Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Projekt 5

Generování konfigurací softwarových komponent z modelů vlastností

Plzeň 2019 Vaněk Jakub

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 9. ledna 2019

Vaněk Jakub

Abstract

Configuration of Software Components Generator from Feature Models. Goal of this thesis is to generate feature model from gramatics of tesa language written in Xtext and to generate final source code from chosen features of this model. This thesis describes and uses knowledge about feature modeling and generative programming. In the opening part of the thesis reader is introduced with feature modeling, tools used for feature modeling, generative programming and Xtext framework. In the later part implementation design and final solution is described.

Abstrakt

Cílem této práce bude vygenerovat šablonu jako model vlastností ze zápisu gramatiky jazyka tesa pomocí frameworku Xtext a na základě vybraných vlastností z tohoto modelu vygenerovat finální zdrojový kód. V práci jsou popsány a využity znalosti o modelování vlastností a generativním programování. Čtenář je v úvodní části seznámen s problematikou modelování vlastností, nástroji které jsou pro modelování, nebo tvorbu modelů využívány. Dále bude seznámen s problematikou generativního programování a frameworkem Xtext. V pozdějších částích bude vysvětlen návrh implementace a finální řešení problému.

Obsah

1 Úvod

Generování softwarových komponent bude využívat generativního programování. Generativní programování je způsob programování, kdy je zdrojový kód programu generován na základě šablony. Tuto šablonu tvoří různé vzory.

K vytvoření šablony, pro vygenerování finálního zdrojového kódu bude v práci využíván model vlastností. Model vlastností zachycuje všechny různorodosti a podobnosti všech možných variant finálního produktu. Různé charakteristiky, které finální produkty rozlišují se nazývají právě vlastnosti. Vlastnosti jsou základními stavebními kameny modelu vlastností, který zobrazuje závislosti mezi nimi.

V úvodní části se tato práce zabývá analýzou problematiky modelů vlastností, nástroji které se pro modelování vlastností využívají, generativního programování a Xtextu. Následně bude popsán návrh implementace, popis samotné implementace a na závěr zhodnocení dosažených výsledků.

Tento dokument se vztahuje k předmětu PRJ5, což je předmět související s bakalářskou prací. Tento dokument obsahuje analýzu problému, ve kterém pojednává o modelování vlastností, generativním programování a Xtextu. Ve finální bakalářské práci bude doplněn návrh implementace, popis řešení a zhodnocení dosažených výsledků.

2 Modelování vlastností

Účelem této kapitoly je seznámit čtenáře s problematikou *modelování vlastností* (z angl. *feature modeling*), které je v práci použito jako šablona pro generování zdrojového kódu konfigurátoru a s nástroji, které modelování vlastností umožňují.

2.1 Motivace

Motivaci pro modelování vlastností si ukážeme na příkladě. Představme si, že společnost vyvíjí kávovar. Kávovar bude vždy připravovat kávu, avšak jeho vlastnosti se mohou lišit na základě předem určených specifikací. Součástí specifikace může být požadavek, že kávovar bude vyvíjen pro různé trhy, například pro Evropský trh a trh v USA. Dále může být vyžadováno vytvoření dvou různých edic kávovaru, pro příklad standartní edici a deluxe edici, kde deluxe edice bude oproti standartní edici obsahovat trysku na čistou horkou vodu a displej. Za těchto předpokladů je třeba si uvědomit, že kávovar pro Americký a Evropský trh bude využívat jiný adaptér. Pro Evropský trh je potřeba klasických 220V a pro USA 120V. Zároveň je u kávovaru možno si zvolit, zda bude nebo nebude mít nastavitelné množství kávových zrn a množství vody, ze které bude káva připravena.

Kávovar se v tomto případě nazývá *produktovou řadou* (z angl. *product line*), *produktovou rodinou* (z angl. *product famiy*), nebo také *konceptem* (z angl. *concept*). Produktová rodina, či koncept, jsou pojmy, které označují skupinu produktů, které fungují na stejném principu, ale liší se od sebe různými vlastnostmi, tudíž je vytvořeno několik variant. Varianty je třeba spravovat. Je potřeba znázornit, které vlastnosti bude jaká varianta obsahovat, jak na sobě vlastnosti závisí, které a zda jsou potřeba. K tomu nám slouží právě *model vlastností* (z angl. *feature model*).

Tento model je tvořen procesem, který nazýváme modelování vlastností. Vytvořit takovýto model má hned několik výhod. Základní výhodou je přehlednost. Pokud zkonstruujeme správný model vlastností, je v něm na první pohled vidět, jaké produkty můžeme z vlastností sestavit. Model se poté dá použít pro nové verze stejného produktu tím, že se některé vlastnosti změní, nebo také jednoduše přidají.

Hlavní motivací je tedy identifikace a zachycení variability, znovupoužitelnost a rozšířitelnost systému či produktu.

2.2 Řada softwarových produktů

Řada softwarových produktů (z angl. *Software product line*) je v softwarovém inženýrství pojem, který označuje kolekci podobných softwarových systémů s podobným zaměřením. Klade důraz na podobnosti mezi softwarovými produkty. Během tohoto procesu je vytvořen koncept, kde lehké odlišnosti vytvoří sadu konkrétních produktů. Nejedná se ale pouze o recyklaci využitelných částí jednoho z hotových produktů, ale o strategické vytvoření základních stavebních kamenů tak, aby byli jednodušše rozšiřitelné a neměnné.

2.3 Model rodiny produktů

Model rodiny produktů (z angl. *Family model*) je podle [Pure13] model, který popisuje jak budou výsledné produkty skupiny produktů sestaveny nebo generovány ze specifikovaných částí. Každá komponenta modelu rodiny produktů reprezentuje jeden nebo více funkčních prvků produktu z řady produktů.

2.4 Vlastnosti

Každý koncept má určité *vlastnosti* (z angl. *features*. Pojem vlastnosti můžeme aplikovat jak na vyráběné produkty, tak na různé vlastnosti systému. Příkladem vlastností u vyráběných produktů mohou být vlastnosti znázorněné v příkladu s kávovarem. Vlastnosti jsou také užitečné při vyvíjení softwarových produktů. Požadavkem může být například kompatibilita s různými operačními systémy. Ve výsledném systému bude mít pak každá vlastnost odlišnou implementaci a na základě poskládání těchto vlastností v jeden celek získáme finální systém. Vlastnosti nám tedy zajišťují variabilitu mezi odlišnými produkty se stejným základem.

2.4.1 Model vlastností

Model vlastností znázorňuje závislosti mezi vlastnostmi. Vlastnosti v modelu tvoří strom, kde kořenovou vlastností je koncept. Koncept obsahuje další vlastnosti jako své potomky.

2.4.2 Model variant

Model variant (z angl. *Variant model*) je podle [Pure13] model, který z vybraných vlastností modelu vlastností tvoří finální produkt. Jsou v něm zachycené pouze výsledně použité vlastnosti.

2.4.3 Grafické znázornění

Modely vlastností jsou zobrazovány v *diagramu vlastností* (z angl. *feature diagram*). Diagram vlastností je zakreslován jako stromový graf, kde koncept je kořenovým uzlem a všechny další uzly jsou jeho vlastnostmi. Každý uzel může mít N potomků.

2.4.4 Typy vlastností

Jak už bylo řečeno, každý produkt u kterého je požadavek na určitou variabilitu, znovupoužitelnost či rozšiřitelnost obsahuje vlastnosti. Tyto vlastnosti mohou být několika typů. V této části se budeme věnovat základním typům těchto vlastností, tak jak jsou popsány v [Cza98].

Povinné vlastnosti

Povinné vlastnosti (z angl. *Mandatory features*) jsou vlastnosti, které výsledný produkt musí obsahovat. Jsou to vlastnosti, na kterých je koncept založen a které jsou zahrnuty v jeho popisu. Povinná vlastnost musí být ve výsledném modelu zahrnuta, pokud je zahrnutý i její rodič. Například náš kávovar bude mít vždy mlýnek na kávu, odkapávač, zásobník zbytků, zásobník vody a další. Bez těchto vlastností se neobejde žádná varianta výsledného produktu. Tyto vlastnosti mají význam hlavně v případě, že jejich rodičovská vlastnost povinná není. Pokud je tedy zahrnut rodič, je nutné zahrnout i tuto vlastnost.

Povinnou vlastnost v diagramu značíme jednoduchou hranou zakončenou vybarveným kruhem.

ZDE BUDE OBRÁZEK POVINNÝCH VLASTNOSTÍ

Každá instance konceptu C má vlastnost *f1* a *f2* a každá co má *f1* má *f3* a *f4*. Z toho vyplývá, že každá instance konceptu C má vlastnosti *f3* a *f4*. Můžeme tedy říct, že koncept C je popsán sadou vlastností:
{C,f1,f2,f3,f4}.

Volitelné vlastnosti

Volitelné vlastnosti (z angl. *Optional features*) mohou být zahrnuty v popisu konceptu. Jinými slovy, pokud je zahrnut rodič, volitelná vlastnost může a nemusí být zahrnuta. Pokud rodič volitelné vlastnosti zahrnutý není, nemůže být zahrnuta ani volitelná vlastnost na něm závislá. V příkladě s kávovarem může být touto vlastností například zmíněné nastavitelné množství kávových zrn a množství vody. Tuto vlastnost mít kávovar může, ale zároveň nemusí a tvoří tedy další varianty produktu. Volitelné vlastnosti v diagramu značíme jednoduchou hranou zakončenou prázdným kruhem.

ZDE BUDE OBRÁZEK VOLITELNÝCH VLASTNOSTÍ

Každá instance konceptu může mít vlastnost *f1*, vlastnost *f2*, obě, nebo žádnou. Pokud má vlastnost *f1*, může mít také vlastnosti *f3* a *f4*. Koncept můžeme popsat sadou vlastností:

- *{C}*
- *{C,f1}*
- *{C,f2}*
- *{C,f1,f2}*
- *{C,f1,f3}*
- *{C,f1,f2,f3}*

Alternativní vlastnosti

Alternativní vlastnosti (z angl. *Optional features*) jsou vlastnosti, kde existuje možnost výběru mezi více vlastnostmi. V kávovaru je takovou vlastností edice, kdy je potřeba si vybrat mezi standart nebo deluxe edicí, ale výsledný produkt nemůže obsahovat obě. Alternativní vlastnosti mohou být volitelné, nebo povinné. Pokud je soubor alternativních vlastností povinný, je třeba vybrat právě jednu z nich. Pokud jsou alternativní vlastnosti volitelné, je třeba vybrat nejvýše jednu z nich. Alternativní vlastnosti v diagramu značíme prázdným obloukem mezi hranami.

ZDE BUDE OBRÁZEK ALTERNATIVNÍCH VLASTNOSTÍ

V instanci jsou znázorněný povinné alternativní vlastnosti. Musíme si tedy v

levé větvi vybrat mezi vlastnostmi *f1* a *f2* a na pravé větvi mezi vlastnostmi *f3*, *f4* a *f5*. Koncept můžeme popsat sadou vlastností:

- *{C,f1,f3}*
- *{C,f1,f4}*
- *{C,f1,f5}*
- *{C,f2,f3}*
- *{C,f2,f4}*
- *{C,f2,f5}*

Slučitelné vlastnosti

Slučitelné vlastnosti (z angl. *Or-features*) jsou vlastnosti, kde stejně jako u alternativních vlasntostí existuje možnost výběru. Oproti alternativním vlastnostnem ale znázorňují situaci, kdy je možnost výběru více než jedné vlastnosti. Mohou být opět volitené nebo povinné. Pokud je slučitelná možnost volitelná, není třeba vybrat žádnout z nich. Pokud je povinná, je třeba vybrat alespoň jednu z nich. Slučitelné vlastnosti v diagramu značíme plným obloukem mezi hranami.

ZDE BUDE OBRÁZEK SLUČITELNÝCH VLASTNOSTÍ

Dalši typy vlastností

Model vlastností může nabývat velké složitosti. Proto je nutné zavést další typy vlastností a rozšíření původního modelu vlastností. Jedním z těchto rozšíření je zavedení pojmu *kardinalita*. Kardinality označují počet kopií jednotlivé vlasnosti. Počet těchto vlastností v diagramu označujeme jako interval ve tvaru $[m...n]$, kde m je dolní a n je horní hranice počtu výskytů těchto vlastností. Tato notace se poprvé vyskytla z důvodu inspirace autorů z UML diagramů v publikaci [Rieb02].

Autoři [Rieb02] také nahrazují alternativní a slučitelné vlastnosti pojmem *skupina vlastností* (z angl. *feature group*). Kardinalita zde označuje kolik vlastností z dané skupiny je možné použít. Alternativní vlastnosti zde nazýváme *XOR skupinou* (z angl. *XOR group*) a slučitelné vlastnosti *OR skupinou* (z angl. *OR group*).

Diagram vlastností dokáže efektivně zachytit veškeré vlastnosti a závislosti mezi nimi. Problém však může nastat v případě, kdy nějaká vlastnost vyžaduje jinou vlastnost, která není jejím přímým rodičem. Pro takové případy je zde zaveden pojem *omezení* (z angl. *constraints*). Tato omezení si můžeme představit u požadavku, že kávovar bude mít displej pouze v případě, že velikost nádrže na vodu bude více než půl litru. Tyto dvě vlastnosti spolu v diagramu nijak nesouvisí a využívají tedy omezení. Omezení mohou být dvojího typu. Podle autorů [Pas05] to jsou *lokální omezení* (z angl. *local constraints*) a *globální omezení* (z angl. *global constraints*). Pojem lokální omezení je používán, pokud se toto omezení vztahuje pouze ke společnému rodiči těchto vlastností. Pojem globální omezení je používán pokud se tato omezení vztahují na vlastnosti napříč diagramem.

Shrnutí

Na základě těchto typů vlastností dokážeme z libovolného konceptu sestavit hotový model vlastností. Díky němu jsme schopni zachytit veškeré varianty finálního produktu. Jsme schopní zjistit co jaká varianta vyžaduje a co musí obsahovat.

Diagram by krom názvů vlastností a typu závislostí měl obsahovat i informace o vlastnostech. Tyto informace by měly obsahovat důvod, proč je tato vlastnost vyžadována, jak souvisí se zbytkem modelu a veškeré další informace, které mohou být při vývoji užitečné.

3 Modelovací nástroje

V této kapitole se budeme zabývat nástroji, které umožňují modelování vlastností a které by bylo vhodné použít pro účely práce. Cílem je ukázat, jaké modelovací nástroje existují, k čemu se používají a zachytit rozsah jejich funkcí. Bude vysvětleno, jaké mají nástroje výhody a nevýhody a odůvodnění výběru nástroje.

3.1 pure::variants

Pure::variants je jeden z mála komerčně využívaných modelovacích nástrojů od společnosti pure-systems. Není zaměřen pouze na modelování vlastností, ale svou funkcionalitou se snaží pokrýt všechny fáze vývoje software. Samotný software je plug-inem do vývojového prostředí Eclipse. Umožňuje práci s několika modely a pro každý z těchto modelů má vlastní editor.

Hlavním modelem je *Feature Model* nebo-li model vlastností. Software zobrazuje model vastností ve stromové architektuře. Umožňuje vytvoření čtyř různých závislostí: povinné, volitelné, alternativní a slučitelné. Umožňuje také do modelu zanést pravidla o omezeních, které jsou nezbytné při výsledné konfiguraci.

Dalšími modely jsou *Family model* nebo-li model rodiny produktů. Tento model zobrazuje elementy této rodiny a dokáže na ně namapovat vlastnosti. Dále *Variant description model*, který vlastnosti dokáže konfigurovat. Posledním modelem je *Variant result model*, který narozdíl od předchozích jako jediný nemá vlastní editor. Tento model popisuje konkrétní výstupní variantu produktu, její popis a informace pro její sestavení.

V našem případě využijeme feature model jako zobrazení závislostí mezi částmi konfigurace. Na základě tohoto feature modelu budeme schopni vytvořit libovolné množství variant description modelů, ze kterých bude generován výsledný kód.

ZDE BUDE OBRÁZEK PURE::VARIANTS

3.2 Xfeature

Nástroj *Xfeature* je dalším plug-inem do vývojového prostředí Eclipse, který poskytuje grafické uživatelské rozhraní pro práci s modely vlastností. Modely vlastností zde vyjadřují model rodiny produktů a modely aplikací. K modelu vlastností přistupují jako k meta-modelu vlastností. Model vlastností i jeho konfigurace jsou zde popsány pomocí XML dokumentu, který odpovídá určitému XML schématu (meta-modelu). Uživatel je schopen vytvořit vlastní XML schéma, které však musí odpovídat jeho meta-modelu. Nové vlastnosti jdou tedy tvořit pouze v souladu s meta-modelem. Tvorba modelů vlastností je zde prováděna pomocí kontextového menu. Umožňuje tvorbu povinných, volitelných i alternativních vlastností.

ZDE BUDE OBRÁZEK XFEATURE

3.3 Feature Modeling Plug-in

Nástroj *Feature Modeling Plug-in* je také zásuvný modul do vývojového prostředí Eclipse. Umožňuje vytváření, editaci i výslednou konfiguraci vlastností s kardinalitou a atributy podle [Cza05]. Tento nástroj se již dále nevyvíjí.

Jako ve většině nástrojů je model vlastností zobrazován ve stromové struktuře. Velkou výhodou je možnost rozdělit celý model do menších celků, na které se dá odkazovat a celý model tím zpřehlednit. Po grafické stránce se velmi podobá nástroji pure::variants. Ovládání probíhá skrz kontextové menu a je intuitivní a přehledné.

Nástroj neumožňuje vytváření samostatných modelů pro výslednou konfiguraci. Tato konfigurace probíhá na stejném stromě jako editace modelu vlastností pomocí úprav jeho částí, což může celý model znepřehlednit.

3.4 Software Product Lines Online Tools

Nástroj *Software Product Lines Online Tools* je nástroj implementovaný jako webová aplikace. Aplikace je zdarma a volně k použití. Umožňuje vytvoření a editaci modelu vlastností pomocí grafického rozhraní. Lze zde přidávat povinné, volitelné vlastnosti i OR a XOR skupiny. Editor také umožňuje vytvoření globálních omezení.

Nástroj umí během konfigurace hlídat dodržení pravidel včetně omezení a dokáže konfiguraci opravovat na základě vytvořeného modelu vlastností.

Model vlastností je ukládán do databáze a je možné ho sdílet s ostatními uživateli. Konfiguraci následně umí exportovat do souboru formátu CSV nebo XML.

3.5 Vlastní nástroj

Další možností využití nástrojů pro modelování vlastností je vytvořit nástroj vlastní. Jelikož je zadání velmi specifické, bylo by možné vytvořit vlastní nástroj, který bude umět pracovat se specifickými daty. Požadavky na takový nástroj by byly:

- graficky zobrazit model vlastností z gramatiky psané v Xtextu
- umožnit vytvoření modelu variant z vytvořeného modelu vlastností
- validace vytvořené varianty na základě typů vlastností včetně globálních omezení
- export a import modelů variant, kvůli sdílení mezi uživateli
- vygenerování šablony v jazyce tesa

Nástroj by nemusel umět editaci modelu vlastností, jelikož by měl pouze zobrazovat závislosti zavedené v gramatice.

Odhad času vývoje

Vývoj takového nástroje musí projít všemi fázemi vývoje software. Těmito fázemi jsou analýza, návrh implementace, implementace, testování nástroje a validace všech předešlých fází. Během vývoje by docházelo k několika iteracím, kde by se na základě problémů v pozdějších fázích muselo vracet k dřívějším fázím a modifikovat je tak, aby výsledný nástroj souhlasil se všemi požadavky.

Analýza by zahrnovala sběr požadavků od koncových uživatelů, jejich vyhodnocení a seznámení s technologiemi potřebnými pro vývoj, jemiž jsou gramatika psaná v Xtextu, seznámení s konfigurátorem TesaTK, seznámení se s technologií modelování vlastností a její konfigurace. Dále by bylo třeba zvážit, jaký programovací jazyk by byl pro vývoj nejvhodnější a výběr odůvodnit. Odhadovaný čas analýzy by v takovém případě byl v řádu 80-120 hodin, tedy 10-15 pracovních dní.

Během návrhu implementace by bylo potřeba navrhnout parser, který dokáže z gramatiky v Xtextu dynamicky tvořit model vlastností, tedy namapovat typy vlastností na syntaxi jazyka Xtext. Dále by bylo potřeba vybrat vhodné prostředky pro jejich zobrazení, což úzce souvisí s výběrem programovacího jazyka a frameworku, který by se využil. Na základě objektové analýzy by bylo potřeba navrhnout strukturu aplikace. Odhadovaný čas návrhu implementace by se mohl opět pohybovat v řádu 80-10 hodin, tedy 10-15 pracovnách dní.

Implementace by zahrnovala vytvořit parser z gramatiky, celé uživatelské prostředí včetně zobrazení modelu vlastností, generátor konfigurace variant jako šablon v jazyce Tesa a export modelů variant. Dále by bylo třeba vypořádat se se všemi problémy, které by během implementace mohli nastat. Odhadovaný čas implemetace by se mohl pohybovat v rozmezí 1200-2000 hodin, tedy 150-250 pracovních dní.

Během testování by bylo třeba napsat jednotkové testy a otestovat tak celou aplikaci a opravit veškeré chyby, které by se při implementaci mohli objevit. Odhadovaný čas testování a oprav by se mohl pohybovat v rozmezí 800-1000 hodin, tedy 100-125 dní.

Pokud by všechny předešlé fáze dopadli úspěšně proběhla by validace všech fází a nástroj by se mohl začít používat. Celkový odhadovaný čas strávený vývojem této aplikace by se tak pohyboval okolo 2160-3220 hodin, tedy 270-400 dní. Pokud odhadneme cenu jedné programátorské hodiny na 1000kč, dostaneme odhadovanou cenu nástroje, která by činila 2,16-3,22 milionu korun. Za předpokladu, že by se na práci podílelo okolo 2-3 zaměstnanců, vývoj by trval zhruba 120 dní, tedy 6 měsíců. Důležité je si uvědomit, že vývoj by mohl probíhat v několika iteracích a časová náročnost by se mohla zvýšit i o 100% nebo více.

3.6 Výběr

V této části bude zhodnocen výběr nástroje, který bude v práci využit na základě ceny a funkcí, které jednotlivé nástroje nabízejí.

3.6.1 Cena

Cena je důležitým faktorem při výběru nástroje. Pure::variants je komerčně využívaný nástroj s velkým množstvím funkcí a podporou od svého vydavatele. To z něj činí nástroj s vysokou cenou. Cena jedné licence je 10 000 eur. Tuto licenci je třeba každý rok prodloužit. Cena tohoto prodloužení je 20% z pořizovací ceny, tudíž 2000 eur. Cenu za vlastní nástroj jsme odhadli

zhruba na 3 miliony korun. Tato cena se však může lišit v milionech korun od výsledné ceny, která by byla třeba na vývoj. Pokud budeme předpokládat, že nástroj bude využívat 5 lidí a bylo by tedy třeba zakoupit 5 licencí na nástroj pure::variants, návratovost tvorby vlastního nástroje by se pohybovala kolem osmi let.

Ostatní zmíněné nástroje jsou volně dostupné a jsou zdarma.

3.6.2 Funkce

V této části se budeme zabývat funkcemi, které jednotlivé nástroje nabízejí a jak tyto funkce souvisejí s účely bakalářské práce

pure::variants

Nástroj pure::variants splňuje veškeré požadavky na nástroj. Je možné v něm vytvořit model vlastností pomocí importu správně strukturovaných souborů ve formátu CSV. Z tohoto modelu lze vytvořit konfigurace, které jsou znázorněny v samostatném stromu, což zaručuje přehlednost oproti jiným nástrojům, kde se konfigurace tvoří přímo ve stromě modelu vlastností. Tyto konfigurace lze následně exportovat do souborů ve formátech XML a CSV. Nástroj také sám hlídá dodržení závislostí při tvorbě konfigurace.

XFeature

Nástroj XFeature umožňuje jednoduchou validaci konfigurací díky své reprezentaci vlastností pomocí XML. XFeature neumožňuje import ani export dat. Tato data by se ale dali zpracovat, díky jeho reprezentaci. Grafické zobrazení vlastností se liší od klasických grafických znázornění popsaných v [Cza98], což může být pro uživatele matoucí. Při velkém množství vlastností se také může strom stát nepřehledným.

Feature Modeling Plug-in

Nástroj Feature Modleing Plug-in umožňuje import a export modelu vlastností ve formátu XML. Umožňuje validaci tvořených konfigurací. Tyto konfigurace se však tvoří přímo ve stromě modelu vlastností, což může být nepřehledné.

Software Producl Lines Online Tools

Nástroj Software Product Lines Online Tools dokáže na základě modelu vlastností tvořit a validovat konfigurace exportovatelné do souborů formátu

CSV a XML. Velkým nedostatkem je však absence možnosti importu modelu vlastností. Model vlastností se dá importovat pouze z existující databáze uložené na stránkách nástroje.

3.7 Shrnutí

Pokud srovnáme všechny klady a zápory dostupných nástrojů, zjistíme, že pro účely bakalářské práce by byla využitelná většina z nich. Některé mají ovšem nedostatky, kvůli kterým nebudou použity. Největším nedostatkem je absence importu modelu vlastností v nástroji Software Product Lines Online Tools. Tento nedostatek tento nástoj vylučuje. Absence importu a exportu dat v nástroji XFeature je také problémem, který by se však dal obejít díky reprezentaci vlastností jako XML. Grafické znázornění vlastností však není příliš přehledné pro větší množství vlastností a je tudíž také nevhodný. Dalším nástrojem by mohl být Feature Modeling Plug-in, který podporuje většinu požadavků na nástroj, avšak tvoření konfigurací přímo ve stromě, ve kterém je zobrazován model vlastností, je nepřehledné a zároveň tvorba více konfigurací může být složitá. Všechny tyto problémy nemá nástroj pure::variants. Jediným nedostatkem tohoto nástroje je jeho vysoká cena. Pro splnění všech požadavků by bylo tedy možné vytvořit nástroj vlastní. Jeho vývoj by však delší a dražší, než používání nástroje pure::variants po dobu deseti let.

Z těchto důvodů je pure::variants nejvhodnějším nástrojem a bude použit pro potřeby bakalářské práce.

4 Generativní programování

Generativní programování je druh programování, který odděluje model od implementace. Realizuje se pomocí generátorů. Generátor využívá modelu jako šablony, ze které je generován výsledný kód. Generátory jsou založeny na modelech, které definují sémantiku.

4.1 Motivace

Motivace pro generativní programování vznikla současně se vznikem počítačů. Stroje nerozumí naší řeči a jejich funkce jsou pouhými elektrickými signály. Vznikla potřeba vymyslet způsob, kterým budeme procesoru předávat informace o tom, co se po něm vyžaduje. Psaní těchto instrukcí pomocí nul a jedniček je ale nepřehledné a pro člověka nesrozumitelné. Vznikly tedy první sady instrukcí procesorů, které byli pro člověka čitelné. Tyto instrukce byly překládány přímo do strojového kódu, podle kterého procesor pracoval. Tento překlad je právě generováním kódu. Instrukce jsou využity jako šablona a překladač generuje strojový kód na základě těchto instrukcí.

S rozmachem programování vznikl požadavek na vyšší abstrakci tak, aby se instrukce co nejvíc podobali lidské řeči. To vedlo ke vzniku velkého množství programovacích jazyků, překladačů a interpreterů, které tyto komplexnější instrukce překládají na strojový kód. Překladače jsou tedy generátory, které na základě šablony generují strojový kód.

Generativní programování je o návrhu a implementaci softwarových modulů, které je možné zkombinovat a generovat tak specializované a rozsáhlé systémy splňující specifické požadavky [Cza98]. Cílem je zmenšit mezeru zdrojovým kódem a konceptem, dosáhnout vysoké znovupoužitelnosti a adaptability software a zjednodušit správu velkého množství variant komponenty a zvýšit efektivitu [CEG98].

4.2 Generátory

Podle [Cza98] jsou pro generování použity dvě základní metody - *kompoziční* a *transformační*. Generátory založené na kompozici se nazývají *kompoziční generátory* a generátory založené na transformaci se nazývají *transformační generátory*. Oba dva typy generátorů si ukážeme na příkladě, kde budeme vytvářet instanci hvězdy z různých komponent. Hvězda je sestavena z modelu

vlastností, který obsahuje:

- počet cípů
- vnitřní poloměr
- vnější poloměr
- úhel popisující naklonění prvního cípu

Nyní si ukážeme, jak k vytvoření hvězdy přistupují obě dvě metody.

ZDE BUDE OBRÁZEK CZARNECKI FIGURE 52

4.2.1 Kompozice

V kompozičním modelu spojíme několik komponent dohromady, které tak vytvoří požadovaný celek. K tomu, abychom byli schopni generovat různé hvězdy, je potřeba mít sadu konkrétních komponent různých velikostí a tvarů.

ZDE BUDE OBRÁZEK CZARNECKI FIGURE 55.

Kruh je popsán pouze vnitřním poloměrem, zatímco cípy jsou popsány vnitřním poloměrem, jejich počtem a vnějším poloměrem. K tomu abychom sestavili čtyřcípou hvězdu, která je vidět na obrázku, je potřeba jeden kruh a čtyři cípy vybrané ze sady konkrétních komponent, na základě jejich vlastností.

ZDE BUDE OBRÁZEK CZARNECKI FIGURE 53.

Efektivnějším způsobem sestavení instance je použití *generativních komponent* místo konkrétních komponent. Generativní komponenta využívá abstraktního popisu komponent a generuje komponentu na základě popisu. Například místo celé sady všech možných kruhů a cípů potřebujeme dvě generativní komponenty, a to generativní kruh a generativní cíp. Generativní kruh má jako parametr vnitřní poloměr a generativní cíp má jako své parametry vnitřní poloměr, vnější poloměr a úhel.

ZDE BUDE OBRÁZEK CZARNECKI FIGURE 54

Konkrétní komponenty jsou následně vygenerovány a poskládány do požadovaného obrazce.

4.2.2 Transformace

Narozdíl od kompozice nespojujeme jednotlivé komponenty, ale provádíme určitý počet transformací, které vyústí v požadovaný výsledek. Není potřeba mít nadefinovanou sadu komponent, nebo jejich generování, ale je třeba mít nadefinované transformace, které můžeme s instancí provádět. V tomto příkladě jsou to transformace:

- přidej 4 cípy
- zvětš vnější poloměr
- otoč o 45 stupňů

ZDE BUDE OBRÁZEK CZARNECKI FIGURE 53

4.2.3 Shrnutí

I když se transformační generátor zdá být jednodušší, většina generátorů, které známe, jsou kompoziční. Příkladem takového generátoru může být editor GUI, který na základě grafického editoru, ve kterém poskládáme grafické komponenty dohromady, vygeneruje kód, který je popisuje.

Kompoziční generátor bude použit v této práci. Na základě vytvořené varianty z modelu vlastností bude generován kód jazyka tesa. Komponenty zde budou zaznamenány jako záznamy v souboru formátu XML. Z těchto komponent bude následně vygenerován výsledný kód.