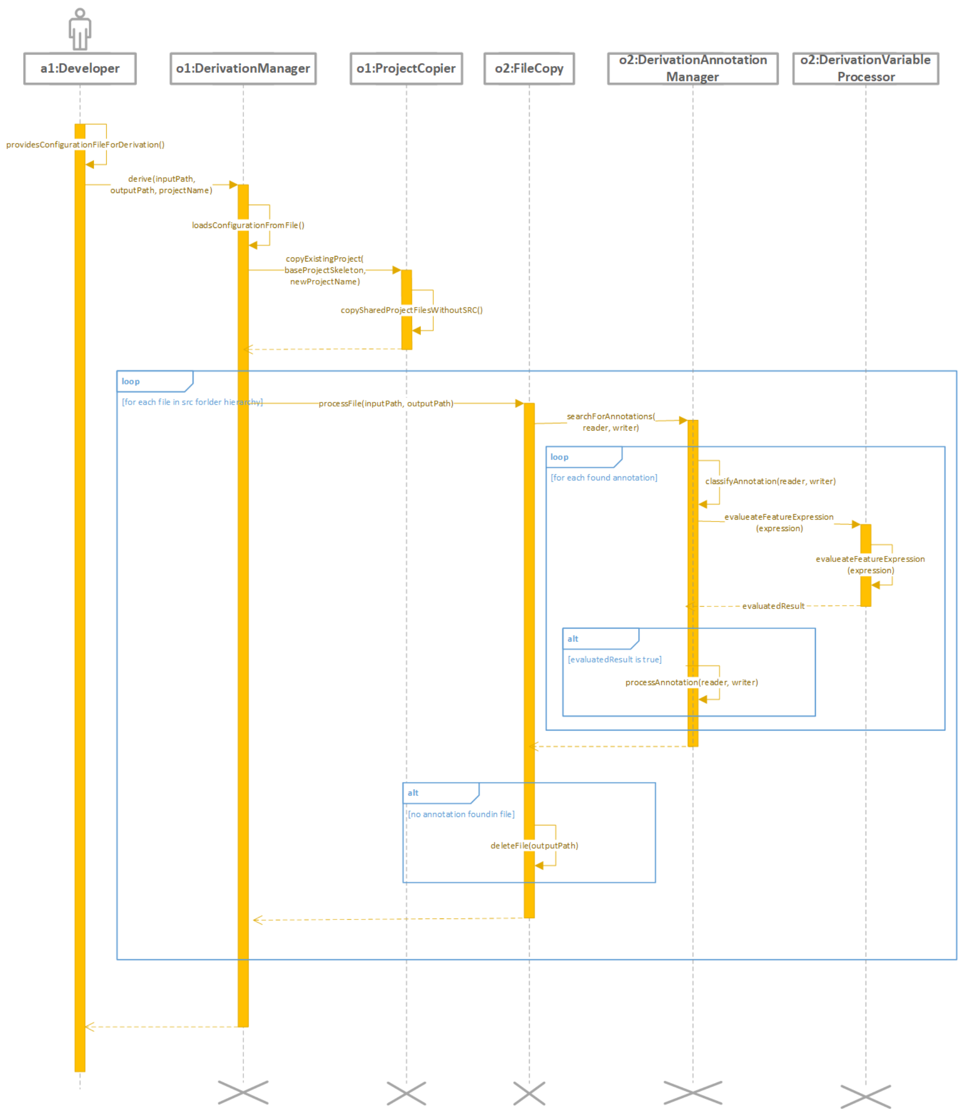
Obr. 7: Class diagram for product derivation

method will be called. In our project we created 2 * 2 * 2 * 2 * 3 = 48 derivations as depicts picture 9.



```
{
    "playerNames": {
        "type": "boolean",
        "numberCases": 2
    },
    "computerOpponent": {
        "type": "boolean",
        "numberCases": 2
    },
    "statistics": {
        "type": "boolean",
        "numberCases": 2
    },
    "challenge": {
        "type": "boolean",
        "numberCases": 2
    },
    "difficulty": {
        "type": "String",
        "numberCases": 3,
        "cases": ["easy", "medium", "hard"]
    }
}
```

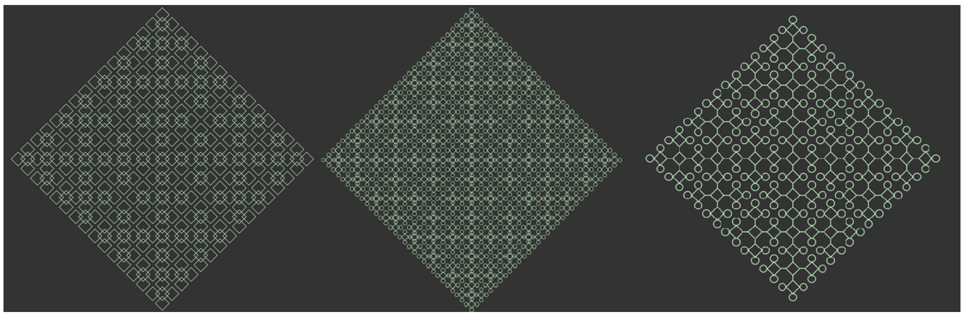
Obr. 9: Derivation of all Battleship game types


a1:Developer
o1:DerivationManager
o1:ProjectCopier
o2:FileCopy
o2:DerivationAnnotation Manager
o2:DerivationVariable Processor
providesConfigurationFileForDerivation()
derive(inputPath, outputPath, projectName)
loadsConfigurationFromFile()
copyExistingProject(baseProjectSkeleton, newProjectName)
copySharedProjectFilesWithoutSRC()
loop
[for each file in src forlder hierarchy]
processFile(inputPath, outputPath)
searchForAnnotations(reader, writer)
loop
[for each found annotation]
classifyAnnotation(reader, writer)
evalueateFeatureExpression (expression)
evalueateFeatureExpression (expression)
evaluatedResult
alt
[evaluatedResult is true]
processAnnotation(reader, writer)
alt
[no annotation foundin file]
deleteFile(outputPath)

Obr. 8: Sekvenný diagram pre deriváciu produktu

8 Software product line for fractals

SOA vyu



Obr. 10: Rôzne podoby Anklet fraktálu

Riešenie

9 Výsledná modularita a odvodené produkty

Po postupn

9.1 Úprava funkcionality a pouitie jazyka AspectJ

V návr

9.2 Generovanie derivácií Battleship hry

Vytvorili

10 Podobná práca

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.