Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Študijný odbor: Softvérové inžinierstvo

Miloslav Šípka

Podpora modelovania vlastností

Diplomová práca

Vedúci diplomovej práce: Ing. Valentino Vranić, PhD.
december 2005

Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný odbor: Softvérové inžinierstvo

Autor: Miloslav Šípka
Diplomová práca: Podpora modelovania vlastností
Vedenie diplomovej práce: Ing. Valentino Vranić, PhD.
december 2005

Modelovanie vlastností (feature modeling) je technika konceptuálneho modelovania používaná v metódach doménového inžinierstva na zachytenie spoločných a premenlivých vlastností konceptov domény. Táto práca má za cieľ špecifikovať navrhnúť a s časti aj implementovať CASE nástroj, ktorý zohľadní rôzne notácie a prístupy k modelovaniu vlastností, ako aj rôzne využitie takto vytvoreného modelu. Preto v úvode porovnáva spoločné a rozdielne črty jednotlivých prístupov k modelovaniu vlastností s cieľom vytvoriť kompaktnú notáciu, tak aby zohľadňovala čo najviac s analyzovaných prístupov. Výsledkom je notácia, ktorá kombinuje Czarneckého a Eiseneckerov prístup k modelovaniu vlastností, ktorý zavádza kardinality podľa vzoru UML a prístupov Gurpa, Bosha a Svahnberga, ktoré definujú v modeli vlastnosti body rozšírenia a pridávajú tak novú informáciu na hranu v diagrame modelu vlastností. Ďalej práca analyzuje dostupné prostriedky na podporu modelovania vlastností. Na základe takt zístakých poznatkov potom prezentuje metamodel, ktorý zohľadňuje jednotlivé analyzované prístupy a špecifikuje CASE nástroj, ktorý takýto metamodel podporuje.

Annotation

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree Course: Software engineering

Author: Miloslav Šípka
Thesis: Support for feature modeling
Supervisor: Ing. Valentino Vranić, PhD.
2005, December

Feature modeling is an important approach to capturing commonalities and variabilities in system families and product lines. This work focuses on a specification, design and implementation of CASE tool, which includes support of different notations and approaches to feature modeling. First it analyzes the commonalities and variabilities among different approaches to feature modeling to unite them into a consistent unit. Proposed notation basis combines the extended Czarnecki-Eisenecker notation (with UML like cardinalities) with Gurp-Bosh-Svahnberg edge decorations and identifies mandatory parts of the information associated with features. Further this work analyzes existing CASE tools for feature modeling. Based on this knowledge this work presents metamodel, which supports important parts of analyzed notations and specify CASE tool, which support this metamodel.

Prehlásenie

Prehlasujem, že prácu som vypracoval sám.

Miloslav Šípka

Obsah

1 Úvod

V úvodnej kapitole je predstavené modelovanie vlastností ako jedna s techník konceptuálneho modelovania. Taktiež bude predstavený samotný model vlastností ako aj zápis modelu vlastností pomocou takzvaných diagramov modelu vlastností. Vysvetlený bude aj význam jednotlivých prvkov takejto reprezentácie.

1.1 Modelovanie vlastností

Modelovanie vlastností (feature modeling) je technika konceptuálneho modelovania používaná v metódach doménového inžinierstva (časť 1.2) na zachytenie spoločných a premenlivých vlastností konceptov domény.

Cieľom doménového inžinierstva je minimalizovať náklady na znovupoužiteľné systémy a softvér všeobecne. Modelovanie vlastností pomáha identifikovať rôzne kontexty, v ktorých bude nejaký systém použitý a porozumieť im. Len na základe takéhoto porozumenia je možné odhadnúť náklady na jeho vývoj.

Ďalším prínosom modelovania vlastností je že nám umožňuje rozhodnúť sa, ako bude výsledný systém alebo ich skupina štrukturovaný. Na základe podrobného modelu vlastností je možné určiť, ktoré moduly systému sa budú vyvíjať ako znovupoužiteľé, a ktoré budeme vyvíjať špeciálne, len pre jeden konkrétny systém.

Iným prínosom je, že model vlastností umožňuje plánovať implementáciu domény. Na základe požiadaviek používateľa, ktorý má predstavu o jednom systéme z danej domény, môžeme podľa modelu vlastností rozhodnúť, ktoré súčasti systému budeme implementovať ako znovapoužiteľné, a ktoré nie. Taktiež model domény obsahuje niektoré požiadavky používateľa značne zovšeobecnené, tak aby sa dali aplikovať na celý rad systémov v doméne. Takýmto postupným implementovaním čoraz viac systémov pomaly implementujeme všetky dôležité vlastnosti domény.

Jednou z najdôležitejších úloh pri tvorbe znovupoužiteľného softvéru je identifikovať rôzne kontexty, v ktorých bude použitý a porozumieť im. Dokonalé porozumenie tejto oblasti pomáha odhadnúť náklady na vývoj softvéru. Ďalej umožňuje určiť ako bude výsledný softvér štruktúrovaný a do akej miery jednotlivé moduly pripraviť na opätovné použitie. Bolo by chybou implementovať ako znovupoužiteľný modul nejaký malý modul, ktorý sa v celej doméne vyskytuje len ako súčasť nejakého väčšieho modulu.

1.2 Model vlastností

V tejto časti bude rozobraný samotný model vlastností. Jedná sa o základ modelu vlastností a jeho prvkov. Zatiaľ nebude uvedená notácia použitá v prípade jednotlivých odlišných techník.

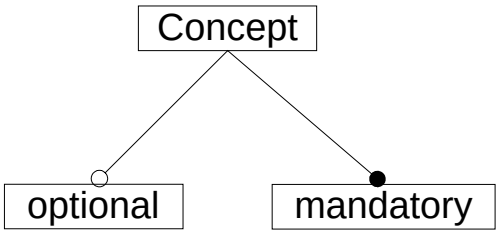
Model vlastností definuje sadu znovupoužiteľných a konfigurovateľných požiadaviek na systémy v doméne. Požiadavky sú všeobecne reprezentované ako vlastnosti. Model vlastností predpisuje, ktorá skupina vlastností je zmysluplná a ktorá nie. Ak teda vyberieme jednu skupinu vlastností, hovoríme, že sme vytvorili inštanciu. Podľa takto vytvorenej inštancie môžeme potom implementovať jeden systém.

Model vlastností je vo všeobecnosti graf vyjadrujúci vzťahy medzi jednotlivými vlastnosťami. V praxi sa však vôbec nepoužíva všeobecný graf ale len strom. Uzly stromu predstavujú vlastnosti a sú samozrejme pomenované. Koreň stromu sa reprezentuje koncept a ostatné uzly stromu (vnútorné uzly aj listy) reprezentujú vlastnosti konceptu. Nie je vylúčené, aby sa vnútorný uzol jedného diagramu vlastností stal konceptom iného. Potom sa samozrejme jeho vlastnosti uvá-

dzajú len v diagrame kde je koncept. V ostatných diagramoch je takýto uzol (vlastnosť) označený znakom ®.

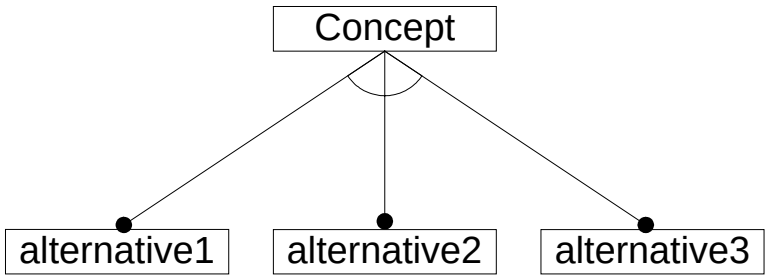
Hrany stromu reprezentujú vzťahy medzi jednotlivými vlastnosťami. Okrem toho v základom modeli hrany reprezentujú aj informáciu o tom, či je tá ktorá vlastnosť pre koncept povinná alebo voliteľná. V rozličných notáciách môže byť práve s hranami spojená aj ďalšia informácia ako napríklad kardinalita.

Ďalej sa v diagramoch vlastností rozlišuje medzi povinnými (mandatory)a voliteľnými (optional) vlastnosťami. Ak je vlastnosť povinná, tak musí byť vždy zahrnutá do opisu inštancie konceptu, ak je do opisu zahrnutý jej rodič. Na rozdiel od toho ak je vlastnosť voliteľná, tak môže byť zahrnutá do opisu inštancie konceptu, ak je do opisu zahrnutý jej rodič. Grafické označenie (Obr. 1)je plný kruh pre povinné vlastnosti (niekedy sa vynecháva) a prázdny kruh pre vlastnosti voliteľné.



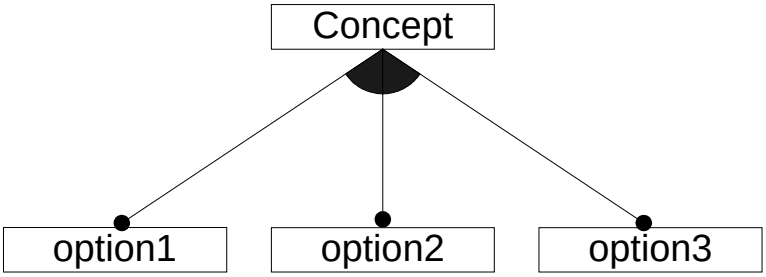
Obr. 1 Diagram vlastností s jednou povinnou a jednou volitelnou vlastnosťou

Diagram ďalej môže obsahovať jednu alebo viac skupín alternatívnych vlastností. Alternatívnosť (Alternative) teda vyjadruje fakt, že práve jedna vlastnosť zo skupiny vlastností musí byť zahrnutá do výsledného opisu inštancie konceptu, ak je jej rodič do tohto opisu zaradený tiež.. Grafické znázornenie je na Obr. 2.



Obr. 2 Diagram vlastností s jednou skupinou alternatívnych vlastností

Okrem alternatívy môžu byť vlastnosti zoskupené aj do disjunkcie. Disjunkcia (Or)vyjadruje fakt, že aspoň jedna z vlastností musí byť zahrnutá do výsledného opisu inštancie konceptu, ak je jej rodič do tohto opisu zaradený tiež.. Grafické znázornenie je na Obr. 3



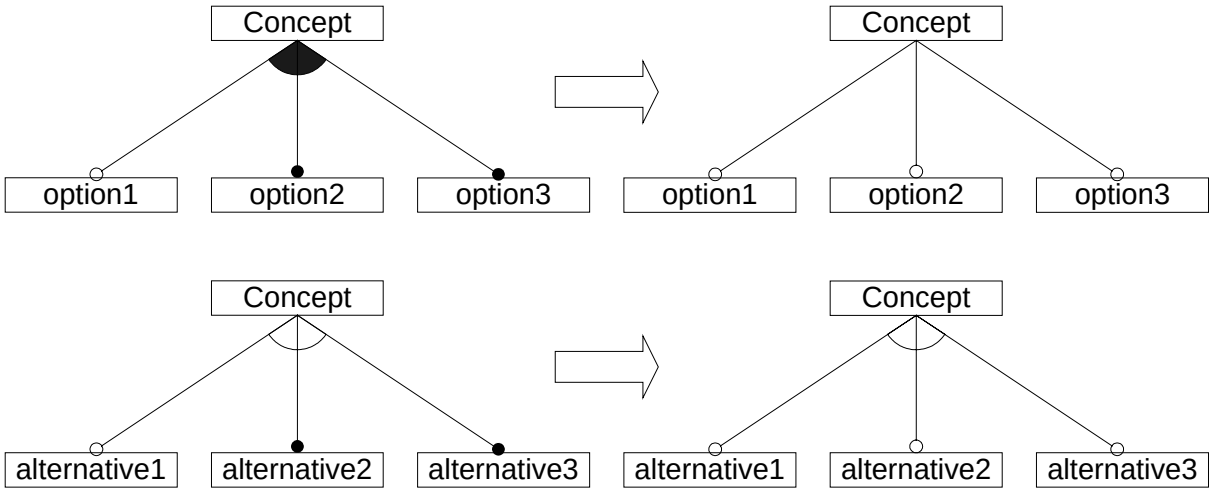
Obr. 3 Diagram vlastností s jednou skupinou disjunktívnych vlastností

Okrem týchto primitívnych konfigurácií, môžu povinná a voliteľná vlastnosť, alternatíva a disjunkcia vytvoriť konfiguráciu s osobitným významom. Je treba povedať, že existujú aj rôzne graficky vyjadrené konfigurácie s rovnakým významom.

Jednou z nich je aj tzv. voliteľná alternatíva teda spojenie alternatívy a voliteľnej vlastnosti. Napríklad Obr. 4 dole. Takáto konfigurácia definuje, že jedna alebo viac vlastností z alternatívy je voliteľná.

Ďalšou konfiguráciou je spojenie disjunkcie a voliteľnej vlastnosti. Napríklad Obr. 4 hore. Takáto konfigurácia definuje že jedna alebo viac vlastností z disjunkcie je voliteľná.

Ako už bolo spomenuté, niektoré konfigurácie sú navzájom ekvivalentné (Obr. 4). Preto bolo nutné zaviesť normalizáciu diagramu vlastností a normálnu formu diagramu vlastností (Czarnecki, Eisenecker 2000).



Obr. 4 Normalizácia diagramu vlastností podľa Chyba! Nenalezen zdroj odkazů.

Okrem vzťahov, ktoré sa graficky reprezentujú v diagrame vlastností, existujú aj vzťahy, ktoré je nutné reprezentovať mimo neho. Je to napríklad vzájomné vylúčenie. (Excludes). To indikuje, že vybranie jednej vlastnosti do opisu konceptu vylučuje vybranie inej. Opakom je vzťah závislosti resp. potrebnosti (Requires). Tá indikuje , že vybranie jednej vlastnosti do opisu konceptu implikuje vybranie inej.

2 Prístupy založené na modelovaní vlastností

V tejto kapitole budú predstavené rôzne prístupy a techniky, ktoré používajú aj modelovanie vlastností. Cieľom bude identifikovať ich podstatné vzájomné rozdiely.

2.1 FODA

Feature-Oriented Domain Analysis (FODA) je metóda, v ktorej bolo prvý krát použité aj modelovanie vlastností. Preto sa budem snažiť predstaviť túto metódu. Zameriavať sa budem len na fázy, ktoré využívajú modelovanie vlastností.

FODA je metóda určená na odhaľovanie a reprezentovanie spoločných vlastností vzájomne blízkych softvérových systémov. Vzájomne blízke systémy sú tie, ktoré vykonávanú podobnú funkciu a sú preto pravdepodobne nasadené v tej istej aplikačnej doméne.

Pod pojmom vlastnosť (feature) v tejto metóde sa rozumejú na jednej strane spoločné aspekty pozorovanej domény a na strane druhej aj rozdiely medzi systémami v tejto doméne. Vlastnosti sú použité aj na to aby reprezentovali povinné, voliteľné alebo alternatívne (v zmysle vybrania len jednej z alternatív) charakteristiky systémov v doméne.

Metóda FODA ďalej upravuje vzťahy medzi entitami modelu. Sú to tieto:

1. Agregácia a dekompozícia. Agregáciou sa rozumie abstrakcia skupiny entít do jednej všeobecnej. Napríklad škola vznikne agregáciou študentov, učiteľov atď. Postup v opačnom smere je zasa dekompozícia. Je to teda rozdelenie jednej entity na podentity, z ktorých dekomponovaná entita pozostáva.
2. Zovšeobecnenie a špecializácia. Zovšeobecnenie je pokrytie skupiny entít do jednej nadradenej, ktorá predstavuje spoločný základ pre všetky entity. Odlišnosti medzi jednotlivými entitami sú zanedbané.
3. Parametrizácia. Parametrizácia je všeobecne známa metóda, ktorá umožňuje reprezentovať jednu entitu viacerými spôsobmi v závislosti od hodnôt parametrov.

2.1.1 Analýza kontextu

Cieľom analýzy kontextu je definovať šírku domény. V tejto fáze sú skúmané vzťahy medzi doménou a externými prvkami. Výsledkom sú presne určené hranice domény a teda aj jej šírka. Metóda definuje ako výstupný dokument tejto fázy model kontextu.

Opis modelu

Model kontextu pozostáva z jedného alebo viacerých štruktúrovaných diagramov a diagramov toku údajov. Zaujímavý je najmä štruktúrovaný diagram. Ten zaraďuje analyzovanú doménu medzi ostatné domény. Tie sú postavené v hierarchii buď vyššie alebo nižšie ako analyzovaná doména, alebo sú na jej úrovni. Ak je analyzovaná doména časťou nejakej inej domény je táto zaradená vyššie. Nižšie zaradené domény (tzv. Poddomény) sú tie ktoré sa nachádzajú v rozsahu analyzovanej domény. Musí byť ale jasné, či tieto poddomény sú už dobre známe a ich analýza nebude ďalej predmetom dokumentu alebo sa budú ešte len analyzovať. Ostatné domény, ktoré sa nejakým spôsobom dotýkajú analyzovanej sú tiež v diagrame.

Diagram toku údajov reprezentuje na tejto úrovni toky údajov medzi analyzovanou doménou a ostatnými doménami ako aj elementmi v rámci analyzovanej domény. Oproti diagramom tokov údajov v iných použitiach musí diagram pri analýze kontextu vyjadrovať aj variabilitu tokov údajov. Táto musí byť vyjadrená napríklad sprievodným opisom. Taktiež je možné použiť viac diagramov, pričom každý reprezentuje iný kontext.

Z nášho pohľadu je zaujímavé, že variabilnosť dátových tokov, ktorú musí v tomto prípade diagram vyjadrovať, je zväčša dôsledkom rôznych vlastností aplikácií v analyzovanej doméne. Pre takýto prípad metóda vyžaduje popis pomocou vlastností. Preto aj modelovanie vlastností môže byť vykonávané v tejto fáze buď paralelne s modelovaním kontextu, alebo sa s mím môže striedať v jednotlivých iteráciách.

Pre každú entitu, ktorá vystupuje v spomenutých diagramoch by mali byť dostupné aspoň tieto informácie:

- Názov
- Pre funkcionálne entity popis ich funkčnosti
- Pre dátové entity popis obsahu dát, ktoré obsahujú
- Opis variability ako je napr. frekvencia výskytu, čas viazania a iné
- Štandardy a znovupoužiteľné komponenty
- Ďalšie fakty opisujúce entitu
- Zdroj informácií o entite

Použitie modelu

Jednou s najdôležitejších úloh pri tvorbe znovupoužiteľného softvéru je identifikovať rôzne kontexty, v ktorých bude použitý a porozumieť im. Dokonalé porozumenie tejto oblasti pomáha odhadnúť náklady na vývoj softvéru. Ďalej umožňuje určiť ako bude výsledný softvér štruktúrovaný a do akej miery jednotlivé moduly pripraviť na opätovné použitie. Bolo by chybou implementovať ako znovupoužiteľný modul nejaký malý modul, ktorý sa v celej doméne vyskytuje len ako súčasť nejakého väčšieho modulu.

2.1.2 Modelovanie domény

Pre doménu, ktorú sme vymedzili v analýze kontextu, budeme ďalej analyzovať rôzne aspekty. Modelovanie domény zahŕňa tri hlavné aktivity:

- Analýzu a modelovanie vlastností
- Tvorbu entitno-relačných modelov
- Funkcionálnu analýzu.

Modelovanie vlastností

Cieľom modelovania vlastností je pohľad z používateľovej strany na základné vlastnosti alebo lepšie povedané schopnosti systému. V porovnaní s inými technikami modelovania je odlišný najmä pohľad z používateľovej stany. Aj keď napríklad diagram toku údajov je tiež možné predstaviť z používateľovho uhla pohľadu, nevyjadruje všetky vlastnosti systému. (Môže len vyjadriť, aké dáta do systému vstupujú a aké vystupujú a poprípade aké procesy ich spracovávajú a kde sa dáta ukladajú). Na rozdiel od toho môže model vlastností vyjadriť aj funkcionálne aj nefunkcionálne požiadavky a to dokonca aj v jednom diagrame. V zásade sa schopnosti aplikácie dajú rozdeliť na tri skupiny:

1. Služby poskytované aplikáciou
2. Výkon aplikácie
3. Nároky na hardvér a softvér pre aplikáciu. (Alebo ak máme hovoriť o schopnostiach, tak na akých platformách bude výsledná aplikácia schopná behu)

Existujú aj iné pohľady (Kang, Cohen a i. 1990), podľa ktorých spadajú všetky vlastnosti do jednej so štyroch kategórií:

1. Cieľový operačný systém a prostredie (hardvér a iný inštalovaný softvér)
2. Samotné schopnosti aplikácie.
3. Technológia známa v doméne
4. Implementačné techniky

Tieto štyri kategórie predstavujú aj návod na úspešnú a najmä úplnú identifikáciu všetkých vlastností.

Okrem tohto pohľadu, existujú aj iné pohľady na kategorizáciu vlastností. Tie nie sú veľmi odlišné od pôvodnej metódy FODA. Napríklad prístup MBSE (Software Engineering Institute 1997) navrhuje takéto delenie:

- Vlastnosti kontextu (Context features). Sú to tie vlastnosti, ktoré opisujú celkovú úlohu aplikácie. Sem patria aj vlastnosti ako je presnosť, správnosť, výkon a podobne.
- Vlastnosti reprezentácií (Representation features) sú vlastnosti, ktoré vyjadrujú ako sú informácie prezerané používateľom, alebo v akej forme sú generované pre iné programy.
- Operačné vlastnosti (Operational features) sú vlastnosti, ktoré opisujú jednotlivé funkcie aplikácie teda to, čo aplikácia robí.

Model vlastností reprezentuje skupinu vlastností celej rodiny systémov jednej domény a vzťahy medzi nimi. Cieľom je vhodné štruktúrovane vlastností a vzťahov medzi nimi.

Pri modelovaní vlastností sa rozlišuje aj čas ich viazania. Ten súvisí najmä s implementáciou produktu. Metóda rozlišuje tri druhy viazania vlastností. Sú to:

1. Viazanie v čase kompilácie. Takéto viazanie je najviac obmedzujúce. Je odporúčané pre vlastnosti, ktorá do konceptu hodláme určite zaviesť. Dôvodom pre túto voľbu môže byť aj vyššia efektivita výsledného kódu.
2. Viazanie v čase zavádzania programu do pamäte. Takto viazané vlastnosti máme možnosť vybrať na začiatku behu programu ale ďalej už ostávajú nemenné.
3. Viazanie počas behu programu alebo dynamické viazanie. Takéto vlastnosti môžu byť menené ľubovoľne počas behu programu.

FODA vytváranie entitno - relačného modelu

Ďalším krokom podľa metódy FODA je identifikácia entít a relácií medzi nimi. Jej výsledkom je entitno-relačný model. Táto práca sa týmto krokom nebude zaoberať.

Funkcionálna analýza

V tejto časti sa identifikujú spoločné a rozličné funkcie aplikácií a programov v doméne. Táto časť priamo nadväzuje na predošlé. Základom pre funkcionálny model sú povinné vlastnosti identifikované pri vytváraní modelu domény a povinné entity identifikované pri vytváraní entitno-relačného modelu.

FODA modelovanie architektúry

Po analýze a vytvorení modelu domény je ďalším krokom tvorba konkrétneho riešenia problémov analyzovaných v predošlých častiach. Takýto model architektúry je vlastne model návrhu aplikácie na veľmi vysokej úrovni abstrakcie.

Model architektúry musí zodpovedať výsledkom, ku ktorým sme dospeli v predošlých krokoch, teda problémom v analyzovanej doméne. Na druhej strane musí byť dostatočne variabilný a pripravený na to, aby vedel odraziť prípadnú zmenu v doméne, ktorú reprezentuje. Preto je aj samotný model vytváraný v niekoľkých vrstvách abstrakcie. Napríklad **Kang, Cohen a i. 1990** navrhuje takáto vrstvy:

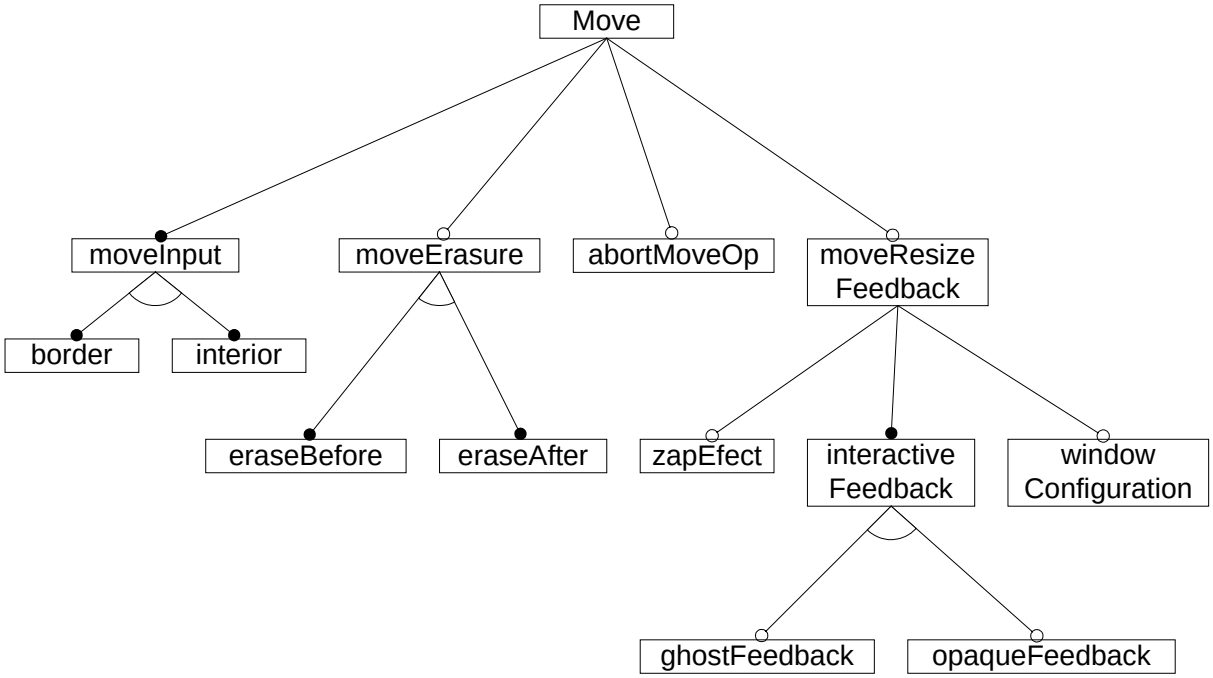
1. Architektúra domény zobrazuje rôzne procesy v doméne a komunikáciu medzi nimi.
2. Vrstva modulov domény **Kang, Cohen a i. 1990** ju nazýva Domain Utilities Layer pretože navrhuje aplikáciu rozdeliť na viacero podprogramov teda utilít.) reprezentuje zoskupenie funkcií do jednotlivých modulov.
3. Vrstva spoločných modulov (Common Utilities Layer) pridáva do modelu funkcie a moduly spoločné pre viacero domén.
4. Systémová vrstva reprezentuje to, čo nám dáva k dispozícii cieľový operačný systém alebo zvolené implelemnačné prostredie.

Metóda samotná sa zameriava na tie najvyššie vrstvy. Hlavným cieľom je identifikácia všetkých funkcií a objektov systému a ich vhodné začlenenie do spolupracujúceho systému viacerých balíkov.

2.1.3 Príklad modelu vlastností

V predošlých kapitolách som opísal metódu FODA z pohľadu modelovania vlastností. Venoval som sa len tým krokom, v ktorých je práve modelovanie vlastností použité ako nástroj. Ďalej som naznačil, ako sa výsledný model vlastností používa v ďalších krokoch.

Na Obr. 5 je príklad diagramu vlastností pre operáciu hýbania okna. Ten vznikol analýzou viacerých grafických systémov ako sú Windows od spoločnosti Microsoft alebo jemu podobné produkty od spoločnosti Sun.



Obr. 5 Diagram vlastností pre operáciu Move (hýbanie okna)

Najskôr aspoň pre niektoré vlastnosti prednesieme kompletné údaje. Ich štruktúru navrhuje už Kang, Cohen a i. 1990 a ukážka kompletného katalógu vlastností je napríklad Cohen, Stanley a i. 1992.

Vlastnosť `moveInput`

- Opis: Reprezentuje, ktoré časti okna slúžia ako vstup pre operáciu hýbania s oknom
- Zdroj[1]: Windows, X11

Vlastnosť `border`

- Opis: Okno je možné hýbať len ak klikneme a pretiahneme rám alebo vopred definovanú časť okna
- Zdroj: Windows

Vlastnosť `interior`

- Opis: Okno je možné hýbať ak klikneme a pretiahneme ľubovoľný bod v okne
- Zdroj: X11

Vlastnosť `moveErasure`

- Opis: Reprezentuje zmazania obrazu okna z jeho pôvodnej pozície
- Zdroj: Windows, X11

Vlastnosť `abortMoveOp`

- Opis: Schopnosť zrušiť operáciu presúvania okna počas jej vykonávania napr. stlačením preddefinovanej klávesy
- Zdroj: Windows

Obdobne by sme pokračovali pre všetky identifikované vlastnosti. My sa však ďalej zameriame na opis tohto čiastočného diagramu vlastností. Napríklad vlastnosť `moveInput` je podľa diagramu povinná pre operáciu `Move`. To znamená, že každá implementácia operácie na posúvanie okna musí mať jasne definované ako sa vyvoláva. Teda, či stačí stlačiť tlačidlo myši na hociktorom bode okna a potiahnuť, alebo treba stačiť tlačidlo myši na definovanom okraji okna. Z diagramu sa dá ďalej vyčítať, že tieto vlastnosti sa nedajú skombinovať. My ale môžeme požadovať, aby sa takéto správanie dalo nastaviť a preto by náš systém mal podporovať obe vlastnosti. Preto by sa zdala alternatíva medzi `border` a `interior` nesprávna a mala by sa nahradiť disjunkciou. Takýto zásah do modelu však nie je správny. Namiesto neho sa zmení typ viazania z doby kompilácie programu na dobu behu programu.

Zvládnuť po implementačnej stránke rôznorodosť, ktorú modeluje diagram vlastností a k tomu ešte pridané doby viazania, je samo o sebe veľmi obtiažne. Kang, Cohen a i. 1990 sa tejto problematike vôbec nevenujú. Navrhujú len reprezentáciu celého diagramu vlastností aj s príslušnými obmedzeniami v jazyku Prolog a načrtávajú algoritmus, ktorý má overiť a doplniť výber vlastností pre jednu implementáciu. Podstatou je vybrať základné vlastnosti systému a potom nechať automat nech na základe dodaných pravidiel výber doplní o vlastnosti, ktoré sú požadované. Tiež je potrebné nechať automat, aby na základe pravidiel vybral vlastnosti, ktoré sú v rozpore so základným výberom. Ak majú tieto dve množiny spoločný prvok nastáva konflikt.

[1] V tomto kontexte sa rozumie systém v ktorom bola daná vlastnosť pozorovaná.

Rozšírením metódy FODA je Feature-Oriented Reuse Method (FORM). Metóda určuje ako sa model vlastností použije pri návrhu aplikácie. Hlavná myšlienka spočíva v tom, že vlastnosti charakterizujú jednotlivé aplikácie v doméne a kód, ktorý implementuje tieto vlastnosti, môže byť zaradený do modulu a znova použitý. Metódu podrobnejšie opisuje Kang, Kim a i. 1998.

2.2 Generatívne programovanie

Generatívne programovanie je metóda návrhu a implementácie softvérových modulov, ktoré majú byť kombinované s cieľom vytvoriť špecializovaný a vysoko optimalizovaný systém spĺňajúci špecifické požiadavky. Cieľom je

1. Zmenšiť konceptuálnu priepasť medzi programom a konceptom domény.
2. Dosiahnuť vysokú znovupoužiteľnosť a adaptabilitu.
3. Jednoducho spravovať veľa rôznych variantov komponent.
4. Zvýšiť efektivitu.

Na naplnenie týchto cieľov sa používa:

1. Oddelenie záujmov. Je treba zabrániť, aby ten istý kód programu riešil naraz viac ako jeden problém. Preto sa v generatívnom programovaní každý záujem (riešenie konkrétneho problému) izoluje do oddelenej časti kódu.
2. Parametrizácia rozdielov.
3. Analýza a modelovanie závislostí a interakcie. Zväčša nie je povolená ľubovolná hodnota parametra. Taktiež môže nejaká hodnota parametra určovať hodnotu iného. Tieto závislosti predstavujú horizontálne závislosti, pretože sa vyskytujú medzi parametrami na tej istej úrovni.
4. Oddelenie priestoru problému od priestoru riešenia. Oba priestory majú rôznu štruktúru a preto sa vytvára mapovanie medzi nimi. Takémuto mapovaniu sa hovorí aj vertikálna konfigurácia, pretože predstavuje interakciu medzi parametrami na rôznej úrovni abstrakcie.
5. Eliminácia nadbytočnosti a doménovo špecifická optimalizácia. Definovaním komponentov staticky (v čase kompilácie) je možné ušetriť mnoho nadbytočného overovania.

Samotné modelovanie vlastností je prevzaté z metódy FODA (časť 2.1). Na rozdiel od originálnej metódy sú k vlastnostiam pridané odlišné sprievodné informácie. Sú to tieto:

- Zdôvodnenie. Každá vlastnosť by mala obsahovať aspoň stručné zdôvodnenie svojho výskytu. Taktiež každá voliteľná vlastnosť by mala byť doprevádzaná informáciou o tom, kedy ju zvoliť a aké sú obmedzenia a prípadné podmienky jej voľby.
- Zainteresovaná osoba a program. Je rozumné označiť, kto má záujem na implementovaní vlastnosti. V prípade komponentu to môže byť aj program, ktorý komponentu používa.
- Príklad systému, ktorý vlastnosť používa. Táto informácia je ekvivalentná s informáciou o zdroji, ktorú identifikuje Kang, Cohen a i. 1990.
- Obmedzenia (Constraints). Obmedzenia predstavujú pravidlá medzi voliteľnými vlastnosťami všeobecne aj v rôznych diagramoch vlastností. Existujú dve dôležité obmedzenia a to vzájomné vylúčenie (mutual-exclusion) dvoch voliteľných vlastností a jeho opak (requires).
- Pravidlá pre parametre (default dependency rules). Predstavujú vodítko pri vytváraní inštancie modelu.

Aj obmedzenia aj pravidlá môžu byť horizontálne alebo vertikálne. Horizontálne pravidlá predstavujú pravidlá medzi vlastnosťami na tej istej úrovni abstrakcie. Na rozdiel od toho vertikálne pravidlá prepisujú vlastnosti na vyšších úrovniach abstrakcie na vlastnosti na nižších (implementačných) úrovniach.

- Availability sites, binding sites, a binding mode. Availability sites predpisujú, kde a komu je daná vlastnosť prístupná. Pomocou availability sites môžeme napríklad modelovať, aké položky menu sa zobrazia, pre koho, v akom čase a kontexte. Binding sites predpisujú, kedy môže byť daná vlastnosť viazaná a kým. Binding mode predpisuje, či je daná vlastnosť viazaná staticky, alebo sa jej viazanie môže s časom meniť, prípadne sa daná vlastnosť viaže len na dobu jej potreby (dynamicky).
- Otvorenosť / Zatvorenosť. Tento atribút predpisuje, či sa vo vlastnosti očakáva ďalšie pridávanie podvlastností, alebo je už jej analýza dokončená.
- Priorita.

2.3 Multiparadigmový návrh s modelovaním vlastností

Multiparadigmový návrh definuje doménu ako predmet záujmu. Rozlišujú sa dva druhy domén podľa ich rola vo vývoji softvéru. Aplikačná doména je doména, v ktorej je aplikovaný proces vývoja softvéru. Doména riešenia je doména, v ktorej sú vyjadrené riešenia. Zväčša je riešenie vyjadrené v konkrétom programovacom jazyku (jazykoch).

Modelovanie vlastností pre multiparadigmový návrh je založené na modelovaní vlastností podľa Czarnecki, Eisenecker 2000. Ten je tiež odvodený z metódy FODA, ktorá bola z pohľadu modelovania vlastností opísaná v časti 2.1. Cieľom modelovania vlastností je práve nájsť súlad a mapovanie medzi aplikačnou doménou a doménou riešenia. Do pôvodného modelovania vlastností Vranić 2004 zavádza niektoré nové prvky:

1. Definuje dva typy zdôvodnenia. Zdôvodnenie samotného zaradenia vlastností do modelu (tzv. Presence rationale) a zdôvodnenie, prečo je (alebo nie je) vlastnosť vložená do konkrétnej inštancie kontextu (tzv. Inclusion rationale).
2. Na rozdiel od FODA definuje okrem času viazania aj mód viazania a rozširuje počet časov viazania.

Čas viazania je určený časmi viazania dostupnými v jednotlivých doménach. Podľa Coplien 1999 sú typické časy viazania tieto:

- Viazanie pri samotnom písaní programu (Source time). Samotný programátor pri písaní kódu aplikácie vie, ktoré vlastnosti má implementovať.
- Viazanie pri kompilácii programu (Compile time). Tu sú rozhodnutia o viazaní realizované kompilátorom. Napríklad prekladač vyberá, ktorú z preťažených metód volá.
- Viazanie v čase linkovania programu (Link time). V tomto prípade robí rozhodnutia o viazaní linker.
- Viazanie v čase zavádzania (spúšťania) programu, tzv. Load time.
- Viazanie v čase behu programu (Run time).

Definované len dve módy viazania a to statický a dynamický. Statický mód viazania znamená, že vlastnosť nesmie byť viazaná v čase vykonávania programu.

Aby sa vyhol konfliktom pre rovnakých menách vlastností zavádza Vranić 2004 konvenciu obdobnú s konvenciou používanou modernými programovacími jazykmi, ktoré umožňujú definovať priestory mien. Konvencia zápisu má tvar

Concept.Feature1.Feature2....FeatureN

Dokonca zavádza aj odlíšenie domény, z ktorej daná vlastnosť je

Domain:Concept.Feature1.Feature2....FeatureN

Okrem tohto Vranić 2004 zavádza zápis obmedzení pomocou predikátovej logiky. Samotné logické výrazy zapisuje len s menami vlastností alebo konceptov. Výskyt mena vlastnosti ,napríklad f, alebo konceptu v takomto výraze nahradzuje formálne správnu formu zápisu predikátom is_in_instance(f). Predikát is_in_instance(f) je pravdivý, ak je vlastnosť f zahrnutá do inštancie konceptu a nepravdivý, ak nie je. Teda formálne správny zápis

is_in_instance(Feature1) $\wedge$ is_in_instance(Feature2)

nahradzuje zjednodušeným zápisom

Feature1 $\wedge$ Feature2

Ďalej zjednodušuje aj takýto zápis

Concept.Feature1.Feature2a $\wedge$ Concept.Feature1.Feature2b

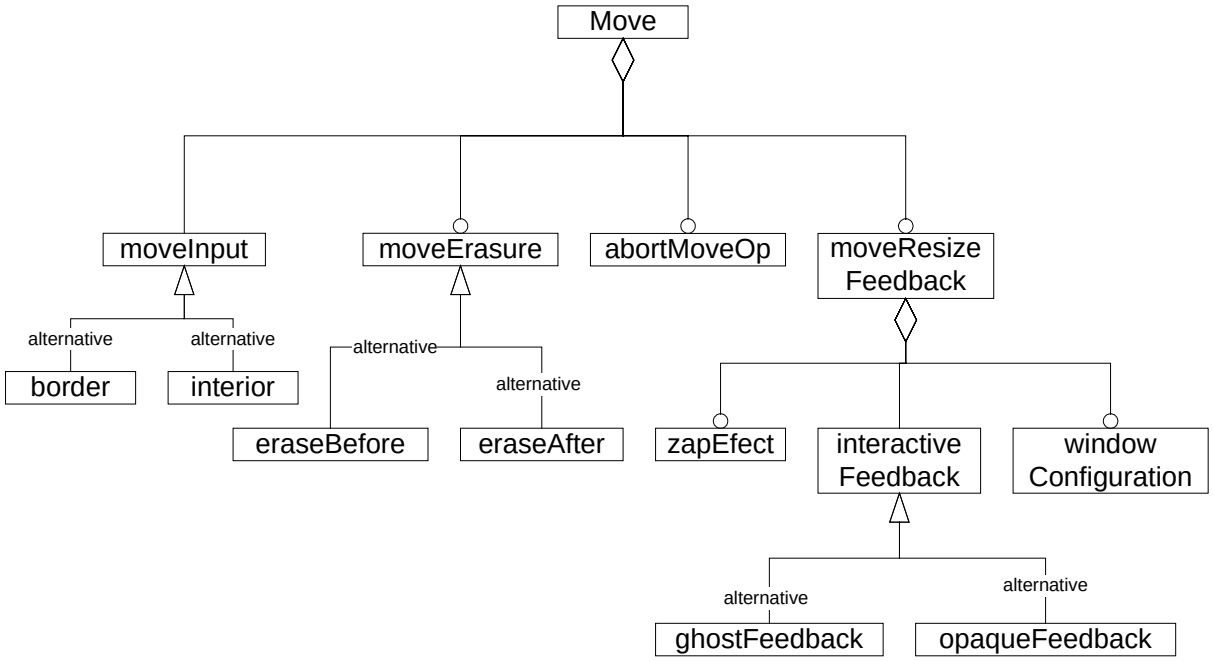
na

Concept.Feature1.(Feature2a $\wedge$ Feature2b)

2.4 FODAcom

Ďalšie práce (Vici, Argentieri a i. 1998, Giss, Favaro a i. 1998 a Gurp, Bosh a i. 2001) sa snažia priblížiť modelovanie vlastností k Jazyku UML. UML je v súčastnej dobe už štandardom a v dobe, kedy boli tieto práce vytvorené sa ním stával. Okrem toho má priblíženie modelovania vlastností k UML aj ďalšie výhody. Tou najmarkantnejšou je to, že UML je masívne podporovaný CASE nástrojmi.

V prvom priblížení (Vici, Argentieri a i. 1998) bolo UML použité skôr pre nedostupnosť nástroja pre modelovanie vlastností definované podľa metódy FODA a naviac takéto priblíženie prinieslo veľmi zaujímavé rozšírenie diagramu vlastností. Po vzore UML sa totiž medzi vlastnosťami kategorizoval vzťah na zovšeobecnenie a agregáciu. Takéto vzťahy používa UML v statických diagramoch, ako je napríklad diagram tried. Na Obr. 6 je podľa prístupu definovanom vo FODAcom prekreslený diagram z Obr. 5. Jediný prvok, v ktorom sa líši takýto diagram vlastností od diagramu tried UML, je to, že je tu ponechaná informácia o tom, či je vlastnosť povinná alebo nie. To samozrejme predpokladá, že sme stotožnili vlastnosť (entitu z diagramu vlastností) s triedou (entita z diagramu tried UML). Czarnecki, Eisenecker 2000 však ukázali, že vlastnosť je pojem oveľa širší, ako je pojem trieda a nie všetky vlastnosti sa dajú stotožniť s triedou.



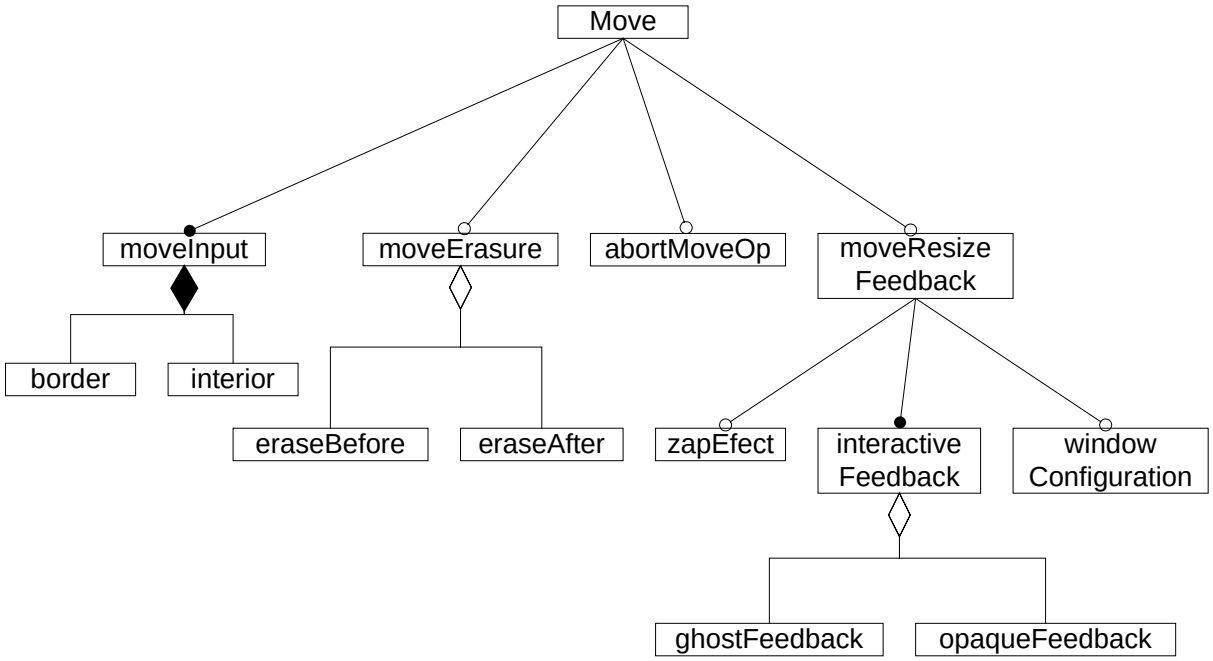
Obr. 6 Diagram vlastností pre operáciu Move (hýbanie okna) podľa FODAcom

2.5 FeatuRSEB

RSEB (Reuse-Driven Software Engineering Business) je prípadmi použitia riadený proces vývoja softvérových systémov. Napriek tomu je to metóda doménového inžinierstva. Klasické postupy doménového inžinierstva sú v tejto metóde nahradené. Pôvodne sa RSEB zameriaval len na vývoj architektúry systému, na vývoj znovupoužiteľných komponentov pre takýto systém. RSEB nezachytával pôvodne variabilitu konfigurácií ako ju zachytáva FODA pomocou modelu vlastností. Práve na tento účel, teda na zachytenie spoločných a odlišných vlastností, bolo do RSEB pridané i modelovanie vlastností.

Nová metóda FeatuRSEB (Giss, Favaro a i. 1998)je rozšírením pôvodnej koncepcie RSEB. Do pôvodnej koncepcie prináša nové prvky práve z doménového inžinierstva, ako je aj modelovanie vlastností. Zaujímavé je aj použitie modelu vlastností. Po úspešnej analýze domény metódou RSEB máme k dispozícii diagram prípadov použitia, tak ako je známy z UML. Model vlastností predstavuje akúsi mapu, ktorú použijeme pri ďalšej práci s prípadmi použitia. Takáto mapa vyjadruje, čo môže byť skombinované v jednom konkrétnom systéme z domény.

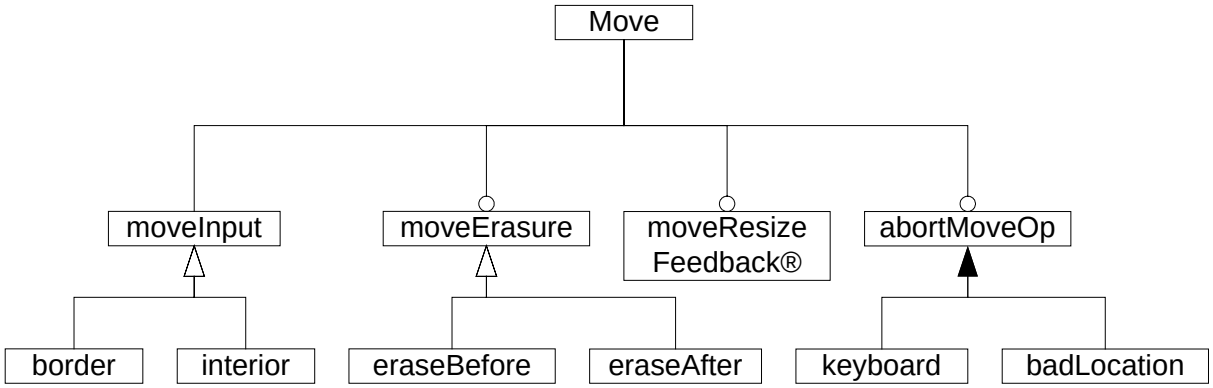
FeatuRSEB prináša do modelu vlastností aj jednu novinku. Je ňou bod rozšírenia (variation point). Bod rozšírenia vnáša parametrizáciu do modelu domény abstraktným spôsobom. Okrem toho sa v bode rozšírenia definuje aj doba (alebo mód viazania). Dostupné sú dynamické viazanie (plný kosoštvorec) a statické viazanie (prázdny kosoštvorec). Na Obr. 7 je príklad diagramu vlastností podľa štandardov FeatuRESB. Je na ňom možné vidieť napríklad, že vlastnosti border a interior sú zjemnením vlastnosti moveInput. Toto je príklad jedného bodu rozšírenia s dynamickým viazaním. V tomto prípade je identifikovaná vlastnosť moveInput ako abstrakcia dvoch konkrétnych vlastností border a interior.



Obr. 7 Diagram vlastností pre operáciu Move (hýbanie okna) podľa FeatuRESB

2.6 Vyjadrenie variability v radoch softvérových produktov

Ďalšou prácou, ktorá rozširuje modelovanie vlastností o prvky jazyka UML je Gurp, Bosh a i. 2001. Pri rozširovaní modelu vlastností berú za základ práve model definovaný v metóde FeatuRSEB (Giss, Favaro a i. 1998). Novinkou je takzvaná disjunktívna (OR) špecializácia. Jej notácia je na Obr. 7. Pri doterajšej alternatívnej špecializácii sa predpokladalo, že pri vytváraní inštancie konceptu sa vyberie jedno jej zjemnenie, či už v čase kompilácie alebo v čase behu programu. Gurp, Bosh a i. 2001 odstraňuje toto obmedzenie. Napríklad hýbanie okna by sme mohli hypoteticky prerušiť klávesnicou (napríklad stlačením ESC), ale aj posunutím okna na nejakú vopred definovanú neplatnú pozíciu (napríklad úplne mimo obraz). Obe tieto vlastnosti sú ale zovšeobecnením vlastnosti abortMoveOp.

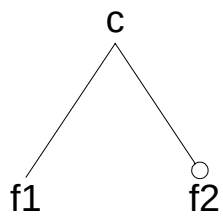
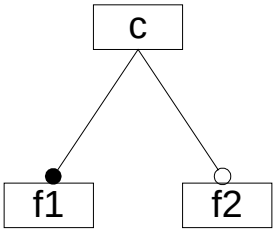
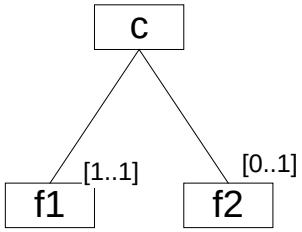
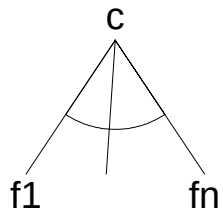
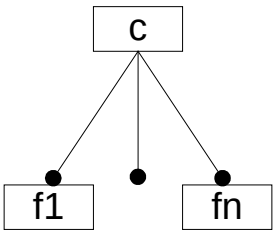
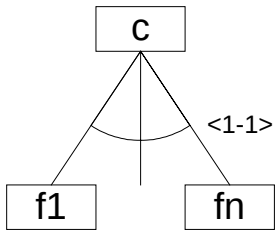
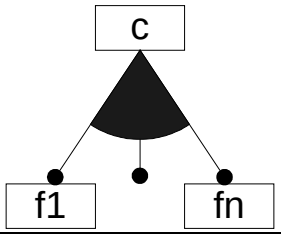
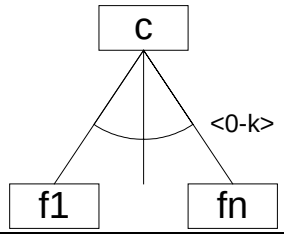
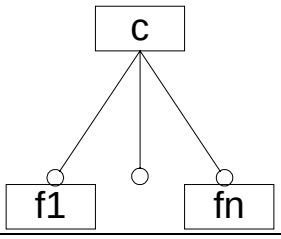
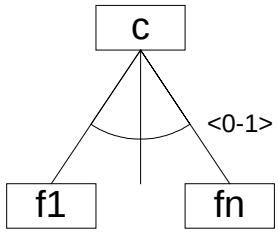


Obr. 8 Diagram vlastností pre operáciu Move (hýbanie okna) podľa notácie Gurp, Bosh a i. 2001

2.7 Rozšírenie FODA notácie o kardinalitu

Riebisch, Böllert a i. 2002 a postupne aj iné práce (Bontemps, Heymans 2004, Czarnecki, Helsen a i. 2004) rozširujú pôvodný model vlastností o kardinalitu podľa vzoru UML. Na Obr. 9 je uvedené porovnanie pôvodnej FODA notácie, jej prvého rozšírenia a rozšírenia o kardinalitu. Z Obr. 9 vidno, že zavedením kardinality sa v modely vlastností nahrádza informácia o povinnej

alebo voliteľnej vlastnosti a o disjunktívnych alebo alternatívnych vlastnostiach. Táto informácia je už obsiahnutá v samotnej kardinalite.

Pôvodná notácia FODA (Kang, Cohen a i. 1990)	**Rozšírená notácia FODA (Czarnecki, Eisenecker 2000)**	**Notácia rozšírená o kardinalitu (Czarnecki, Helsen a i. 2004)**
Povinná a voliteľná vlastnosť 	Povinná a voliteľná vlastnosť 	Povinná a voliteľná vlastnosť 
Alternatívne vlastnosti 	Skupina XOR 	Skupina s kardinalitou <1-1> 
	Skupina OR 	Skupina s kardinalitou <0-k> 
	Skupina XOR s voliteľnými vlastnosťami 	Skupina s kardinalitou <0-k> 

Obr. 9 Porovnanie rozdielnych notácií v diagramoch vlastností

2.8 Reprezentácia modelu vlastností v Prologu

Prvú formálnu reprezentáciu modelu vlastností navrhli už autori techniky modelovania vlastností FODA (Kang, Cohen a i. 1990). Jedná sa o reprezentovanie stromu vlastností pomocou predikátov jazyka Prolog. Príkladom takejto reprezentácie je predikát

```
daFeature(exposeAfterMove,
      'Expose the window at the end of a move operation.',
```

```
optional,                                   %type
moveWindowOp,                               %parent
[requires(overlappedLayout)],               %constraint
'SunView window system experience').        %source
```

Predikát postupne reprezentuje meno, opis, typ (povinná / voliteľná), rodičovskú vlastnosť, obmedzujúce podmienky a zdroj vlastnosti.

Prolog nebol pre reprezentáciu zvolený náhodou. Autori preň navrhli algoritmus na overenie, či je nejaká predom vybraná konfigurácia v zhode s definovanými obmedzeniami alebo nie. Jeho činnosť sa dá zhrnúť do týchto krokov:

1. Vstupom je množina používateľom vybraných (označených) vlastností
2. Označ vlastnosť za platnú ak:
    a. Je označená ako platná
    b. Je povinná
    c. Nie je označená ako neplatná
    d. Je požadovaná platnou vlastnosťou.
3. Označ vlastnosť ako neplatnú ak je vylúčená platnou vlastnosťou.
4. Ak existuje vlastnosť označená ako neplatná aj ako platná indikuj chybu
5. Zaruč správne vybranie alternatív
    a. Najmenej jedna alternatívna vlastnosť musí byť vybraná ako platná
    b. Nie viac ako jedna alternatívna vlastnosť môže byť vybraná ako platná

Je treba povedať, že Prolog nezapisuje program v zmysle postupnosti krokov, ale ako množinu pravidiel, kde na poradí samozrejme nezáleží. Správnou implementáciou týchto pravidiel môžeme teda overiť, či náš pôvodný výber vlastností zodpovedá obmedzeniam, ktoré sme definovali pre náš model vlastností.

ďalšou z nožnosrí ako reprezentovať model vlastností predstavili Zhang, Sun a i. 2004. Im sa podarilo vyjadriť sémantiku modelu vlastností v jazyku Alloy. Alloy (Jackson 2002) je štruktúrovaný jazyk na vyjadrenie modelov postavený na báze predikátovej logiky prvého rádu. Na rozdiel od Prologu je určený na formálny opis modelov. Preto sa na formálny opis modelov vlastností hodí viac ako Prolog.

2.9 Zhrnutie

Druhá kapitola predstavila rôzne prístupy k modelovaniu vlastností ako aj k modelu vlastností samotnému a jeho notácii.

Pre modelovanie vlastností položila základ metóda FODA (časť 2.1). Na základoch, ktoré položila metóda FODA, stavajú aj ďalšie metódy napríklad FORM. Podstatné rozšírenie FODA ohľadom modelovania vlastností predstavuje notácia a prístup v modelovaní vlastností pre generatívne programovanie (časť 2.2).

Ďalším rozšírením modelovania vlastností je pridanie predikátovej logiky na zápis obmedzení, ktoré sa kladú na model vlastností. Tento koncept priniesol modelovanie vlastností pre multiparadigmový návrh (časť 2.3). Dovtedy boli obmedzenia len striktne kategorizované napríklad len na dva typy obmedzení, a to vzájomné vylúčenie dvoch vlastností a jeho opak, teda keď jedna vlastnosť implikuje zahrnutie druhej do inštancie konceptu.

Ďalšími zmenami, ktorými prešlo modelovanie vlastností od jeho vzniku, je jeho približovanie k UML. Prvé takéto rozšírenie bolo prezentované ako metóda FODAcom (časť 2.4). Táto metóda priniesla do modelu vlastností nové hodnotenie vzťahov medzi vlastnosťami. Jedná sa konkrétne o vzťah zovšeobecnenia a agregácie.

Na metódu FODAcom nadväzuje ďalšia metóda a to FeatuRSEB (časť 2.5) táto metóda vnáša do modelovania vlastností body rozšírenia a viazanie v týchto bodoch.

Posledným prebratím prvkov UML do modelovania vlastností predstavuje práca o notácii variability v líniách softvérových produktov (časť 2.6). Tá si všíma, že nejakú abstraktnú vlastnosť môžeme zjemniť aj viac ako jednou zjemňujúcou vlastnosťou. Je to teda zovšeobecnenie bodu rozšírenia z metódy FeatuRSEB.

Okrem prevzatia kategorizácie vzťahov medzi vlastnosťami je možné z jazyka UML prevziať aj vyjadrenie kardinality (z jazyka UML sa preberá notácia a nie myšlienka samotná). Takémuto doplneniu modelu vlastností sa venuje časť 2.7.

3 CASE prostriedky podporujúce modelovanie vlastností

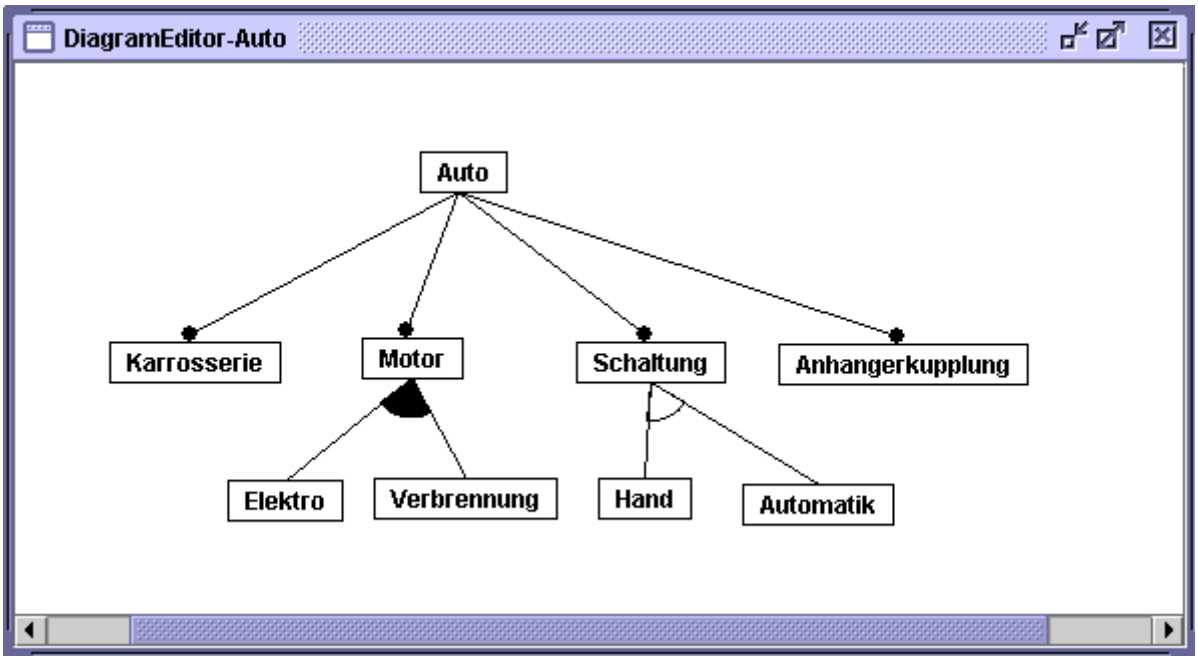
V dnešnej dobe je aj tá najdokonalejšia metóda používaná v akejkoľvek fáze vývoja softvéru silne závislá od podporných CASE prostriedkov. Cieľom tejto kapitoly je podať prehľad dostupných prostriedkov pre modelovanie vlastností. Na základe tohto prehľadu zistiť, aké postupy pri modelovaní vlastností preferujú ostatní autori.

Prostriedky pre modelovanie vlastností sa dajú rozdeliť v zásade na dve skupiny. Do prvej skupiny patria prostriedky, ktoré boli vytvárané špeciálne pre podporu modelovania vlastností. To samozrejme nevylučuje, že môžu byť použité aj na podporu iných metód pri návrhu softvéru. Druhou skupinou sú prostriedky, ktoré boli vytvorené s cieľom čo najväčšej všeobecnosti. Do takýchto prostriedkov je potom možné, s vynaložením väčšieho či menšieho úsilia integrovať aj modelovanie vlastností. Asi najznámejším z takýchto prostriedkov je komerčný produkt Visio spoločnosti Microsoft. Autori tohto produktu ponúkajú rozsiahle možnosti pri vytváraní diagramov a definovaní vlastných prvkov v diagramoch. Preto je možné použiť program Visio i na kreslenie diagramov modelov vlastností.

3.1 Captain Feature.

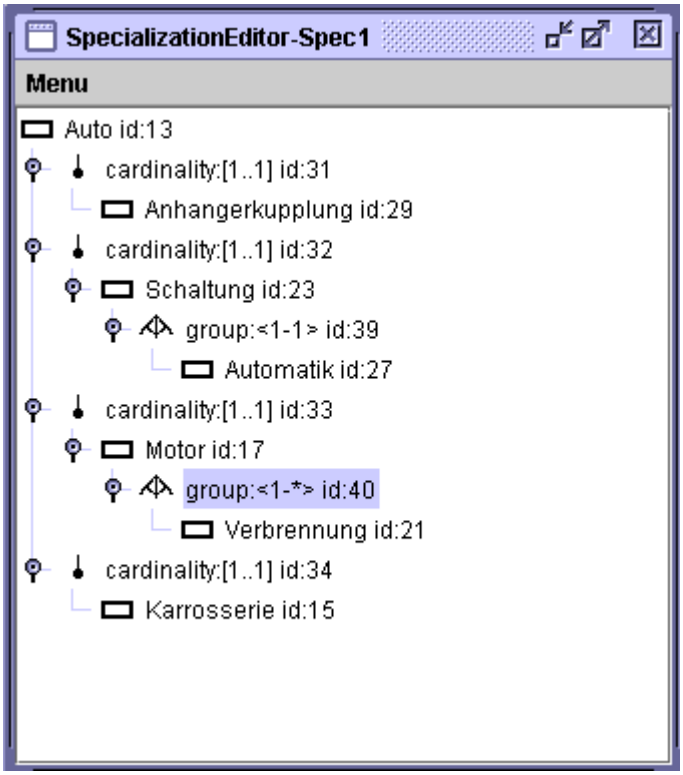
Captain Feature (Bednasch a i. 2005) je nástroj určený výlučne na podporu modelovania vlastností.

Pre diagram modelu vlastností program podporuje dve notácie. Základnou notáciou v programe je notácia podľa FODA rozšírená o kardilanitu, aj keď v starších verziách nebola táto notácia podporovaná vôbec. Druhou podporovanou notáciou je notácia pre Czarneckého generatívne programovanie (Czarnecki, Eisenecker 2000). Tá je o niečo menej všeobecná ako notácia rozšírená o kardinalitu. Program vie síce v tejto notácii diagram zobraziť ale konfigurácie, ktoré je možné zaznačiť len v rozšírenej notácii, ponechá v nej.



Obr. 10 Editor diagramu modelu vlastností programu Captain Feature

Ďalšou funkciou programu je vyberanie konkrétnej inštancie (v programe sa nazýva špecializáciou) z editovaného modelu vlastností (Obr. 11). Inštancia je podmnožina množiny všetkých vlastností. Aby bola platná, musí byť vybraná tak, aby boli splnené všetky obmedzenia dané modelom vlastností. Captain Feature ponúka aj možnosť, overenia či je daná špecializácia platná.

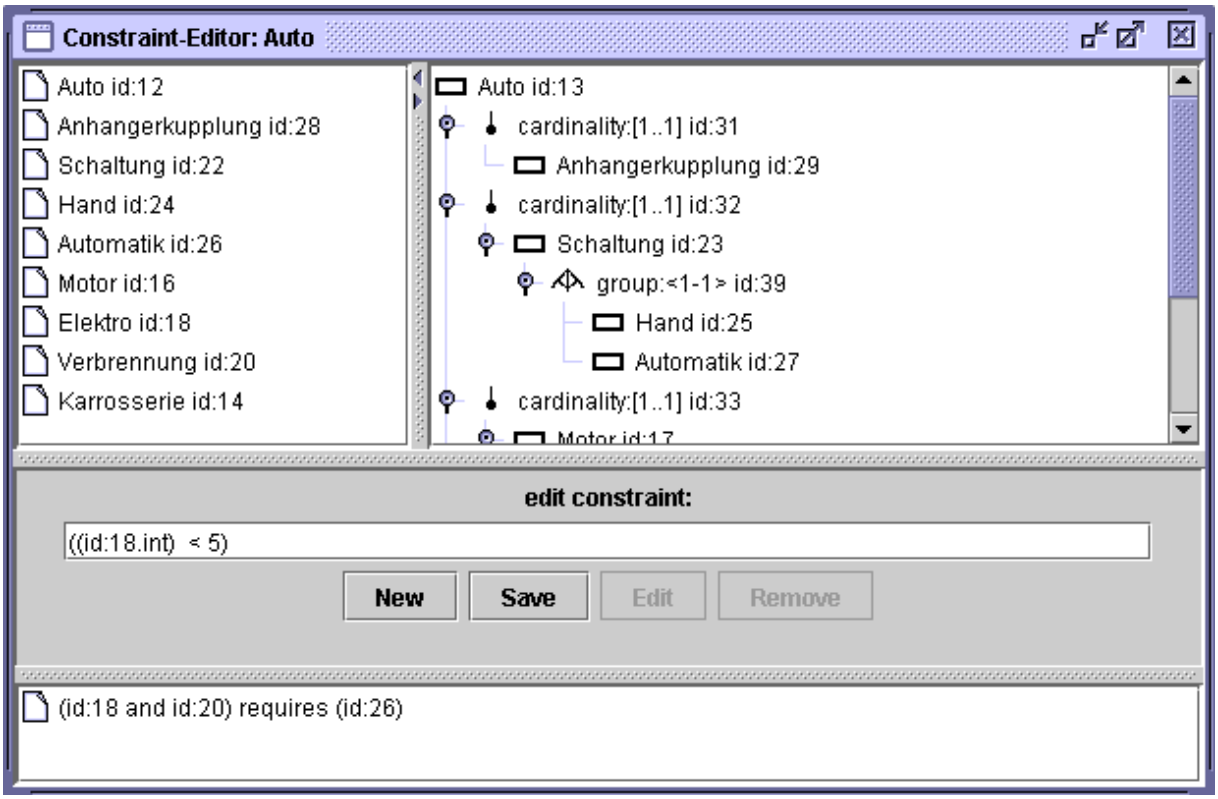


Obr. 11 Editor inštancie programu Captain Feature

Ďalším zaujímavým prvkom v programe je editor obmedzení (Obr. 12). Ten je pomerne nepohodlný a jednoduchý. Svoj účel však spĺňa. Obmedzenia sú zapísané pomocou gramatiky CFCL (ISO/IEC 14977 1996). Celá gramatika je potom popísaná v elektronickej dokumentácii dodanej k programu.

V obmedzeniach na model vlastností môžu okrem typických relácií requires a mutual exclusion vystupovať aj matematické a logické operátory. V rámci obmedzení je možné používať aj kardinality.

Napriek možnosti overovania aj pomerne všeobecnému zápisu obmedzení kladených na model vlastností, je editor obmedzení v programe Captain Feature zatiaľ len veľmi ťažko použiteľný, pretože namiesto mien vlastností sa v zápisoch obmedzení používajú len interné ID označenia vlastností.



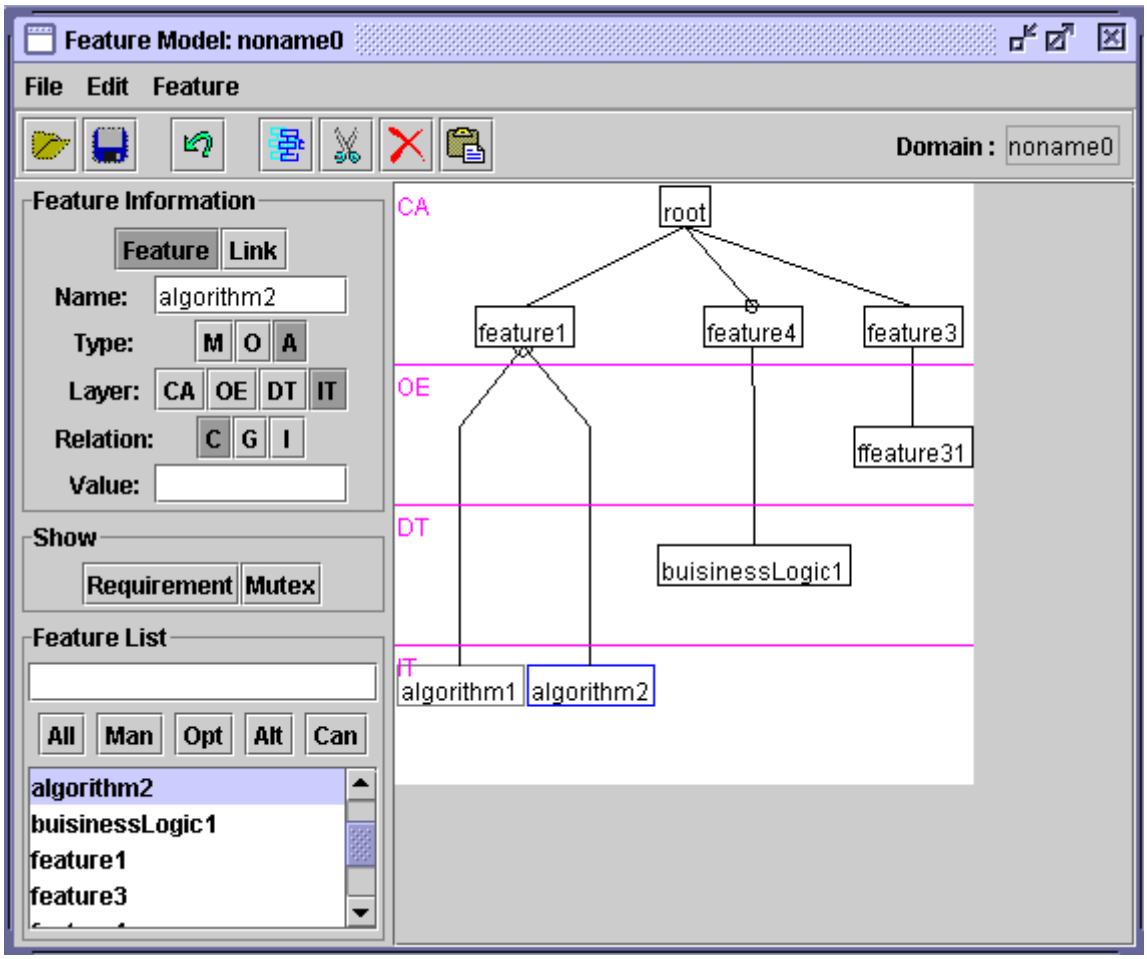
Obr. 12 Editor obmedzení programu Captain Feature

3.2 ASADAL

Program ASADAL bol vytvorený pre podporu metódy FORM spomenutej v časti 2.1. Metóda FORM je len rozšírením a doplnením pôvodnej metódy FODA. Program ASADAL, okrem modelovania vlastností, podporuje aj iné techniky popísané a používané v metóde FORM.

Z tohto dôvodu je samotná podpora modelovania vlastností ako aj tvorby modelu vlastností a príslušných diagramov veľmi obmedzená. Program podporuje len notáciu definovanú v metóde FODA. Tak tiež definuje len dva typy obmedzení presne tak ako FODA.

Zaujímaním prvkom je rozčlenenie vlastností podľa vrstiev tak, ako ho definuje metóda FODA a to na Cieľový operačný systém a prostredie (Operating enviroment layer -OE), Samotné schopnosti aplikácie (Capability layer CA), Technológia známa v doméne (Domain technology - DT) a Implementačné techniky (Implementation technique - IT). Takéto názorné delenie v diagrame modelu prezentuje Obr. 13.



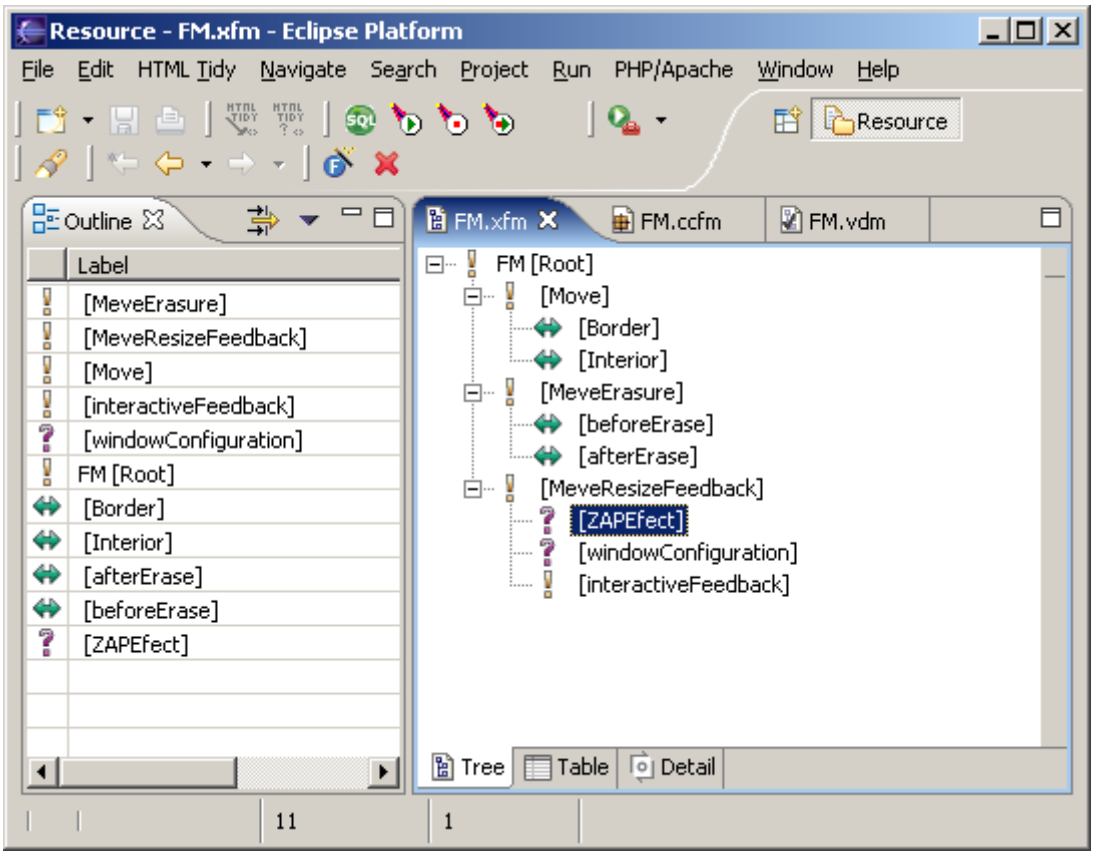
Obr. 13 Editor diagramu modelu vlastností programu ASADAL

3.3 pure::variants

Pure variants (Pure-systems 2005) je komerčný produkt dodávaný ako modul do dnes veľmi populárneho prostredia Eclipse. Zvláštnosťou programu je to, že ponúka viacero pohľadov na model vlastností. Okrem klasického pohľadu na strom, ktorý predstavuje diagram modelu vlastností, ponúka program aj rôzne zoznamy a iné pohľady na model vlastností.

Program umožňuje k vlastnostiam priradiť prakticky akúkoľvek prídavnú informáciu. Užívateľ má totiž možnosť zadefinovať si vlastnú štruktúru prídavných informácií. Program ďalej umožňuje vytvárať aj obmedzenia. Pre obmedzenia síce neexistuje zápis pomocou predikátovej logiky, ale k dispozícii množstvo predefinovaných pravidiel pre obmedzenia, ktoré sa dajú použiť.

Program je vo svojej podstate určený na vytváranie architektonického návrhu pre línie softvérových produktov. Modelovanie vlastností je v programe použité ako podpora pre tento architektonický návrh. Preto je samotné modelovanie vlastností dosť okliešťené. V základnej verzii nie je k dispozícii ani grafický pohľad na model vlastností. Aj podporovaná notácia pre modelovanie vlastností je na úrovni základnej Czarneckého notácie (Czarnecki, Eisenecker 2000).



Obr. 14 Pohľad na jednoduchý model vlastností v module Pure Variants

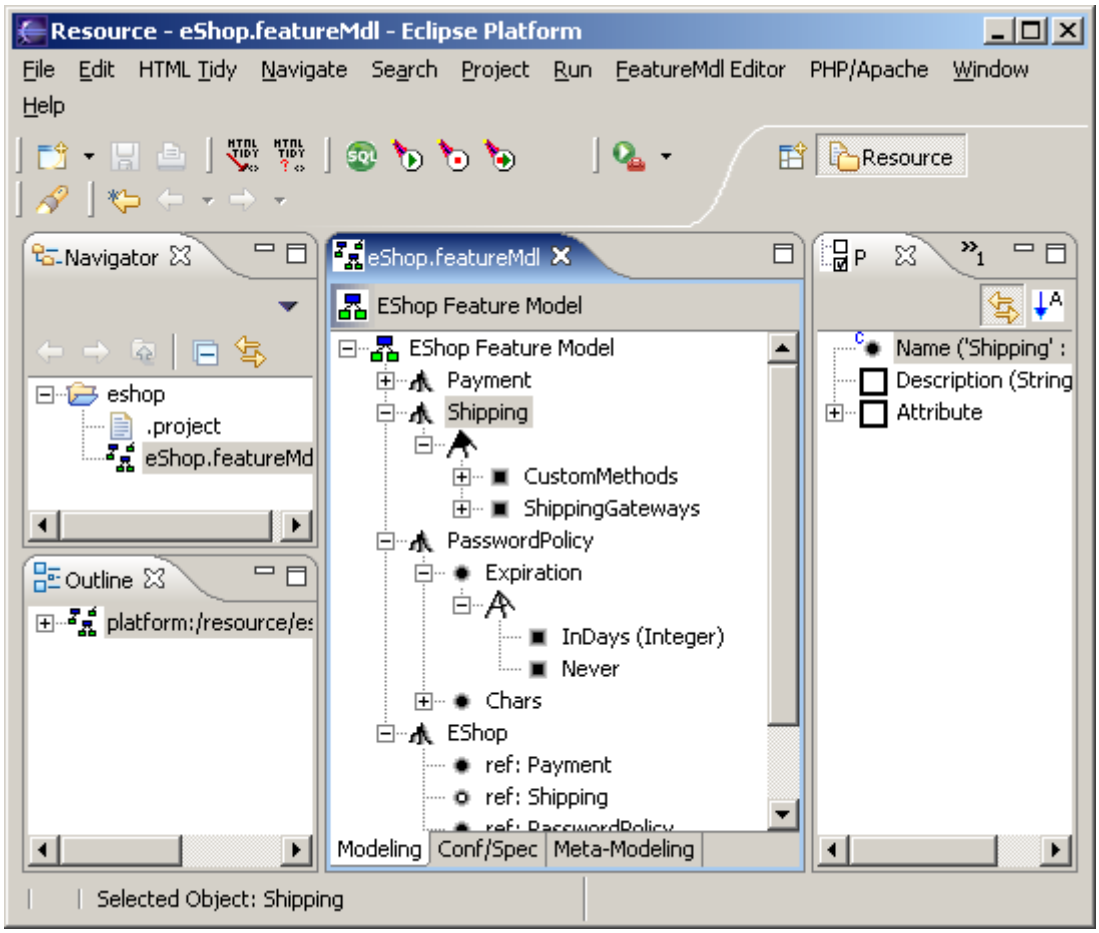
3.4 fmp

Pre prostredie Eclipse je aj posledný analyzovaný nástroj (Czarnecki a i. 2005). Jedná sa takisto o jednoduchý modul. Tak ako to bolo v prípade pure::variants ani tento modul neposkytuje editovanie diagramu modelu vlastností. Namiesto toho zobrazuje len strom (štandardný ovládací prvok), kde sú jednotlivé vlastnosti a aj ich skupiny zobrazené ako uzly. Na rozdiel od modulu pure::variants, tento modul zobrazuje v strome ako uzly aj skupiny vlastností. Takýto strom je zobrazuje Obr. 15 v strede.

Modul samotný je svojou podstatou veľmi podobný programu Captain Feature (časť 3.1). Podobne ako program Captain Feature obsahuje metamodel použitý pri tvorbe programu, ktorý sa bohužiaľ nedá voľne meniť. Na rozdiel od programu Captain Feature je ale v medamodely možné definovať si aspoň vlastnú štruktúru prídavných informácií.. Ďalej modul podporuje základnú a aj rozšírenú Czarneckého notáciu (časť 2.7).

Podobne ako program Captain Feature aj tento modul ponúka editor na vytvorenie jednej inštancie nejakého konceptu s modelu vlastností. Ďalej je k dispozícii aj editor na zapisovanie obmedzení.

Ako prídavok k fmp existuje ďalší modul a to fmp2rsm (Czarnecki, Antkiewicz 2005). Ten umožňuje integráciu modelu vlastností a modelu v jazyku UML. Bohužiaľ tento prídavok zatiaľ pracuje len pre jeden konkrétny CASE nástoj ba dokonca, je presne stanovená aj jeho verzia (Rational Software Modeler verzia 6.0.1).



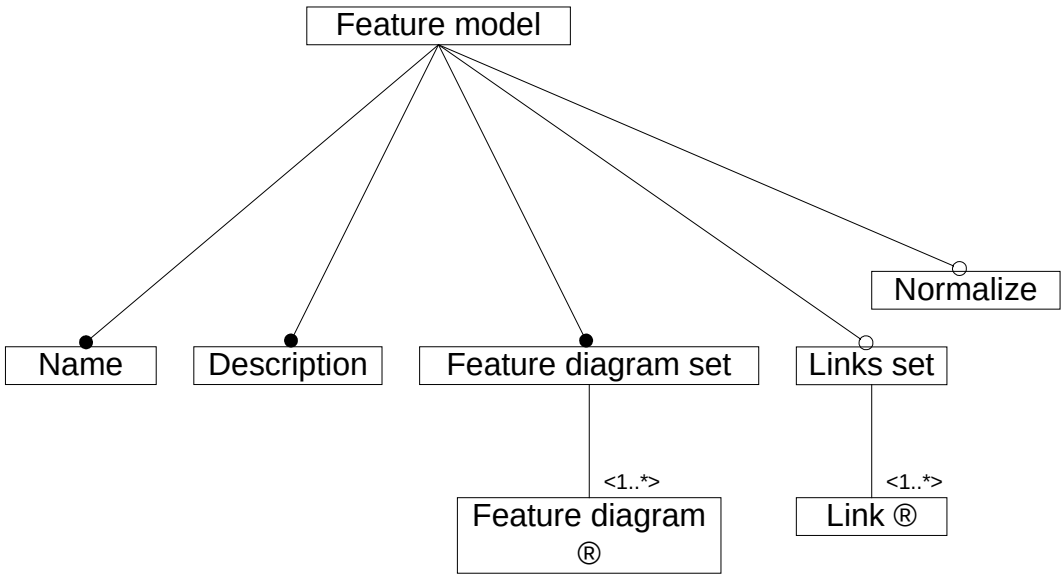
Obr. 15 Feature Modeling Plugin pre prostredie Eclipse

4 Metamodel

V tejto kapitole je prezentovaný metamodel navrhovaný pre pripravovanú aplikáciu. Metamodel je zachytený pomocou modelovania vlastností samotného. Táto časť sa opiera o metamodel, ktorý uverejnil Vranić (Vranić 2004a). Samotná podstata metamodelu sa však líši. V pôvodnej práci bolo pri vytváraní metamodelu cieľom identifikovať rozdielne prístupy jednotlivých notácií. Takto prezentovaný metamodel nepredstavuje metamodel v pravom slova zmysle. Nejedná sa o metamodel modelu vlastností ale o metamodel samotného modelovania. Pri jeho použití je ešte pred začatím modelovania potrebné vybrať jednu jeho konkrétnu inštanciu. Jedna takáto inštancia predstavuje potom už metamodel v pravom slova zmysle.

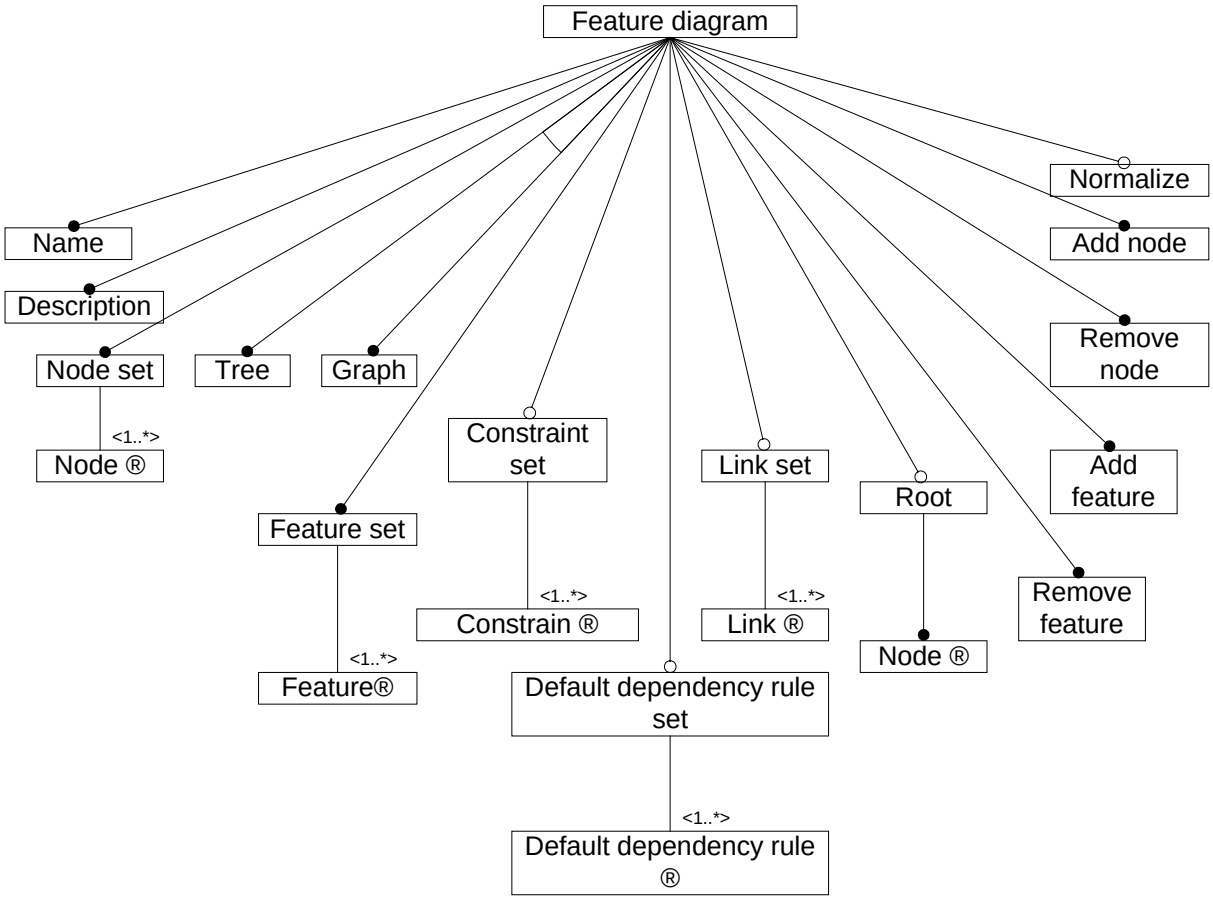
Tento spomenutý fakt samozrejme bráni implementácii prostriedku, ktorý by podporoval čo najviac notácií a bol by v čo najširšej miere konfigurovateľný. Cieľom úprav bude teda odstrániť tento nedostatok. Inými slovami upraviť metamodel tak, aby boli všetky rozhodnutia možné v procese modelovania a nie len na jeho začiatku. To znamená prechod k dynamickému viazaniu vlastností ale aj ich skupín. Napriek týmto úpravám sa v metamodel menil minimálne. V tejto práci je uvedený celý pre to, že je základom pre neskorší návrh nástroja na podporovanie vlastností. Zmeny sa prejavia až pri diagramoch, ktoré opisujú jednotlivé prvky metamodelu (Obr. 19, Obr. 20 v časti 4.2).

4.1 Model vlastností a diagram modelu vlastností



Obr. 16 Diagram metamodelu vlastností pre koncept Model vlastností

Model vlastností reprezentuje model domény (Obr. 16). V najjednoduchšom prípade má model vlastností meno (Name) a popis (Description). Ďalej model vlastností pozostáva z množiny diagramov vlastností (Feature diagram set). Voliteľne môže model obsahovať aj odkazy na iné modelovacie elementy a to aj mimo modelovania vlastností (Links set). V neposlednom rade je možné nad modelom vlastností vykonať operáciu normalizácie.



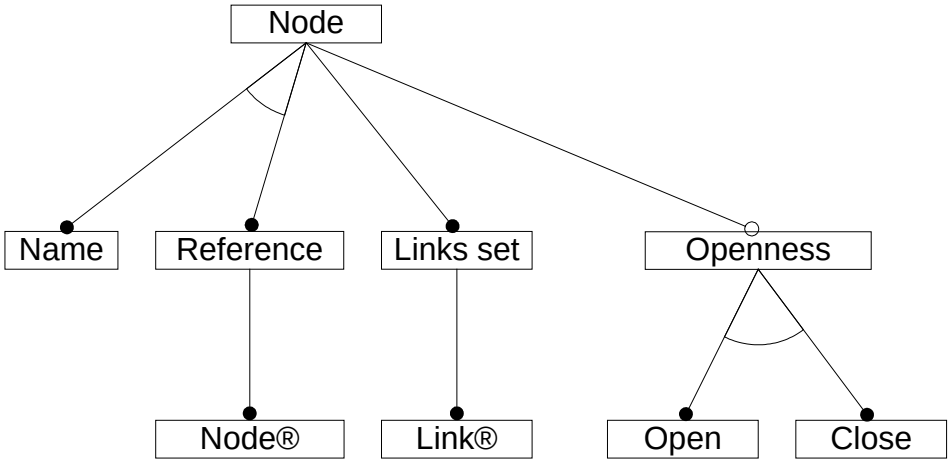
Obr. 17 Diagram metamodelu vlastností pre koncept Diagram vlastností

Diagram modelu vlastností prezentuje grafický popis konceptu (Obr. 17). Diagram modelu vlastností v minimálnom prípade obsahuje meno a popis (Name a Description). Ďalej obsahuje množinu uzlov (Node set), množinu v ňom zobrazených vlastností (Feature set). Ďalej voliteľne obsahuje množinu obmedzení, ktoré platia pri vytváraní jeho inštancie (Constraint set), množinu východzích pravidiel pre vytváranie inštancie modelu (Default dependency rule set) a množinu odkazov na ďalšie modelovacie elementy (Links set). Diagramom modelu vlastností je vo všeobecnosti graf (Graph), ale vo väčšine prípadov sa jedná o strom (Tree).

Diagram modelu vlastností je základný pohľad na samotný model vlastností. Práve cez takýto pohľad sa pridávajú a uberajú vlastnosti v celom modeli a vytvárajú vzťahy medzi nimi (Add feature, Remove feature, Add node, Remove node).

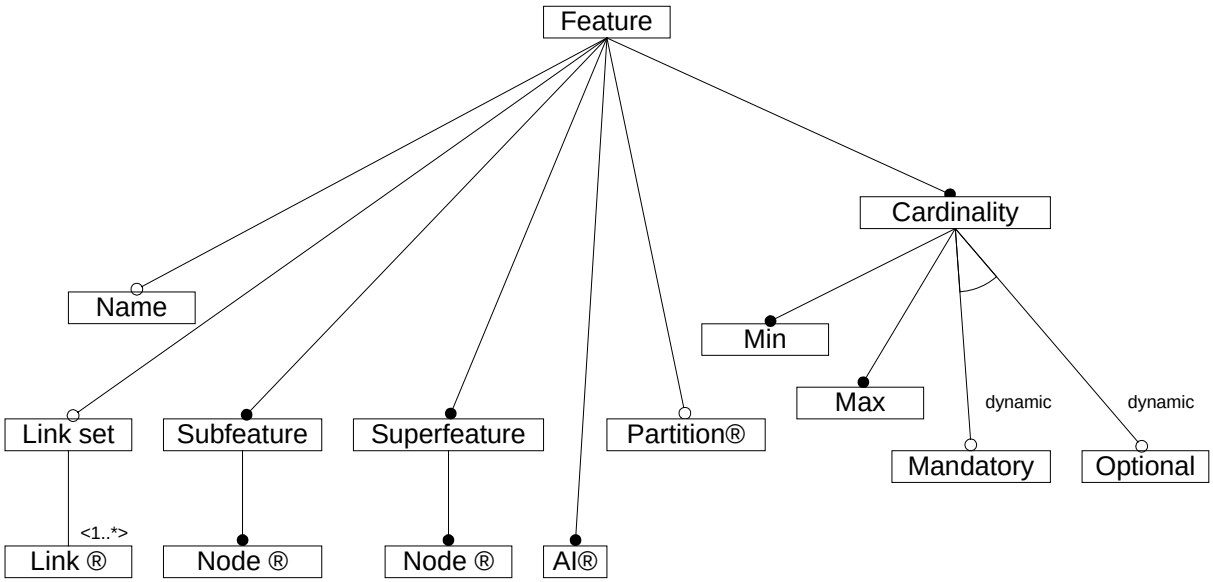
4.2 Základné prvky

Základnými prvkami diagramu modelu vlastností sú uzol (Node), vlastnosť samotná (Feature) a skupina (Partition). Pomocou týchto prvkov sa vytvára model vlastností a príslušné diagramy.



Obr. 18 Diagram metamodelu vlastností pre koncept Uzol

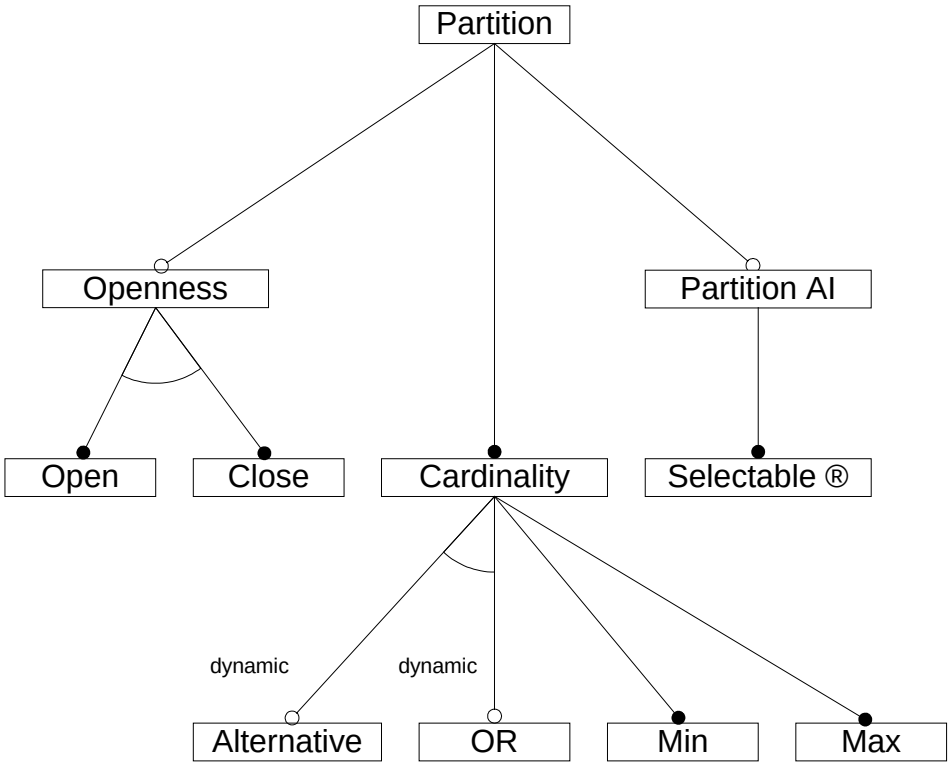
Uzol reprezentuje v metamodeli vlastnosť v jej základnom význame (Obr. 18). Uzol má svoje meno (Name) a referenciu na ďalší koncept (Reference). Taktiež nesie odkazy na iné modelovacie elementy (Links set). Vlastnosť otvorenosť (Openness) udáva, či sú očakávané ďalšie nové vlastnosti (Open), alebo je pridávanie nových vlastností už zakázané (Close).



Obr. 19 Diagram metamodelu vlastností pre koncept Vlastnosť

Podľa už uvedeného chápania významu elementu uzol (Node) je vlastnosť reláciou medzi dvoma uzlami. Vlastnosť (Feature) vyjadruje variabilitu svojej podradenej vlastnosti (Subfeature) rešpektujúc svoju nadradenú vlastnosť (Superfeature). Vzťah medzi uzlami môže byť pomenovaný (Name). Taktiež môže mať vzťah medzi uzlami odkaz na iné modelovacie elementy (Links set).

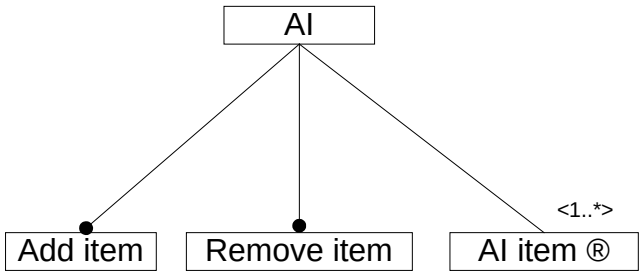
Pôvodný metamodel, ktorý bol prevzatý ako základ pre túto prácu, (Vranić 2004a) neupravoval kombinované používanie kardinality a povinných, či voliteľných vlastností v jednom modeli. Takéto rozhodnutie muselo byť urobené ešte pred začiatkom samotného procesu modelovania. To, či je vlastnosť povinná (Mandatory) alebo voliteľná (Optional), je priamo určené aj kardinalitou vlastnosti (Czarnecki, Helsen a i. 2004). Vlastnosť je povinná, ak je jej minimálna kardinalita väčšia ako 1. Inak je voliteľná. Preto aj vlastnosti Mandatory a Optional v Obr. 19 sú pod vlastnosťou kardinality.



Obr. 20 Diagram metamodelu vlastností pre koncept Skupina

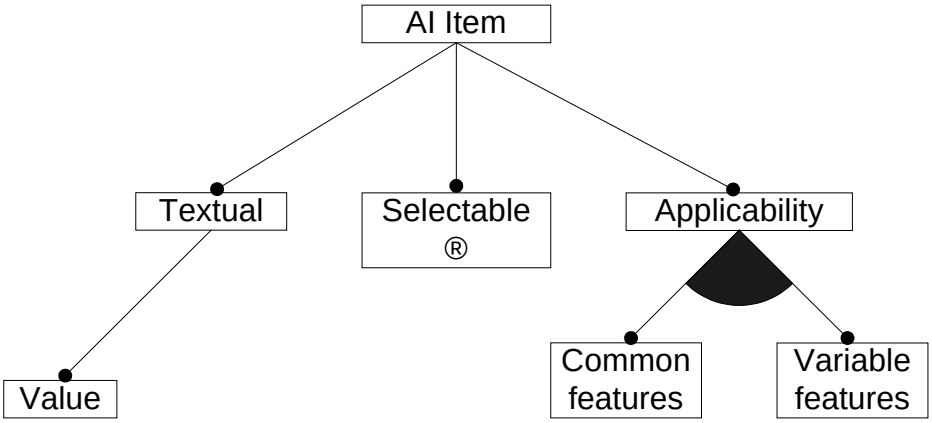
Vlastnosti v jednom bode môžu byť spojené do skupín (Partition, Obr. 20). Jednotlivé skupiny sú navzájom disjunktné. Kardinalita priradená skupine určuje koľko vlastností zo skupiny má byť vybraných do inštancie konceptu. V prípade, že je táto kardinalita 1 - 1, jedná sa o alternatívu (Alternative), tak ako ju definuje samotná FODA (časť 2.1). V prípade, že kardinalita je 1 - n, jedná sa o disjunktívne vlastnosti.

4.3 Prídavné informácie

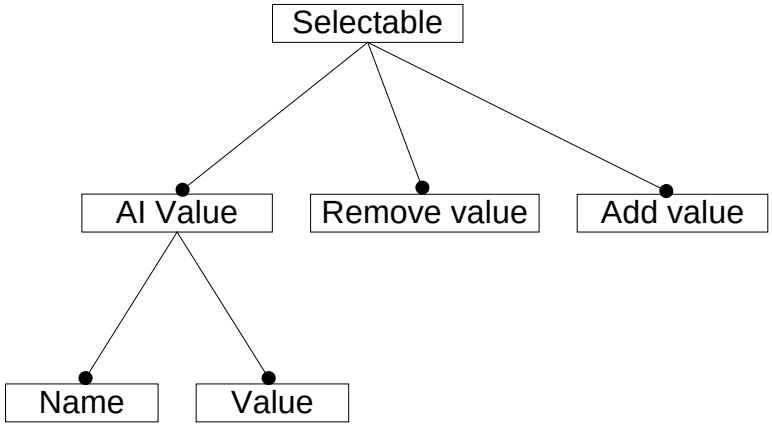


Obr. 21 Diagram metamodelu vlastností pre koncept Prídavná informácia

Prídavná informácia (AI) je v tomto metamodeli definovaná čo najvšeobecnejšie (Obr. 21). Rôzne prístupy a to aj do budúcnosti budú pravdepodobne definovať rôzne prídavné informácie. Celá prídavná informácia je zložená z jednotlivých položiek (AI item).

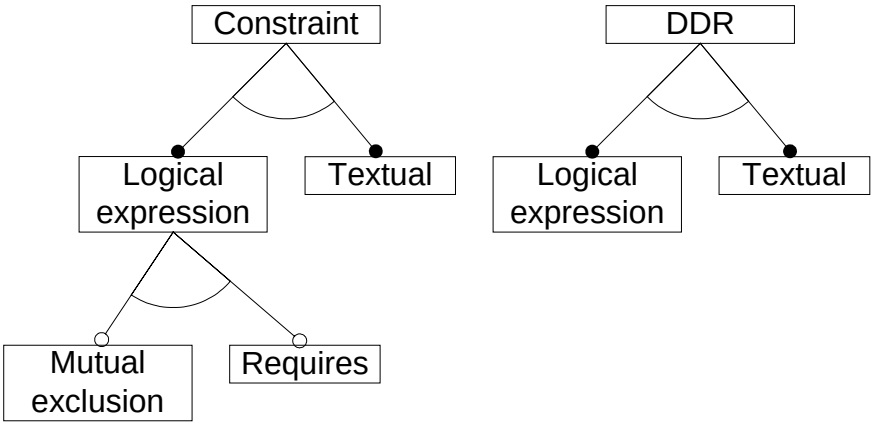


Obr. 22 Diagram metamodelu vlastností pre koncept Položka prídavnej informácie



Obr. 23 Diagram metamodelu vlastností pre koncept Položka prídavnej informácie s dopredu definovanou množinou prípustných hodnôt

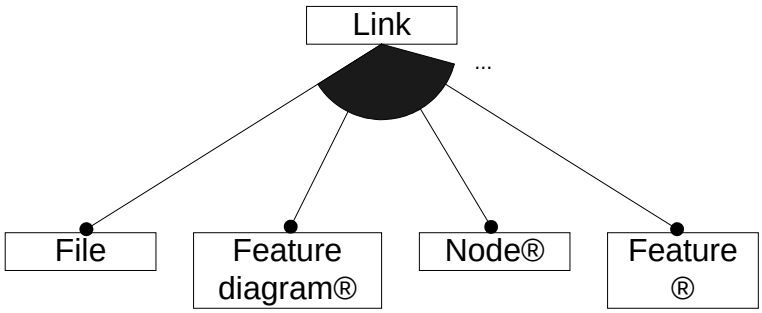
Položka prídavnej informácie môže byť buď textová (Textual), alebo sa môže vyberať z dopredu definovanej množiny hodnôt (Selectable, Obr. 23).



Obr. 24 Diagramy metamodelu vlastností pre koncepty Obmedzenia a Pravidlá pre parametre

Obmedzenie je vo všeobecnosti vyjadrené vo forme logickej formule (Logical expression) alebo len v podobe voľného testu. Špeciálnym prípadom obmedzenia môže byť vyjadrenie vzájomného vylúčenia sa dvoch vlastností (Mutula exclusion) alebo jeho opaku (Requires). Pravidlá pre

parametre (DDR) sú vo všeobecnosti vyjadrené tiež buď to vo forme logického výrazu alebo len ako voľný text.



Obr. 25 Diagram metamodelu vlastností pre koncept Odkaz

Odkaz (Link, Obr. 25) umožňuje, aby model vlastností, alebo jeho časť odkazovala na iný modelovací element, a to buď v modeli samotnom alebo aj na iný modelovaný element. Preto je aj tento koncept otvorený.

5 Východiská tvorby nástroja na podporu modelovania vlastností

Ako vidno v časti 2.9 dostupné nástroje na podporu modelovania vlastností sú jednoznačne orientované na podporu konkrétnej notácie alebo prístupu k modelovaniu vlastností. Žiadna z aplikácií neponúka také možnosti nastavenia, aby mohla podporovať metamodel navrhnutý v kapitole 4. Kvôli týmto nedostatkom spomínaných dostupných nástrojov bolo ako riešenie zvolený návrh a implementovanie samostatne stojacej aplikácie, ktorá bude podporovať aspoň v kapitole 4 navrhovaný metamodel. Táto kapitola predstaví východiská, s ktorých bude možné formulovať požiadavky na systém a prezentovať rozhodnutia, ktoré bolo nutné urobiť, kým bolo možné pristúpiť k návrhu a implementácii systému.

5.1 Východiská

Pred tým ako budú špecifikované požiadavky na navrhovaný systém, je nutné vybrať s analyzovaných prístupov a prostriedkov na modelovanie vlastností tie vlastnosti, ktoré sú prínosom pre túto prácu. Cieľom je teda vybrať tie prvky, ktoré umožňujú použiť ten ktorí prístup alebo prostriedok na inú úlohu, okrem tej na ktorú bol určený. Naopak je cieľom potrieť tie vlastnosti, ktoré sú charakteristické pre ten ktorý prístup a dali by sa zovšeobecniť. Potom je samozrejme cieľom identifikovať takéto zovšeobecnenie.

5.1.1 Vizualizácia modelu

Systémy ako Caprain Feature (časť 3.1) alebo ASADAL (časť 3.2) sa príliš orientovali na tvorbu a vizualizáciu diagramu modelu vlastností. Naopak systémy ako pure::variants (časť 3.3) alebo FM plugin pre prostredie Eclipse (časť 3.4) vôbec neumožňovali tvorbu diagramu a zameriavali sa len na tvorbu modelu vlastností. V tejto práci navrhovaný nástroj sa bude prikláňať viac k posledne uvedenej skupine nástrojov. Na rozdiel od nich bude možné vytvárať aj diagramy modelu vlastností a vizualizovať tak model vlastností aj s príslušnými informáciami

5.1.2 Kardinalita

Prvé prístupy k modelovaniu (samotná FODA opísaná v časti 2.1) vlastností nepoužívali pri vyjadrovaní početnosti vlastností kardinalitu. Namiesto toho boli používané voliteľné alebo povinné vlastnosti. Okrem toho definovali takzvané alternatívne vlastnosti. Tieto prístupy zavádzali alternatívu v zmysle výberu najviac jednej z viacerých vlastností. O tom, či v inštancii modelu musela byť aspoň jedna vlastnosť so skupiny alternatívnych vlastností alebo nie, rozhodovalo to, či boli vlastnosti v takejto skupine povinné alebo voliteľné. Miernym rozšírením bolo zavedenie OR skupiny (Czarnecki, Eisenecker 2000). Tá rozširovala možnosť alternatívnych vlastností o výber z viacerých z nich.

Najvšeobecnejší prostriedok na vyjadrenie, toho či je vlastnosť povinná alebo voliteľná, je ale kardinalita vlastnosti a taktiež kardinalita skupiny vlastností. Kardinalita vlastnosti udáva, koľkokrát môže byť daná vlastnosť zahrnutá do inštancie konceptu. Ak je teda minimálna kardinalite väčšia ako nula jedná sa napríklad o povinnú vlastnosť a pod. Kardinalita skupiny vlastností naopak udáva, koľko zo zoskupených vlastností môže byť zahrnutých do inštanice konceptu. Mapovanie medzi touto notáciou a jej predchodcami uvádza Obr. 9. Pre implementáciu CASE prostriedku, ktorý má podporiť čo najviac notácií je prirodzeným výberom práve použitie kardinalít.

5.1.3 Prídavné informácie

V zásade slúžia prídavné informácie na priradenie ľubovoľnej používateľskej informácie ľubovoľnému elementu modelu. Tieto informácie sú potom využívané pri konfigurácii iných modelov a samotného kódu systému. Striktne vzaté aj kardinalita je len prídavná informácia ku vlastnosti. Navrhovaný CASE nástroj k nej ale pristupuje osobitne. Je tomu tak preto, že kardinalita vyjadruje základné informácie, o elemente v modely vlastností.

Navrhovaný CASE nástroj umožňuje práve definovaním vhodnej sady prídavných informácií obmedzené editovanie metamodelu. Systém pozná tieto typy prídavných informácií

- Prídavná informácia nesúca boolovskú hodnotu
- Prídavná informácia nesúca číselnú hodnotu
- Prídavná informácia nesúca reťazovú hodnotu.
- Prídavná informácia nesúca jednu s predom zadefinovaných hodnôt.

Hodnoty posledne menovanej prídavnej informácie sú usporiadané. Záleží na používateľovi, či si nad množinou týchto hodnôt definuje usporiadanie alebo nie. Aby sme mohli podporiť notácie, ktoré zavádzanú rôzne prídavné informácie pre vlastnosti a dokonca aj pre ich skupiny a aby sme dokázali zachytiť ich rozmanitosť, musí prezentovaný CASE nástroj implementovať práve používateľom definovanú štruktúru informácií.

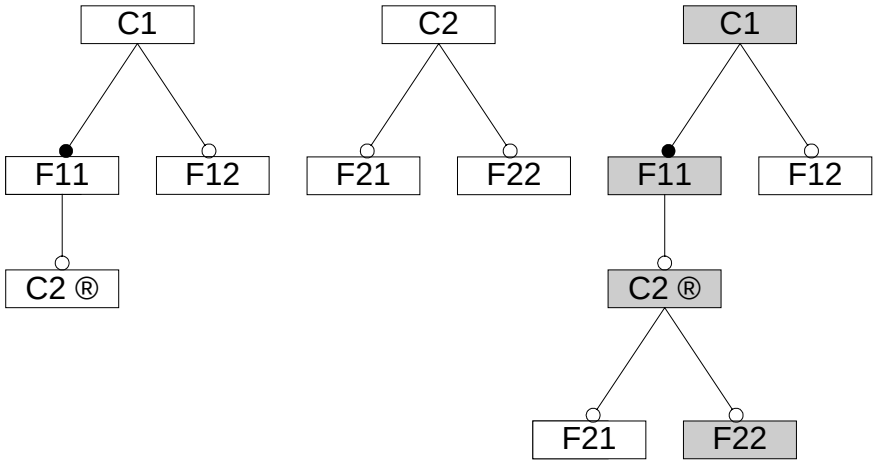
5.1.4 Vyhodnotenie obmedzení

Cieľom tejto šasti bude vytvoriť formailzmus obmedzení tak, aby bolo možné zohľadniť dva sledované ciele:

1. Overovanie obmedzení referencovaného konceptu a odkazovanie sa v obmedzeniach na referencovaný koncept
2. Overovanie obmedzení vzhľadom na čas viazania inštancie konceptu a časy viazania jednotlivých vlastností konceptu.

Za základ budeme brať obmedzenia zapísané v predikátovej logike tak, ako to bolo uvedené v časti 2.3.

Venujme sa najprv prvému bodu a uvažujme koncepty C1 a C2, kde C1 referencuje C2 a inštanciu C1 podľa Obr. 26. Na ľavo a v strede sú diagramy modelu vlastností pre koncepty C1 a C2 a napravo je naznačená celá inštancia konceptu C1 po rozbalení referencie C2.



Obr. 26 Príklad referencivania konceptu a inštancie

Uvažujme ďalej takúto sadu obmedzení pre C1

Cs1: F12 ⇒ C2

A sadu obmedzení pre C2

Cs2: F22 ⇒ ¬F21

Ako je vidieť inštancia z Obr. 26 takéto obmedzenia spĺňa. Cieľom tejto časti je jasne definovať, ktoré z týchto pravidiel a v akom priestore sa vyhodnotia. Pre obmedzenie Cs1 je to jednoduché, lebo to sa triviálne vyhodnotí v priestore mien konceptu C1. Pri vytváraní takejto inštancie je potrebné vytvoriť aj obmedzenie Cs2 to sa vyhodnotí v priestore mien referencovanej vlastnosti C2 ®.

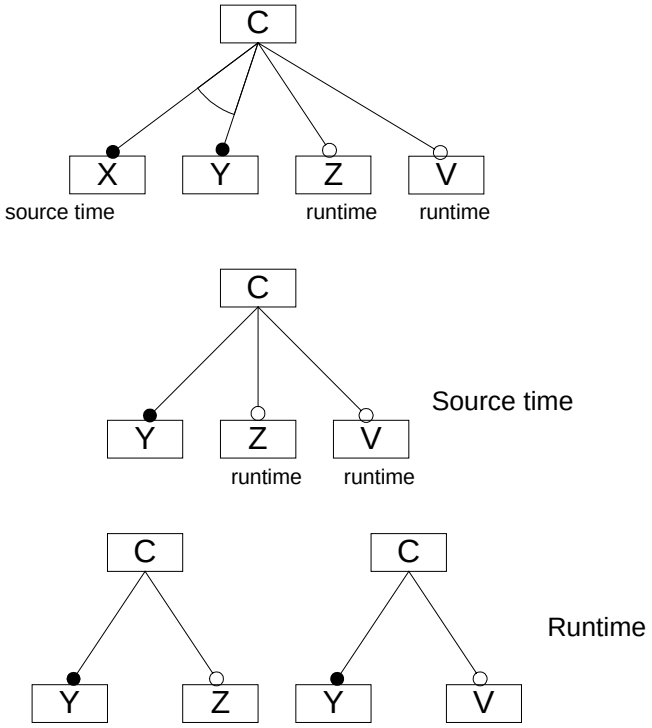
Posledným typom tu zavedených obmedzení je obmedzenie na celý koncept, vrátane ďalších konceptov, ktoré referencuje. Príkladom je obmedzenie Cs3 na koncept C1

Cs3: F12 ⇒ F22

Napriek tomu, že je vlastnosť F22 v modeli jedinečná, presnejšie má jedinečné meno, nie je v tomto prípade odkazovanie sa na ňu len menom vhodné. Dôvodov je hneď niekoľko. Pri vytváraní konceptu nie sú priamo viditeľné vlastnosti referencovaných konceptov. Potom nie je jasné, odkiaľ vlastne pochádza vlastnosť F22. Druhým dôvodom je, že vnesenie novej vlastnosti s rovnakým menom do nejakého referencovaného konceptu, môže spôsobiť chybu, ktorú pri pohľade len na takto referencovaný koncept nie je vidno. Preto pre takýto zápis budeme pokladať za prípustné len plné odvolanie sa na vlastnosť F22 cez oddelenie bodkou, tak ako je to opísané v časti 2.3. Správny zápis obmedzenia Cs3 je teda nasledovný.

Cs3 F12 ⇒ C1 . F11 . C2 . F22

Druhým problémom je vyhodnotenie správnosti inštancie vzhľadom na čas viazania inštancie konceptu a časy viazania jednotlivých jeho vlastností Uvažuje príklad ako na Obr. 27. Problémom je, či sa dá považovať vlastnosť za zaradenú do konceptu, ak je doba jej viazania skoršia, resp. neskoršia, ako doba viazanie inštancie konceptu.



Obr. 27 Príklad vytvárania inštancie v rôznych dobách viazania

Najprv uvažujme inštanciu konceptu vytváranú v čase tvorby zdrojového kódu (Source time) a obmedzenie $Z \Rightarrow X$. Ak by sme Z pokladali za neprítomné v Source time, potom je obmedzenie splnené. Ak však následne v čase vykonávania programu (Runtime) vytvoríme inštanciu, ktorá bude obsahovať vlastnosť Z, vznikne problém. To znamená, že vlastnosť s neskoršou dobou viazanie, ako je doba vytvárania inštancie nie je možné pokladať za neprítomnú v inštancii, lebo vytváranie inštancie pre ďalšie časy môže viesť ku chybovému stavu. Tu ho reprezentuje prípad, kedy sa za behu programu môže vytvoriť inštancia C s vlastnosťami Z a Y.

Uvažujme ďalej inštanciu konceptu vytváranú v čase tvorby Source time a obmedzenie $\neg Z \Rightarrow X$. Ak by sme Z pokladali za prítomné v čase Source Time, dostali by sme sa, podobne ako tomu bolo v prvom prípade, ku chybe. V tomto prípade by inštancia v Source Time nebola platná a nikdy by sme sa tiež nedostali k platnej inštancii {C, Y, W}. To znamená, že vlastnosť s neskoršou dobou viazanie ako je doba vytvárania inštancie, nie je možné pokladať ani za prítomnú v inštancii, lebo vytváranie inštancie pre ďalšie časy opäť môže viesť ku chybovému stavu Vranić 2005.

Jediným spôsobom, ako takýto stav vyriešiť, je preniesť obmedzenie do neskoršieho času vytvárania inštancie. Pre tvorcu CASE nástroje to znamená informovať používateľa, že pre danú dobu viazania nie je možné takéto obmedzenie vyhodnotiť. Pre používateľa resp. tvorcu modelu takáto informácia znamená, že musí toto obmedzenie preniesť do vytvárania inštancie v neskoršom čase. Ak je inštanciu možné takto formálne overiť aj v neskoršom čase, nepredstavuje to ďalšiu prácu. V našom prípade ale bol ďalší čas viazanie Runtime a nie je predpoklad, že v tomto čase je možné inštanciu overiť. V tomto prípade bude musieť používateľ modelu vykonať dodatočné zásahy, napríklad do zdrojového kódu aplikácie, a takto do nej preniesť nevyhodnotené obmedzenia. Ak uvažujeme prenesené obmedzenie $Z \Rightarrow X$, tak takýto zásah znamená pridať do programu ušetrenie tak, aby sa zabránilo vytvoreniu inštancie {C, Z ,Y}.

Vytváranie inštancie a jej vyhodnotenie tvorí základ modelovania vlastností. Používanie referencií na iný koncept v modeli vlastností tiež poznajú všetky prístupy. Preto je logické podporiť overenie inštancie aj v takomto prípade.

5.2 Požiadavky

Základné požiadavky na systém sa dajú zhrnúť do týchto bodov:

- Oddelenie diagramu modelu vlastností od samotného modelu. Systém má podporiť tvorbu modelu vlastností a nie len tvorbu diagramov modelu.
- Obmedzená editácia metamodelu na úrovni prídavných informácií. Nástroj bude umožňovať priradiť k ľubovoľnému elementu modelu ľubovoľný počet prídavných informácií. Prídavné informácie bude možné vybrať len z dopredu definovaných typov. Prídavné informácie bude možné zobraziť v diagrame modelu vlastností ako aj v prehľadoch modelu.
- Podpora vytvárania inštancií a jej overenie podľa pripojených obmedzení. Takto umožniť použitie modelu vlastností na konfiguráciu iných modelov a vytváraného softvétového produktu.
- Podpora pre vytváranie obmedzení na model v predikátovej logike prvého rádu. Obmedzenia organizovať do sád obmedzení. Ďalej podpora pre vyhodnotenie inštancie modelu (resp. jedného konceptu) vzhľadom na takéto obmedzenia.
- Podpora pre vyhodnotenie obmedzení vzhľadom na čas viazania vlastností a čas viazania inštancie. Ako čas viazania inštancie využiť jednu resp. viacero vhodne definovaných prídavných informácií.
- Zavedenie kardinality namiesto povinných a voliteľných vlastností a ohodnotenie všetkých prvkov modelu kardinalitami.
- Zverejnenie modelu pre používanie v ostatných aplikáciách prostredníctvom OLE automatizácie.

5.3 Výber technológie a prostredia pre vývoj aplikácie

V dnešnej dobe existuje veľa rôznych programovacích jazykov a podporných knižníc a prostredí, ktoré podporujú vývoj aplikácií v tom ktorom jazyku. Preto je dnes veľmi ťažké zhodnotiť úplne presne vlastnosti všetkých prostredí a vybrať to najlepšie pre tento problém. Na druhej strane je treba povedať, že väčšina profesionálnych vývojových prostredí je dodávaných komerčne a preto je ich dostupnosť hlavný problém. Preto je zbytočné rozoberať vývojové prostredia, ktoré nie je možné legálne použiť pri riešení tohto problému. Náš výber je treba obmedziť na tie prostredia, ktoré sú legálne k dispozícii. Okrem toho sú hlavné kritériá na výber prostredia to, ako rýchlo a efektívne sa v ňom bude dať implementovať navrhované riešenie a to, aké budú možnosti jeho využitia. Nie je hanbou priznať, že aj miera skúsenosti s tým ktorým vývojovým prostredím je jedným s najzávažnejších kritérií pre výber.

5.3.1 Microsoft Foundation Class (MFC)

MFC je knižnica, ktorú vyvinula firma Microsoft a dodáva ju k svojmu prostrediu C++ (Microsoft Corporation 1998), aby podporila vývoj aplikácií pre svoj operačný systém Windows. Jej vek v rátane prvých verzií už prevršuje úctyhodných 10 rokov. Táto knižnica a knižnice jej podobné sa stali veľmi populárne pre programovanie v prostredí Windows pretože používať priamo aplikačné rozhranie operačného systému Windows je veľmi zložité. Ak vytvárame program pre operační systém Windows je viac ako prirodzené zvoliť si jednu s takýchto knižníc. Napriek jej nespornému úspechu už knižnica zastaráva a a je už pomerne dlho vytláčaná ďalším riešením od tej istej spoločnosti a to frameworkom .NET.

Technológia MFC bola na realizáciu projektu vybraná hneť s niekoľkých dôvodov. Prvým je, že ponúka jednoduché a priamočiare vytvorenie aplikácie podľa vzoru dokument pohľad. Technológia MFC nevyžaduje implementovanie dodatočných medzivrstiev. Tento cieľ je pre realizovanú aplikáciu veľmi dôležitý, pretože bude ponúkať pohľad na pomerne jednoduchý model viacerými spôsobmi. Práve MFC vyžaduje na realizáciu jedného pohľadu majmenšie množstvo vykonanej práce spomedzi analyzovaných technológií. Pre samotnú reprezentáciu údajov v pracovnej pamäti ponúka MFC dostatočné možnosti v podobe kontajnerových tried, ktoré sú už v MFC implementované.

Poslednou výhodou je pomerne jednoduchá implementácia OLE automatizácie. Táto technológia umožní aby bol model vlastností prístupný aj iným Windows aplikáciám, ktoré podporujú túto technológiu. V tomto smere ponúka najjednoduchšie riešenie technológia .NET. Tá umožňuje bez zvláštnej práce pristupovať pomocou mechanizmu OLE automatizácie ku každému objektu. Realizovaná aplikácia však potrebuje s dôvodu zabezpečenia konzistentnosti dát, obmedzovať prístup k jednotlivým triedam. Preto by bolo aj v prípade .NET potrebné implementovať pre OLE automatizáciu zvláštnu medzivrstvu.

Pre tieto výhody bola práce technológia MFC vybraná na realizáciu nástroja na podporu modelovanie vlastností.

5.3.2 Modul pre prostredie Eclipse

Najviac zvažovanou alternatívou, okrem zvolenej implementácie v MFC, bolo vytvorenie vlastného modulu pre aplikáciu Eclipse (Eclipse contributors and others 2005). Prostredie Eclipse ponúka dva základné moduly, ktoré by boli nápomocné pri implementácii takto špecifikovanej aplikácie. Prvým s nich je EMF (Eclipse Modeling Framework). Tento modul obsahuje základné funkcie potrebné na vytvorenie, manipuláciu a ukladanie modelu. Druhým modulom je GEF (Graphical Editing Framework). Tento veľmi komplexný modul realizuje podporu pre grafické editovanie diagramov resp. iných grafických reprezentácií. Vhodnou spoluprácou týchto dvoch modulov (Aniszczyk 2005) by bolo možné docieliť zobrazovanie diagramu modelu vlastností ako aj zobrazovanie modelu samotného.

Veľkou výhodou takéhoto riešenia je integrácia modelu vlastností s inými modelmi. Veľmi dobrú demonštráciu ponúka Mapping Features to UML 2.0 Models Plug-in (Czarnecki, Antkiewicz 2005). Prostredie Eclipse má totiž v sebe zabudovanú možnosť, ako vyriešiť synchronizáciu viacerých prídavných modulov. Použitie tejto možnosti je ale obmedzené. Spojenie takýchto dvoch prídavných modulov treba implementovať minimálne do jedného s nich. To by obmedzilo spoluprácu len na počas vývoja známe prídavné moduly.

Okrem toho sú nevýhodami pomerne komplikované používanie najmä modulu GEF. GEF totiž nezobrazuje priamo dáta modelu ale používa pomocnú hierarchiu objektov. Tá umožňuje veľkú variabilitu grafického editovania ako je kotvenie čiar a objektov, mriežka a pod. a samozrejme je navrhovaná na to, aby čo najlepšie realizovala práve tieto úlohy. Preto je pomerne zložitá synchronizácia medzi samotným modelom na pozadí a touto pomocnou vrstvou objektov. Obtiažne je najmä premietnutie zmeny modelu do jeho grafickej reprezentácie.

6 Vývoj nástroja

V tejto časti bude opísaný postup vývoja nástroja na podporu modelovania vlastností. Bude tu popísané rozdelenie vývoja systému na jednotlivé etapy a potom návrh, ako ich realizovať.

6.1 Etapy vývoja aplikácie

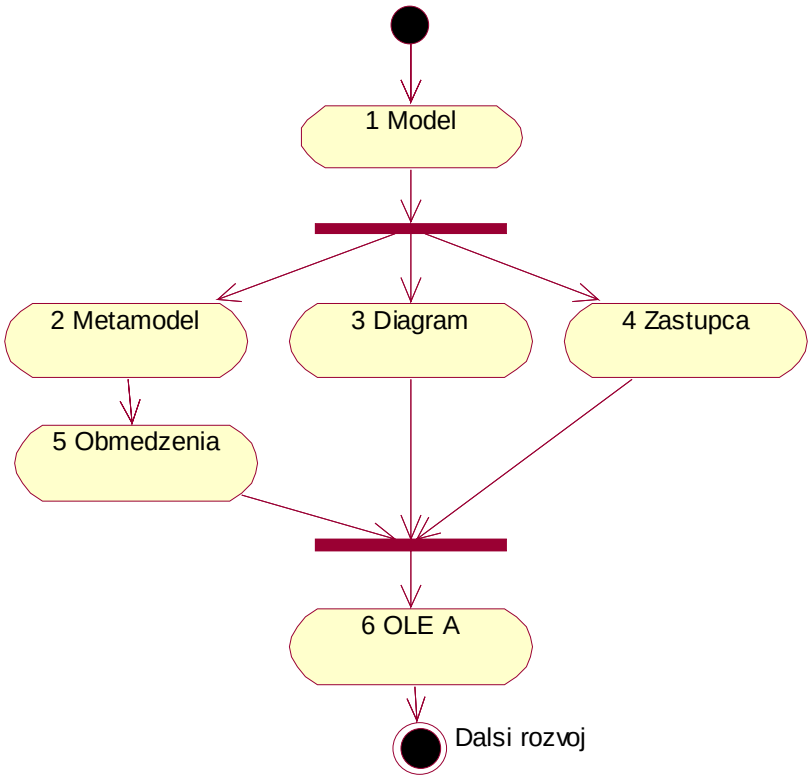
Pre vývoj aplikácie bol zvolený inkrementálny model. Je to preto, že požiadavky na aplikáciu sa dajú jednoznačne rozdeliť do viacerých inkrementov (etáp). A vo väčšine prípadov platí, že nasledujúca etapa je závislá aspoň na čiastočnom dokončení predošlej. Tvorba aplokácie bola teda rozdelená na tieto etapy.

1. Vytvorenie reprezentácie základného modelu. Cieľom tejto základnej etapy je navrhnúť a implementovať reprezentáciu modelu vlastností v najzákladnejšom tvare. Bude realizovaná hierarchická štruktúra prvkov modelu, kde každý prvok okrem koreňa modelu bude mať svoj nadradený prvok v hierarchii a každý prvok môže mať viacero podradených prvkov v hierarchii.
2. Vytvorenie reprezentácie metamodelu. Cieľol tejto etapy je vytvorenie reprezentácie pre prídavné informácie. Je možné v systéme zadefinovať ľubovoľné množstvo typov prídavných informácií a hodnotu konkrétneho typu vybranej prídavnej informácie priradiť ľubovoľnému elementu zo základného modelu.

   Ako prídavná informácia pre prvky modelu bude realizovaná aj informácia o tom, či je prvok (vlastnosť alebo skupina vlastností) zahrnutá v nejakej inštancii.
3. Vytvorenie reprezentácie diagramu modelu. Cieľom tejto etapy je vytvorenie reprezentácie, ktorá umožní udržiavať informácie o tom, ako je základný model zobrazovaný v jednotlivých diagramoch.
4. Vytvorenie reprezentácie zástupcu modelu. Kvôli podpore vytvárania inštancií je nutné vytvoriť reprezentáciu, ktorá umožní vložiť „linku“ jednej časti modelu do inej časti. Po tejto etape budeme mať možnosť vytvoriť v modeli prvok, ktorý bude v niektorých prípadoch zastupovať svoj originál resp. vzor ale v iných prípadoch bude vystupovať ako samostatný element. (V skutočnosti existuje jediný prípad, kedy takýto prvok vystupuje ako samostatný element a to je pri zisťovaní, či patrí do nejakej z inštancií modelu. Inak takýto prvok plne zastupuje svoj originál.)

   Časťou tejto etapy je aj vytvorenie podpory pre pridávanie referencovaných vlastností. Tie sú realizované ako zástupcovia.
5. Vytvorenie mechanizmu vyhodnotenia obmedzení. Táto etapa zahŕňa tvorbu mechanizmu na vyhodnotenie obmedzení resp. zaradenie nejakého už hotového mechanizmu na vyhodnotenie logických výrazov do aplikácie.
6. Vytvorenie reprezentácie tried pre OLE Automatizáciu a implementácie podpory OLE Automatizácie. Napriek tomu, že je prirodzené, keď je hierarchia dokumentu publikovaná prostredníctvom mechanizmu OLE automatizácie, je totožná s hierarchiou, v akej sú dáta reprezentovaná vo vnútri aplikácie, nie je to vždy najvýhodnejšie riešenie. V tomto prípade budeme musieť pred používateľom modelu prostredníctvom OLE automatizácie skryť niektoré informácie, aby nevhodným zásahom nespôsobil nekonzistentnosť reprezentácie modelu. Okrem toho OLE automatizácia pozná takzvané kolekcie, ktoré umožňujú iterovať po jednotlivých prvkoch dokumentu a ich implementácia vyžaduje vytvárať dodatočné triedy pre jednotlivé typy takýchto kolekcií.

Nie všetky etapy vyžadujú pre svoju realizáciu dokončenie predošlej etapy. Jednotlivé závislosti zobrazuje Obr. 28

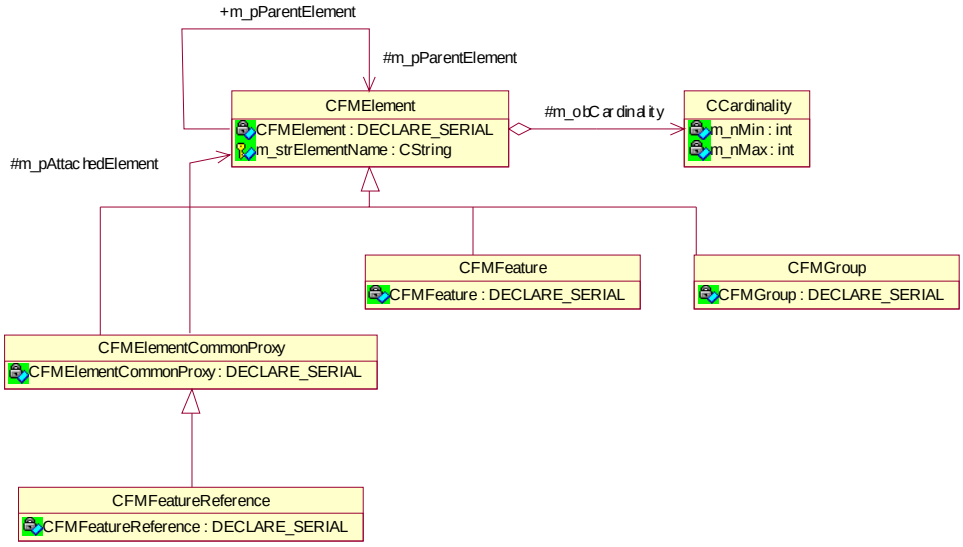


Obr. 28 Etapy vývoja aplikácie

6.2 Návrh časti Model

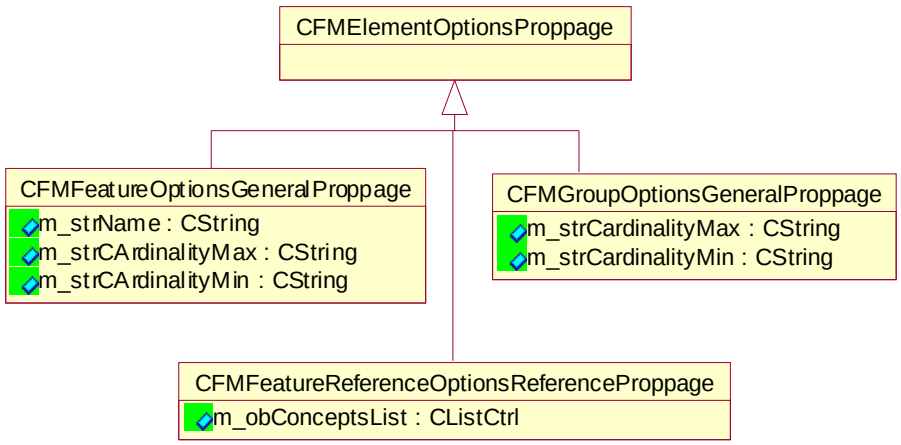
Subsystém Model je zodpovedný za základné udržiavanie reprezentácie modelu vlastností a manipuláciu s ním. Jeho úlohou nie je udržiavať prídavné informácie.

Navrhovaná hierarchia časť Model je na Obr. 29. Základom je trieda `CFMElement`. Tá predstavuje abstraktný základ pre ostatné prvky modelu. Je zodpovedná za udržiavanie základných vlastností prvkov modelu ako je ich meno a kardinalít a je zodpovedná za udržiavanie hierarchie syn rodič. Od nej sú odvodené triedy reprezentujúce vlastnosť samotnú a skupinu vlastností `CFMFeature` a `CFMGroup`.



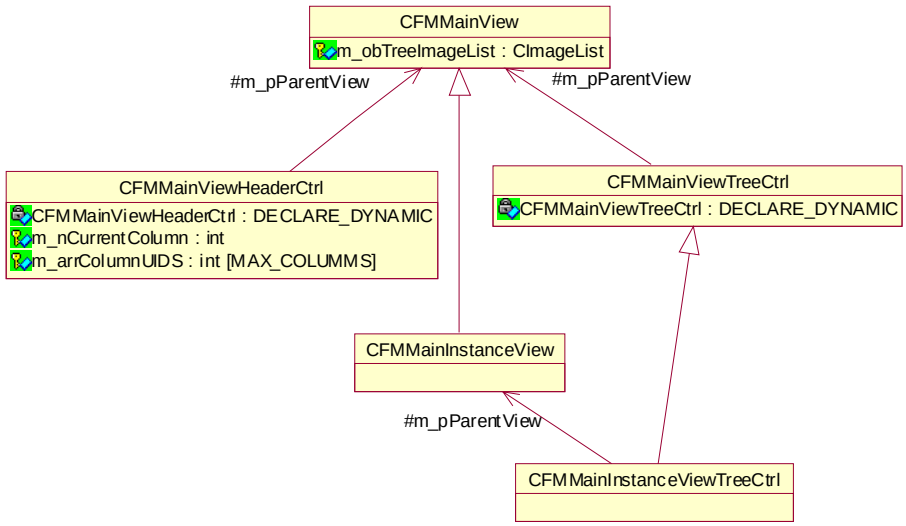
Obr. 29 Model - Základná hierarchia tried

Okrem toho sú jednotliví potomkovia triedy `CFMelement` zodpovední aj za vytváranie dialógov, ktoré umožňujú používateľovi meniť ich vlastnosti. Základná hierarchia tried dialógov je na Obr. 30.



Obr. 30 Model - Hierarchia tried dialógov

Poslednú skupinu tried, ktorá implementuje model, tvoria triedy pohľadov na model a príslušné pomocné triedy. Základná hierarchia je na Obr. 31.



Obr. 31 Model - Hierarchia tried pohľadov

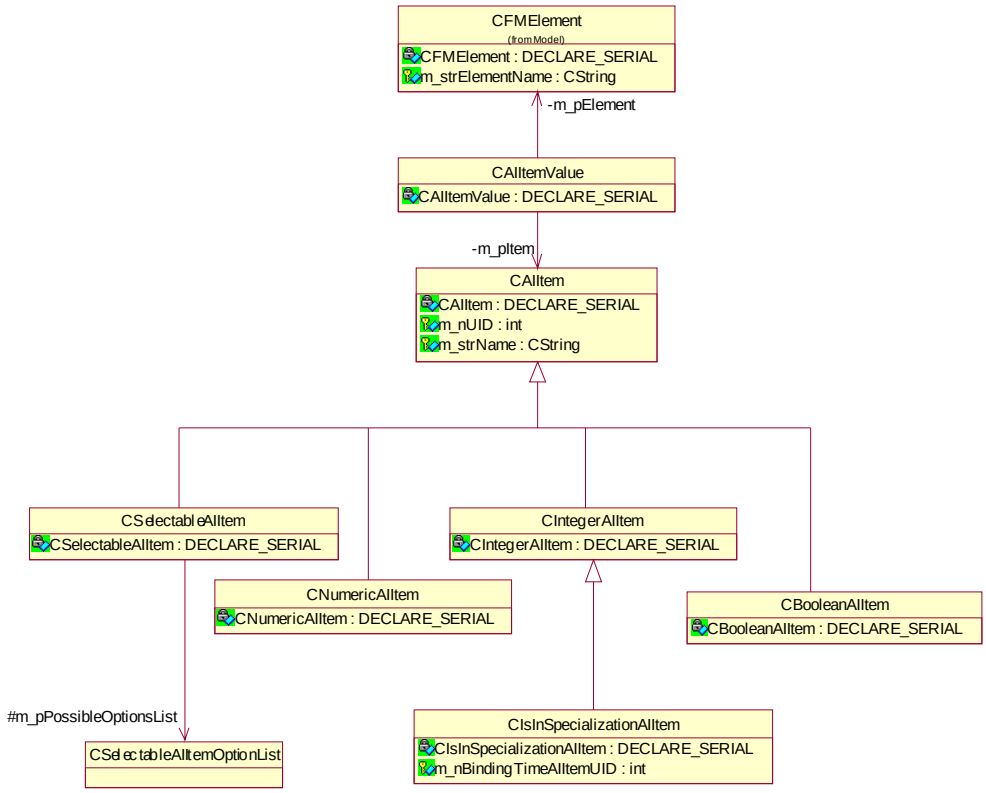
Základom implementácie pohľadu na model je trieda `CFMMainView` a dve jej pomocné triedy `CFMMainViewTreeCtrl` a `CFMMainViewHeaderCtrl`. Trieda CFMMainView implementuje pohľad vo forme stromu s možnosťou zobrazovania ďalších stĺpcov. Trieda `CFMMainViewHeaderCtrl` slúži práve na interakciu s používateľom a umožňuje mu meniť veľkosť stĺpcov pridávať a odoberať stĺpce a určovať, ktoré informácie sa v nich majú zobrazovať. Trieda `CFMMainViewTreeCtrl` nemá zvláštnu úlohu. Jej jediným poslaním je zachytávať správy OS, ktoré sú posielané priamo do prvku TreeControl a predávať ich svojej nadradenej triede `CFMMainView`. Dôvod jej vytvorenia je čisto pre to, že knižnica MFC by neumožnila zachytávať všetky správy OS v triede `CFMMainView`.

Druhou dvojicou tried sú triedy `CFMMainInstanceView` a `CFMMainInstanceViewTreeCtrl`. Tie implementujú pohľad na používateľom vytváranú inštanciu modelu. Zobrazenie inštancie modelu od zobrazenia modelu sa v podstate líši len v dvoch bodoch. Prvým je zobrazenie zaškrtávacích polí v pohľade, ktorými sa prvky modelu vyberajú do inštancie. Druhým rozdielom je to, že referencovaná vlastnosť sa pri zobrazení inštancie zobrazuje aj s podvlastnosťami.

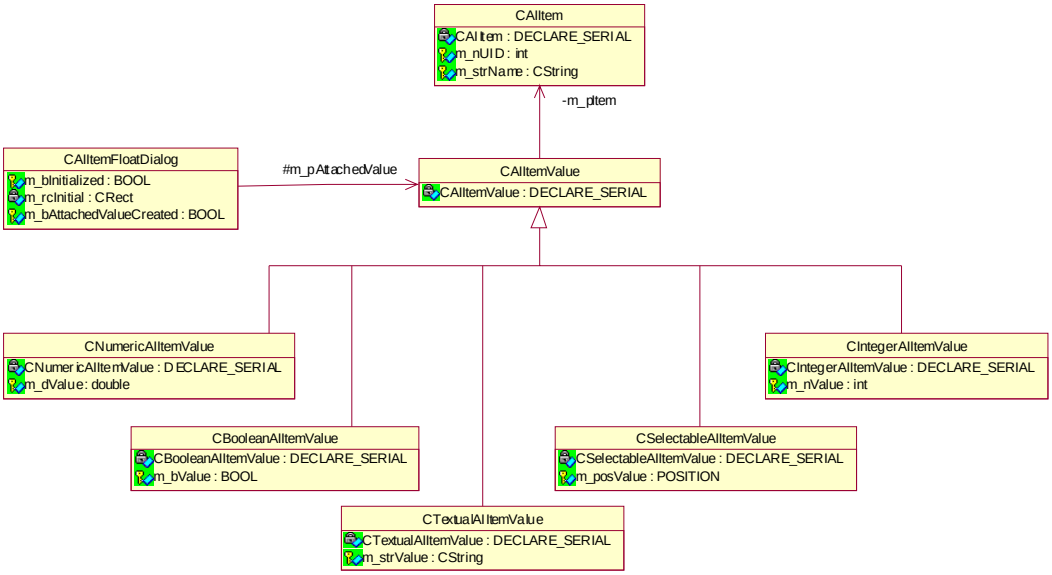
6.3 Návrh časti Metamodel

Subsystém metamodel zodpovedá za udržiavanie štruktúry prídavných informácií a priradenie ich hodnôt k jednotlivým prvkom modelu.

Navrhovaná hierarchia tried pre časť Metamodel je na Obr. 32. Jej základom je trieda `ACIItem`. Tá predstavuje základnú triedu pre implementáciu prídavnej informácie. Od nej sú odvodené triedy, ktoré reprezentujú prídavné informácie s hodnotou pre každý implementovaný typ (viď. Obr. 32). Vzťah medzi prvkom modelu a prídavnou informáciou predstavuje jej hodnota pre kontrétny prvok modelu. Tú implementuje trieda `CAIItemValue`. Každý typ prídavnej informácie má potom aj typ hodnoty (viď Obr. 33).



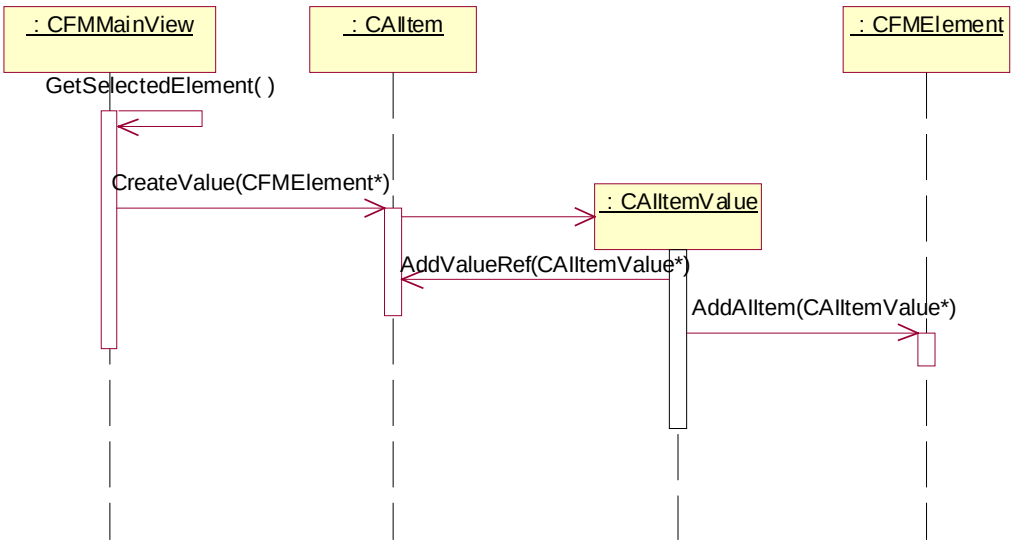
Obr. 32 Metamodel - Hierarchia tried



Obr. 33 Metamodel - Hierarchia tried pre hodnotu prídavnej informácie

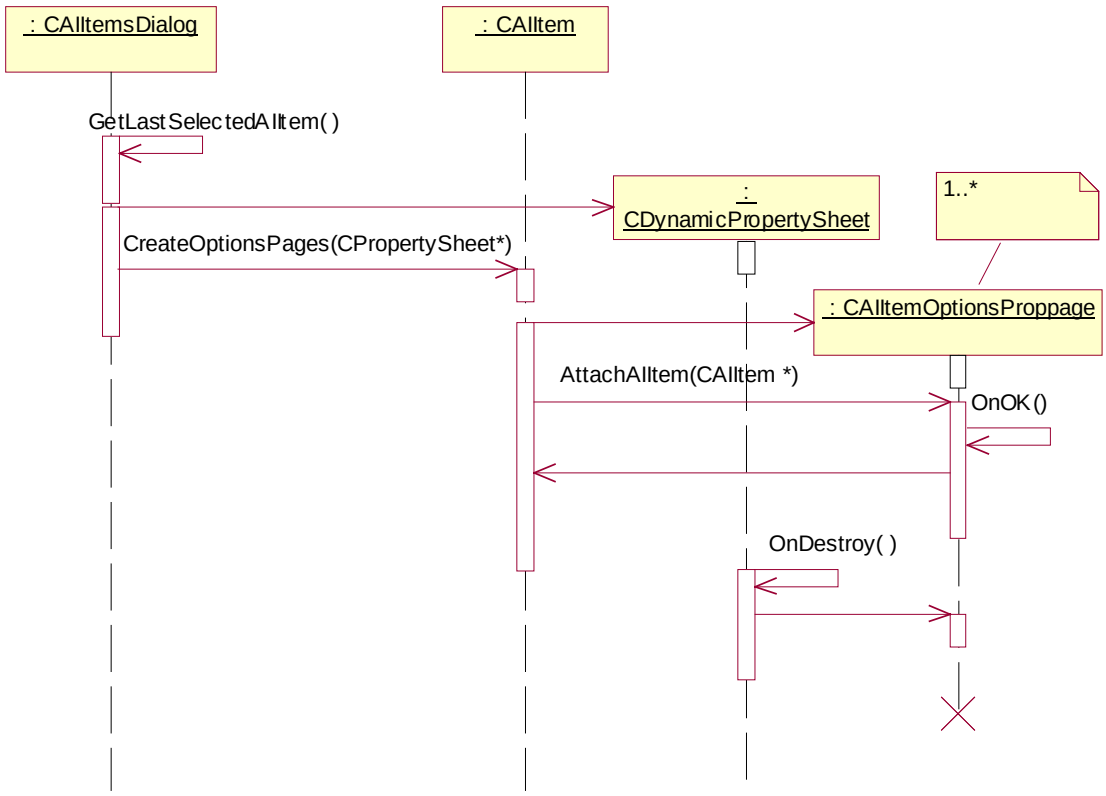
Trieda `CAIItem` slúži tiež ako továrenská trieda pre vytváranie svojich hodnôt tj. inštancií nejakej triedy odvodenej z `CAIItemValue`. Spoluprácu objektov pri vytváraní hodnoty prídavnej informácie demonštruje Obr. 34. Celú reťaz vyvoláva v tomto prípade `CFMMainView` ako reakciu na požiadavku používateľa na priradenie hodnoty prídavnej informácie pre nejaký prvok. Z Obr. 34 je tiež vidno, že za realizáciu prepojenia medzi prídavnou informáciou a prvkom

modelu, teda medzi inštanciami tried `CFMElement` a `CAIItem`, je zodpovedná trieda `CAIItemValue`.



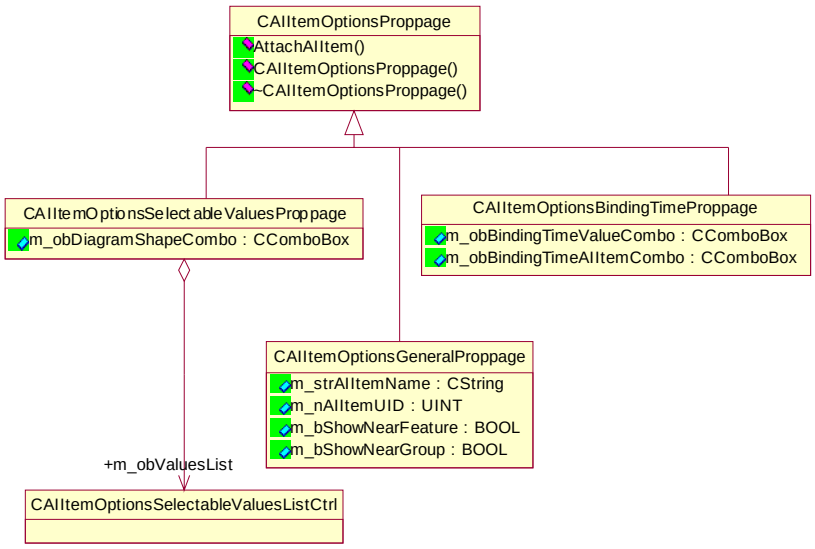
Obr. 34 Metamodel - Aktivity pri vytváraní hodnoty prídavnej informácie

Obdobne, ako v prípade vytvárania triedy CAIItemValue, je to aj v prípade vytvárania dialógov, ktoré umožňujú nastavovať vlastnosti pre prídavné informácie Obr. 35.



Obr. 35 Metamodel - Vytváranie dialógov pre nastavenia vlastností prídavnej informácie

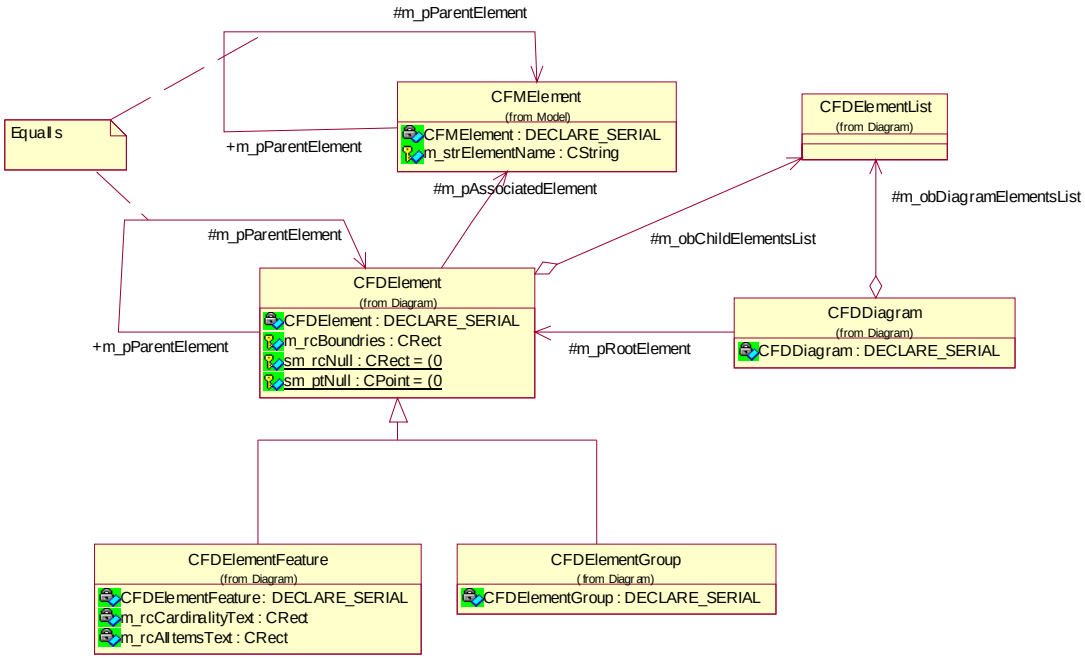
Iniciátorom je trieda `CAIItemsDialog`, ktorá umožňuje používateľovi spravovať všetky prídavné informácie. Trieda odvodená od `CAIItem` môže zobrazovať aj viac ako jeden dialóg pre nastavenia. To umožňuje vybrať všeobecné nastavenia pre všetky triedy prídavných informácií do jedného dialógu a ten zobraziť vždy a špeciálne nastavenia do dialógu druhého, ktorý sa zobrazuje podľa potreby. Dialógy sa zobrazujú potom ako záložky a za ich zobrazenia a likvidáciu je zodpovedná trieda `CDynamicPropertySheet`. Hierarchia týchto dialógov je na Obr. 36.



Obr. 36 Metamodel - Hierarchia tried dialógov

6.4 Návrh časti Diagram

Subsystém diagram zodpovedá za udržiavanie informácií o diagramoch vlastností nakreslených v modeli. Navrhovaná hierarchia tried je na Obr. 37.



Obr. 37 Diagram – Základná hierarchia tried

Základom návrhu sú dve triedy. Prvou je `CFDElement`. Táto je obdobná triede `CFMElement`. Jej hlavnou úlohou je vytvárať hierarchiu presne tak, ako tomu je pri triede `CFMElement`. Jedna inštancie triedy `CFDElement`, alebo tried z nej odvodených, reprezentuje v diagrame jednu inštanciu triedy `CFMElement`, alebo triedy z nej odvodenej.

Druhou základnou triedou je `CFDDiagram`. Tá udržiava po kope inštancie `CFDElement`, ktoré dokopy tvoria jeden diagram vlastností.

6.5 Návrh vyhodnocovača obmedzení

Na vyhodnotenie správnosti zapísaných obmedzení, ako aj na vyhodnotenie ich hodnoty pri overovaní inštancie modelu, sa používa vlastný syntaktický analyzátor implementovaný ako sada rekurzívnych procedúr v triede `CConstraintParser`.

`CConstraintParser` vie vyhodnotiť logické výrazy s menami identifikátorov oddelenými bodkou. Meno identifikátora môže obsahovať znak medzery, okrem začiatku a konca. Začiatočná a koncová medzera v mene identifikátora je ignorovaná. Vo výraze ďalej môžu byť operátory podľa Tab. 1. Priorita znamená, v akom poradí sa analyzátor snaží vyhodnocovať operátory. Operátor s najvyššou prioritou sa vyhodnocuje ako prvý.

Operátor	Precedencia	Význam
>	1	Implikácia
&	3	Logické AND
|	2	Logické OR
+	2	Logické XOR
!	4	Negácia

Tab. 1 Parser - Prehľad operátorov

Pre analyzátor bola najprv zostrojená takáto gramatika G1 = { N1 T1, P1 S1 }

N1 = {I, E, T, F, Impl, And, Or, FullName}

T1 = { &, |,+ , >, ., !, oneName }

P1 = {

I → I Impl E | E | ! I

E → E Or T | T

T → T And F | F

F → (E) | FullName

FullName → oneName . FullName | oneName

Impl → >

Or → | | +

And → & }

S1 = I

Analyzátor ale používa jej zľava nerekurzívny ekvivalent G2 = { N2, T2, P2, S2 }, kde e je prázdne slovo.

N2 = {I, I_2, E E_2, T, T_2 , F, Impl, And, Or, FullName}

T2 = T1

P2 = {

I → E I_2 | ! I

I_2 → Impl E I_2 | e

E → T E_2

E_2 → Or T E_2 | e

T → F T_2

T_2 → And F T_2 | e

F → (E) | FullName

FullName → oneName . FullName | oneName

Impl → >

Or → | | +

And → & }

S2 = I

A konečne tabuľka riadiaca LL1 rozklad je v Tab. 2 . Uvedená je len jej podstatná časť. V prípade, že je na vrchole zásobníka terminál, vylúči sa, ak je na vstupe ten istý terminál a nastane chyba, ak je na vstupe iný terminál. Prázdne polia znamenajú chybu.

	!	>	&	|	+	(	)	oneName	e
I	! I					E I_2		E I_2	
I_2		Impl E I_2					e		e
E		e				T E_2		T E_2	
E_2				Or T E_2			e		e
T		e		e		F T_2		F T_2	
T_2			And F T_2				e		e
F						(E)		oneName	
Impl		>							
Or				|	+				
And			&						

Tab. 2 Parser - Tabuľka LL1 rozkladu G2

6.6 K implementácii nástroja

Niektoré prístupy k tvorbe softvéru (Meyer 1997) neoddeľujú striktne návrh a samotnú implementáciu. Niekedy sa samotný kód považuje za spôsob vyjadrenia návrhu. Pre implementáciu tohto nástroja bol použitý obdobný postup. Po základom návrhu tried a hierarchie dedenia (realizovanom bez CASE nástroja) bol tento nenesený priamo do zdrojového kódu aplikácie. Až s takto vytvoreného kódu bol potom spätne CASE nástrojom vygenerovaný model. Od tohoto kroku boli model a samotný kód aplikácie udržiavané tak, aby si odpovedali. Ďalej bolo možné navrhnúť rozšírenie aplikácie pomocou CASE nástoja a tento návrh preniesť do samotného kódu. Takýto postup bol používaný napríklad pri návrhu spolupráce tried navzájom. Okrem toho bolo stále možné pridať zmeny do kódu aplikácie a tie potom preniesť do modelu. Takýto postup bol použitý pri vytváraní nových tried, kde bolo jasné, ktoré virtuálne metódy má preťažiť.

Týmto postupom síce splynul návrh a impelemntácia do jedného celku. Na druhej strane bolo využitá tá najrýchlejšia cesta k vytváraniu aplikácie. Napríklad pri implementácii tried odvedených od `CFMelement` bol jednoducho skopírovaný jej kód, vymazané implementácie jej metód. Takto veľmi rýchlo vznikla šablóna do ktorej už bolo možné rýchlo vpísať telo preťaženej metódy, resp. zmazať jej deklaráciu ak preťaženie nebolo potrebné. V prípadoch kde bola nutná hlbšia analýza a precízne modelovanie bol použitý CASE nástroj.

6.7 Porovnanie s analyzovanými prístupmi

V tejto časti je prehľad toho, ktoré prístupy alebo ich prvky implementuje v tejto práci navrhovaný a realizovaný CASE nástroj a akým spôsobom sumarizovaný v jednej tabuľke (Tab. 3). Pod položkou prvok uvádzam charakteristický prvok pre prístup uvedený v položke prístup. V poli Implementované potom, či je prvok implementovaný alebo nie. V poli prístup potom uvádzam pre ktorý prístup resp. metódu je prvok charakteristický. CE FODA znamená Czarneckého rozšírenie pôvodnej FODA notácie (Czarnecki, Eisenecker 2000), VVLSP znamená prístupy na Vyjadrenie variability v líniách softvérových produktov (časť 2.6) a MDP FM prístup použitý pri Multiparadigmovom návrhu s modelovaním vlastností (časť 2.3).

Prvok	Prístup	Implementované
Povinné a voliteľné vlastnosti	FODA	Nepriamo. O povinnej alebo voliteľnej vlastnosti rozhoduje kardinalita. Jej vhodným nastavením sa dá docieliť význam povinnej a voliteľnej vlastnosti.
Alternatívne vlastnosti	FODA	Nepriamo. O počte vybraných vlastností v skupine rozhoduje kardinalita. Jej vhodným nastavením sa dá docieliť význam alternatívnych vlastností
Diagram modelu vlastností	FODA	Áno.
Zobrazenie prídavných informácií v diagrame	FODA	Áno
Zadávanie triviálnych obmedzení	FODA	Nepriamo. Obmedzenia je možné zaznamenať rostredníctvom predikátovej logiky.
Zobrazenie triviálnych obmedzení v diagrame	FODA	Nie.
Podpora informácií o vlastnostiach ako je Vlastník a pod.	FODA	Nepriamo. Využitím možnosti definovať vlastnú štruktúru prídavných informácií.
Vytváranie inštancií	FODA	Áno
Obmedzenia prídavných informácií len na niektoré elementy v modely		Nie
Skupina disjunktívnych (OR) vlastností	CE FODA	Nepriamo. O počte vybraných vlastností v skupine rozhoduje kardinalita. Jej vhodným nastavením sa dá docieliť význam alternatívnych vlastností.
Rozdelenie vlastností do vrstiev	FORM	Nepriamo. Dá sa využiť vhodne definovaná prídavná informácia
Rozdelenia do vrstiev v diagrame	FORM	Nie. Je ale možnosť zobraziť zaradenie do vrstiev pri vlastnosti v diagrame a rozdelenie do vrstiev prekresliť ručne.
Prídavná informácia na hrane zoskupených vlastností	FODAcom	Nie. FODAcom využívala takúto prídavnú informáciu na vyjadrenie, že sa jedná o skupinu alternatív, a že v konečnej inštancii bude vybraná len jedna z vlastností.
Bod rozšírenia	ReatuRSEB	Nepriamo. Dá sa využiť vhodne definovaná prídavná informácia. Nie je ale možné povedať systému aby vynútil v takejto skupine kardinalitu [1..1].
Vizualizácia bodu rozšírenia v diagrame	ReatuRSEB	Nepriamo. Dá sa využiť vhodne definovaná prídavná informácia a nastaviť spôsob jej zobrazenia v diagrame
Takzvaná disjunktívna (OR) špecializácia	VVLSP	Nepriamo. Dá sa využiť vhodne definovaná prídavná informácia ako tomu bolo v prípade bodu rozšírenia ale nie je obmedzenie na určenie kadrinality v tomto prí-

Prvok	Prístup	Implementované
		pade.
Ohodnotenie vlastnosti kardinalitou	CE FODA	Áno
Ohodnotenie skupiny vlastností kardinalitov	CE FODA	Áno
Zápis komplexných obmedzení v predikátovej logike	MPD FM	Áno
Parametrizácia v modely vlastností	MPD FM	Nie.
Štruktúra prídavných informácií podľa MDP FM	MPD FM	Nepriamo.
Atribúty vlastností v modely vlastností	CE	Nie.

Tab. 3 Porovnanie navrhovaného systému s analyzovanými prístupmi s modelovaniu vlastností

6.8 Porovnanie s analyzovanými systémami

Táto časť ponúka porovnanie navrhovaného a realizovaného systému s troma ďalšími systémami určenými na modelovanie vlastností (Tab. 4).

Prvok	Navrhovaný systém	Captain Feature	fmp
Editovanie metamodelu	Obmedzené	Nie	Obmedzené
Editovanie modelu	V pohľade typu strom	V diagrame modelu vlastností	V pohľade typu strom
Definovanie prídavných informácií	Áno	Nie	Áno
Referencie na koncept	Áno	Áno	Áno
Zápis obmedzení	Upravenou predikátovou logikou	Vlastním formálnym jazykom	Vlastným formálnym jazykom
Vytváranie inštancie	Áno	Áno	Áno
Vytváranie inštancií referencovaných konceptov	Áno	Nie	Nie
Overenie inštancie vzhľadom na obmedzenia	Áno	Áno	Áno
Overenie inštancie v časti referencované-	Áno	Nie	Nie

Prvok	Navrhovaný systém	Captain Feature	fmp
ho konceptu			
Zobrazenie diagramu pre koncept	Áno	Áno (jediný možný pohľad na model)	Nie
Zobrazenie len časti diagramu pre koncept	Áno	Nie	Nie
Zobrazenie prídavných informácií v diagrame vlastností	Áno	Nie (neumožňuje definovať žiadne prídavné informácie	Nie (nemá diagram vlastností)
Priradenie atribútu vlastnosti	Nie	Áno	Áno
Formát pracovného súboru	Binárny	XML	XML
Špecializácia konceptu	Nie	Nie	Áno

Tab. 4 Porovnanie implementovaného systému s ďalšími analyzovanými systémami na modelovanie vlastností.

6.9 Ďalšie možnosti rozšírenia systému

Prvým s možných rozšírení je pridanie informácie na hranu diagramu modelu vlastností. Pôvodná Czarneckého notácia sa tomu úplne vyhýbala. Výnimkou bolo iba zahrnutie skupiny hrán do alternatívy alebo disjunkcie. Rozšírená Czarneckého notácia síce pridáva na hranu informáciu, ale jedná sa len o kardinalitu. Naproti tomu FODA aspoň kategorizuje vzťah medzi dvoma vlastnosťami. Takáto informácia sa dá vyjadriť vhodným znázornením aj na hrane diagramu modelu údajov. Pretože rôzne prístupy zobrazujú ha hrane diagramu modelu vlastností rôzne informácie, spočíva aj navrhované rozšírenie len v možnosti určiť, ktorá s prídavných informácií sa bude zobrazovať na hrane diagramu modelu vlastností.

Zaujímavou možnosťou je zobrazenie triviálnych obmedzení medzi vlastnosťami, priamo v diagrame modelu údajov. Takúto možnosť nemá žiadny z analyzovaných prostriedkov na podporu modelovania vlastností. Je ju však vidieť v niektorých článkoch najmä v malých modeloch vlastností alebo modeloch s málo obmedzeniami. Pre navrhovanú reprezentáciu s obmedzeniami zapísanými v tvare výrazov predikátovej logiky nie je jednoduchá takéto závislosti získať a potom aj zobraziť. Preto sú dve možnosti ako takéto rozšírenie realizovať. Prvou je nechať používateľovi možnosť nakresliť do diagramu modelu vlastností nejaké obmedzenia podľa jeho ľubovôle. V takom prípade by bolo na používateľovi, ktoré obmedzenia do diagramu naznačí a ktoré nie. Druhou možnosťou by bolo zavedenie špeciálnej sady obmedzení, ktoré by potom bolo možné v diagrame vizualizovať.

Poslednou zo spomenutých možností je doimplementovanie prístupu k modelu cez mechanizmus OLE automatizácie (OLE Automation) tak, ako bolo špecifikované. OLE automatizácia je jedna z technológií, ktorou sa rieši spoluprácu aplikácií na báze MS Windows. Jedná sa o možnosť pristupovať k dokumentu pomocou ľubovoľného jazyka cez jednotné rozhranie. Takto môže pristupovať k modelu vlastností akákoľvek aplikácia, ktorá podporuje túto technológiu. Taktiež je otvorená aj možnosť vnútorného skriptovania (vytvárania makier) pre samotnú aplikáciu. Implementovanie tejto technológie teda umožní využívať model vlastností trebárs aj na riadenie konfigurácie iného modelu, ako tomu je v fm2uml module. Oproti nemu má toto rieše-

nie tú výhodu, že užívateľ môže tento proces riadiť sám ak pozná možnosti svojho UML nástroja. Tým je dosiahnutá aj nezávislosť od použitého nástroja.

7 Záver

Prvá časť tejto práce predstavuje modelovanie vlastností ako techniku konceptuálnej analýzy používanej v doménovom inžinierstve. Z tohto dôvodu práca predstavuje aj proces analýzy domény. Ďalej sa prvá časť tejto práce zameriava na predstavenie metód, ktoré používajú modelovanie vlastností. Podrobnejšie predstavuje metódu FODA (časť 2.1) ktorá prvá priniesla techniku modelovania vlastností. Modelovanie vlastností, tak ako je definované v metóde FODA sa stalo základom pre ďalšie rozširovanie, ktoré popisuje aj táto práca.

Prvým z rozšírení modelovania vlastností je rozšírenie pre generatívme programovanie (časť 2.2). Toto rozšírenie prináša do modelovania vlastností takzvané voliteľné vlastnosti a takzvané disjunktívne alebo OR vlastnosti. Druhým je rozšírenie modelovania vlastností pre použitie v multiparadigmovom návrhu (časť 2.3). Rozšírenie sa v tomto prípade netýka diagramu modelu vlastností alebo jeho notácie. Prínos spočíva v zavedení jednoduchej predikátovej logiky pre zápis obmedzení. Tretie rozšírenie predstavujú dve špeciálne notácie, ktoré do modelovania vlastností prinášajú takzvaný bod rozšírenia. Jedná sa o notácie FODACom a FeatuRSEB (časti 2.4 a 2.5). Štvrtým a najpodstatnejším rozšírením bolo zavedenie kardinalít do modelu vlastností (časť 2.7). Výsledky analitickej časti tejto práce boli publikované už skôr (Šípka 2005).

V druhej časti (kapitola 4) popisuje táto práca metamodel pre modelovanie vlastností. Tento metamodel ukazuje ako je možné všetky analyzované notácie obsiahnuť v jednej notácii a to vhodným použitím kardinality a prídavnej informácie. V prípade, že zavedieme kardinalitu do modelu vlastností, nie je potrebné ďalej definovať povinné ani voliteľné vlastnosti, pretože táto informácia je obsiahnutá priamo v kardinalite. Ak pridáme kardinalitu aj skupine vlastností (tak ako je definovaná v prezentovanom metamodely v časti 4.2) nie je potrebné rozlišovať ani medzi alternatívnymi a disjunktívnymi vlastnosťami. Na druhej strane zavedenie kardinality neobsiahne zavedenie bodov rozšírenia tak ako sú použité v notáciách FODACom a FeatuRSEB. Preto je opäť ku skupine vlastností treba definovať prídavnú informáciu. Tá môže definovať že takáto skupina je jedným s bodov rozšírenia ale môže byť použitá aj na iné účely.

V poslednej časti bol popísaný proces vývoja CASE nástroja, ktorý implementuje v tejto práci navrhnutý metamodel. Návrh a implementáciu takétoto nástroje je prezentovaná ako riešenie problematiky podpory modelovania vlastností. Kapitola 5 popisuje širšiu špecifikáciu požiadaviek na takýto nástroj a podáva zároveň jej zdôvodnenie. Ďalej je tu navrhnuté v akých inkrementoch je možné nástroj navrhnúť a implementovať. Kapitola 6 potom popisuje návrh a implementáciu takéhoto nástroja.

Ďalší možný rozvoj v tejto problematike naznačuje aj kapitola 6. Ten sa dá rozdeliť na niekoľko hlavných prúdov. Prvým je podpora vytvárania inštancií. Tú má zatiaľ nástroj implementovanú len v základnej forme. Ďalšie rozšírenie spočíva v poskytnutí prehľadu alebo zdôvodnenia, prečo bolo konkrétne obmedzenia vyhodnotené ako nesplnené. Zaujímavým by bolo aj vytvorenie podpory pre parametrizáciu v modely valstností. Takéto rozšírenie so sebou prináša nie len rozšírenie používateľského prostredia programu ale aj veľmi závažné rozšírenie mechanizmu vyhodnocovania obmedzení. Parametrizácia totiž pridáva do obmedzení, doteraz zapísaných len v predikátovej logike aj kvantifikátory.

Iným možným rozvojom je podpora integrovania modelu vlastností s inými modelmi. Tu je treba v prvom rade doimplementovať navrhovaný mechanizmus pristupovania k modelu pomocou OLE. Okrem toho je integrácia modelu vlastností s ostatnými modelmi otvorenou oblasťou. Existujú síce trendy, použiť model vlastností, resp. jeho inštanciu na konfigurovanie iných modelov prípadne celého kódu. Toto ale nie je zďaleka jediný možný prístup využitiu modelovania vlastností pri tvorbe ďalších modelov. Je možné bádať v oblasti automatizovanej alebo poloautomatizovanej tvorby ostatných modelov práve na základe modelu vlastností.

Literatúra

(Aniszczyk 2005) ANISZCZYK, Chris: Using GEF with EMF. ©IBM Corporation. 2005.

(Bednasch a i. 2005) BEDNASCH, T. ENDLER, C. LANG M.: CaptainFeature. 2000-2004. https://sourceforge.net/projects/captainfeature/ (29.11.2005)

(Bontemps, Heymans 2004) BONTEMPS. HEYMANS: Semantics of FODA Feature Diagrams. "Proc. Workshop on Software Variability Management for Product Derivation Towards Tool Support". August 2004. http://www.info.fundp.ac.be/~ybo/docs/splc04/foda-semantics.pdf. (12.12.2004)

(Cohen, Stanley a i. 1992) COHEN, Sholom G. STANLEY, Jay L. PETERSON, A. Spencer. KRUT, Robert W.: Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain. Technical Report CMU/SEI-91-TR-028, ESD-91-TR-028. Jún 1992. http://www.sei.cmu.edu/pub/documents/91.reports/pdf/tr28.91.pdf. (12.12.2004)

(Coplien 1999) COPLIEN, James O.: Multi-Paradigm Design for C++. Addison-Wesley. 1999.

(Czarnecki a i. 2005) CZARNECKI, Krzysztof. ANTKIEWICZ, Michał. PIETROSZEK, Krzysztof. KIM, Peter. LAU, Sean: fmp: Feature Modeling Plug-in. ver. 0.5.2. 2005. http://gp.uwaterloo.ca/fmp/ (29.11.2005)

(Czarnecki, Antkiewicz 2005) CZARNECKI, Krzysztof. ANTKIEWICZ, Michał: fmp2rsm: Mapping Features to UML 2.0 Models Plug-in. 2005. http://gp.uwaterloo.ca/fmp2rsm/ (29.11.2005)

(Czarnecki, Eisenecker 2000) CZARNECKI, Krzysztof. EISENECKER, Ulrich W.: Generative Prog-raming: Principles, Techniques and Tools. Addison-Wesley. 2000.

(Czarnecki, Helsen a i. 2004) CZARNECKI, Krzysztof. HELSEN, . EISENECKER, Ulrich W: Staged Configuration Using Feature Models. R.L. Nord (Ed.): SPLC 2004, LNCS 3154, pp. 266–283, 2004.

(Eclipse contributors and others 2005) Eclipse contributors and others: The Eclipse project. http://www.eclipse.org/. (1.12.2005)

(Giss, Favaro a i. 1998)GISS, Martin L. FAVARO, John. D'ALESSANDRO, Massimo: Integrating Feature Modeling with the RSEB. In: P. Devanbu and J. Poulin, editors, Proc. of 5th International Conference on Software Reuse, pages 76–85, Vicoria, B.C., Canada. IEEE Computer Society Press .1998. http://www.favaro.net/john/home/publications/rseb.pdf (11.10.2004).

(Gurp, Bosh a i. 2001) GURP, Jilles van. BOSH, Jan. SVAHNBERG, Mikael: On the Notion of Variability in Software Product Lines. Proceeedings of WICSA 2001. 2001. http://www.jillesvangurp.com/publications/notionOfVariability.pdf (11.10.2004).

(ISO/IEC 14977 1996) ISO/IEC 14977: 1996(E). A syntactic metalanguage Extended BNF http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf., alebo http://www.nist.fss.ru/hr/doc/mstd/iso/14977-96.htm. (12.12.2004)

(Jackson 2002)JACKSON, Daniel: Micromodels of software: Lightweight modeling and analysis with Alloy. Software design group, MIT lab for computer science. 2002

(Kang, Cohen a i. 1990) KANG, Kyo C. COHEN, Sholom G. HESS, James A. NOVAK, William E. PETERSON, A. Spencer: Feature-oriented domain analysis (FODA):A feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA. November 1990.

(Kang, Kim a i. 1998) KANG, Kyo C. KIM, Sajoong. LEE, Jaejoon. KIM, Kijoo. KIM, Gerard Jounghyun. SHIN, Shin: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH). 1998.

(Majewski 2004) MAJEWSKI, Bo: A Shape Diagram Editor. ©Cisco Systems Inc. 2004. http://eclipse.org/articles/Article-GEF-diagram-editor/shape.html. (1.12.2005)

(Meyer 1997) MEYER, D: Object-Oriented Software Construction. Prentice Hall, second edition. 1997.

(Microsoft Corporation 1998) Microsoft Visual Studio 6. Microsoft Corporation. 1998

(Pure-systems 2005) PURE-SYSTEMS: pure::variants. 2004-2005. http://www.pure-systems.com/Variant_Management.49.0.html. (29.11.2005)

(Riebisch, Böllert a i. 2002) RIEBISCH, Matthias. BÖLLERT, Kai. STREITFERDT, Streitferdt. PHILIPPOW, Ilka: Extending feature diagrams with UML multiplicities. Integrated Design and Process Technology, IDPT-2002. June, 2002. http://www.theoinf.tu-ilmenau.de/~streitdf/TheHome/own/data/IDPT2002-paper.pdf. (21.21.2004)

(Smith, Green a i. 1996) SMITH, . GREEN, :Toward Practical Applications of Software Synthesis. In Proceedings of FMSP'96, The First Workshop on Formal Methods in Software Practice, San Diego, California, January 1996

(Software Engineering Institute 1997) Software Engineering Institute: Model-Based Software Engineering (MBSE). http://www.sei.cmu.edu/mbse/is.html. (12.9.2004)

(Šípka 2005) ŠÍPKA, Miloslav: Exploring the Commonality in Feature Modeling Notations. In: Mária Bieliková, editor, Proceedings of IIT.SRC 2005: Student Research Conference in Informatics and Information Technologies, Bratislava, pages 139-144. Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava, April 2005.

(Vici, Argentieri a i. 1998) VICI, Alesandro Dionosi. ARGENTIERI, Nicola. MANSOUR, Azza. D'ALESSANDRO, Massimo. FAVARO, John: FODAcom: An Experience with Domain Analysis in the Italian Telecom Industry. Proceeding the Fifth International Conference of Software Reuse. Victoria B. C. 1998. 1998.

(Vranić 2004) VRANIĆ, Valentino: Multi-Paradigm Design with Feature Modeling. PhD thesis, Slovak University of Technology in Bratislava. 2004. http://www.fiit.sk/~vranic/pub/MPDfm.pdf. (12.12.2004)

(Vranić 2004a) VRANIĆ, Valentino: Reconciling Feature Modeling: A Feature Modeling Metamodel. In: Mathias Weske and Peter Liggesmeyer, editors, Proc. of 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World (Net.ObjectDays 2004), Erfurt, Germany, September 2004. Springer. http://www.fiit.sk/~vranic/pub/MPDfm.pdf. (12.12.2004)

(Vranić 2005) VRANIĆ, Valentino: Inštanciovanie v čase. Osobná komunikácia. 14.11.2005. E-mail: vranic@fiit.stuba.sk.

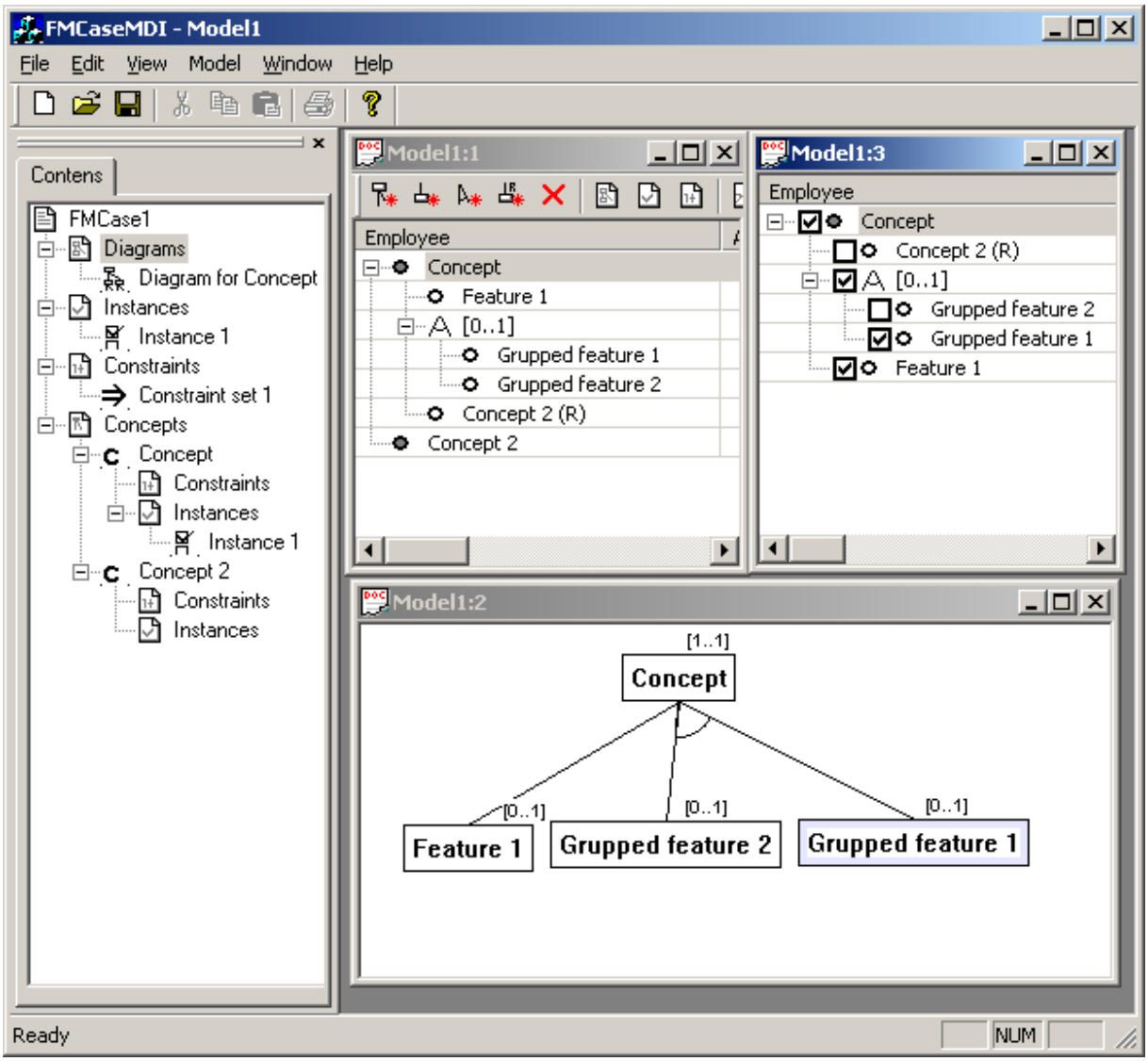
(Zhang, Sun a i. 2004) ZHANG, Hongyu. SUN, Jing. WANG, Hai: Formalizing and Analyzing Feature Models in Alloy. RMIT CS Technical Report TR-04-2. 2004. http://goanna.cs.rmit.edu.au/~hongyu/papers/feature_alloy.pdf. (12.12.2004)

Príloha A FMCaseMDI príručka používateľa

A.1 FMCaseMDI

FMCaseMDI je nástroj na podporu modelovania vlastností realizovaný ako samostatná aplikácia. FMCaseMDI umožňuje tieto základné operácie:

- Editovanie modelu vlastností vrátane obmedzení na model zapísaných v predikátovej logike.
- Editovanie metamodelu na úrovni zadefinovania prídavných informácií.
- Vytváranie inštancií modelu a ich overenie a to aj vzhľadom na čas viazania vlastnosti.
- Vytváranie diagramov modelu vlastností a zobrazovanie prídavných informácií v nich.



Obr. 38 Zákaldná obrazovka programu FMCaseMDI

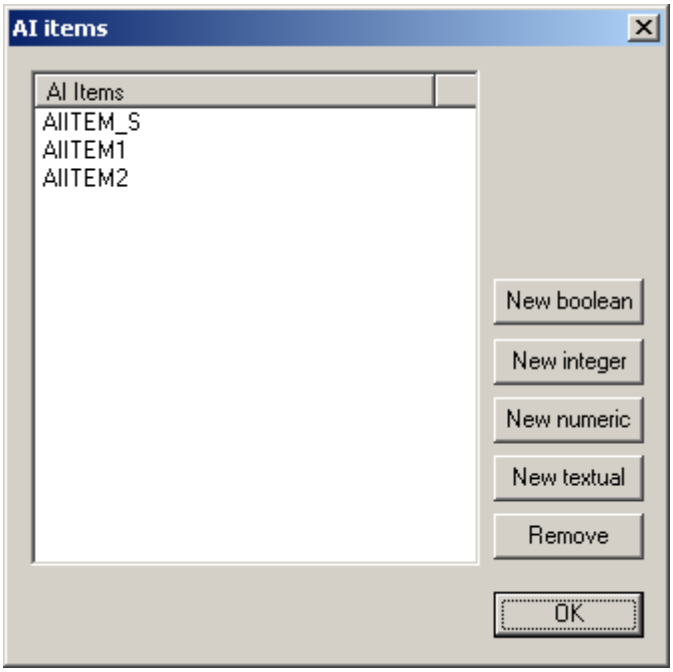
Na Obr. 38 je základná obrazovka programu FMCaseMDI. Zobrazuje jeden otvorený model. Celkom napravo sa nachádza strom, ktorý ponúka prehľad všetkých dostupných pohľadov. Hore v strede sa nachádza editor modelu, ktorý umožňuje editovanie modelu. Na ľavo hore sa nachá-

dza editor inštancií, ktorý umožňuje vytvárať inštanciu jedného konceptu vybraním vlastností. Dole sa potom nachádza diagram modelu vlastností pre koncept Concept.

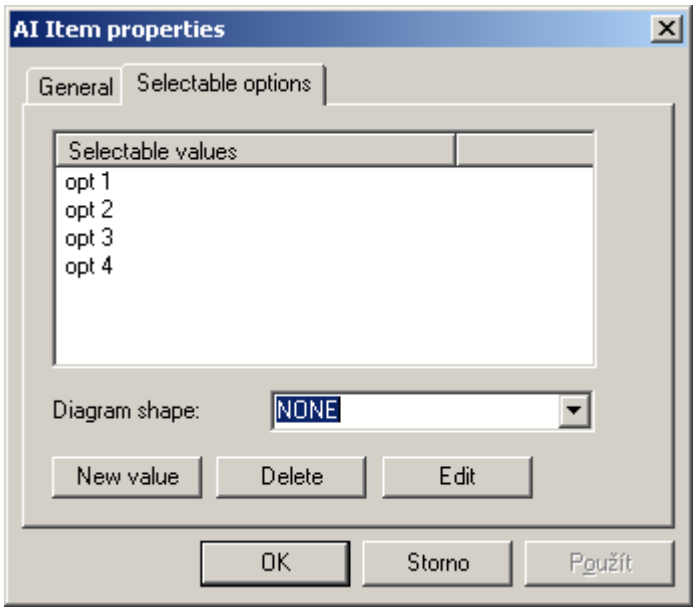
A.2 Definovanie prídavných informácií

Program umožňuje definovanie vlastnej štruktúry prídavných informácií. To je možné po voľbe Model – AI Items z menu aplikácie. Zobrazí sa dialóg z Obr. 39, v ktorom je možné pridať alebo ubrať definovanú prídavnú informáciu. Okrem prídavnej informácie, ktorej hodnota sa vyberá z predom definovaných hodnôt je nutné zadať len meno.

V prípade prídavnej informácie, ktorej hodnota sa vyberá z predom definovaných hodnôt, je nutné definovať minimálne jednu takúto hodnotu. Okrem toho je možné definovať aj zobrazenie v diagrame modelu vlastností. Obr. 40.



Obr. 39 Okno s prehľadom prídavných informácií

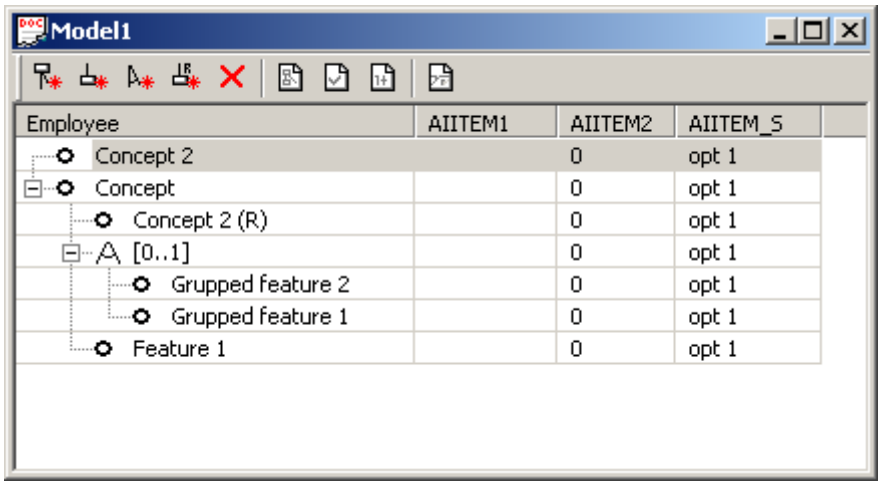


Obr. 40 Definovanie prípustných hodnôt pre prídavnú informáciu.

A.3 Editácia modelu

Na editovanie samotného modelu slúži pohľad z Obr. 41. Tu je možné pomocou menu, prístupného po stlačení ľavého tlačidla myši, alebo pomocou tlačidiel na nástrojovej lište

- Pridať nový koncept
- Pridať novú podvlastnosť pod koncept, inú vlastnosť alebo skupinu
- Pridať novú skupinu vlastností
- Pridať novú referenciu na koncept
- Zmazať páve vybraný prvok z modelu v vlastností
- Otvoriť (alebo vytvoriť nový ak ešte neexistuje) diagram vlastností pre vybraný koncept alebo aj vybranú vlastnosť.
- Vytvoriť novú inštanciu konceptu
- Vytvoriť novú sadu obmedzení pre koncept
- Zmeniť niektoré vlastnosti vybraného prvku



Obr. 41 Okno pre editovanie modelu

V pohľade sa dajú konfigurovať prídavné informácie, ktorých hodnota sa má zobraziť. Funkčnosť je prístupná cez menu dostupné po stlačení pravého tlačidla myši na záhlaví stĺpcov podhľadu.

Okrem týchto funkcií slúži pohľad z Obr. 41 na nastavovanie hodnôt prídavných informácií. To sa realizuje dvojitým kliknutím na vlastnosť v stĺpci, v ktorom je zobrazená hodnota prídavnej informácie, zadaním jej hodnoty a potvrdením klávesom ENTER.

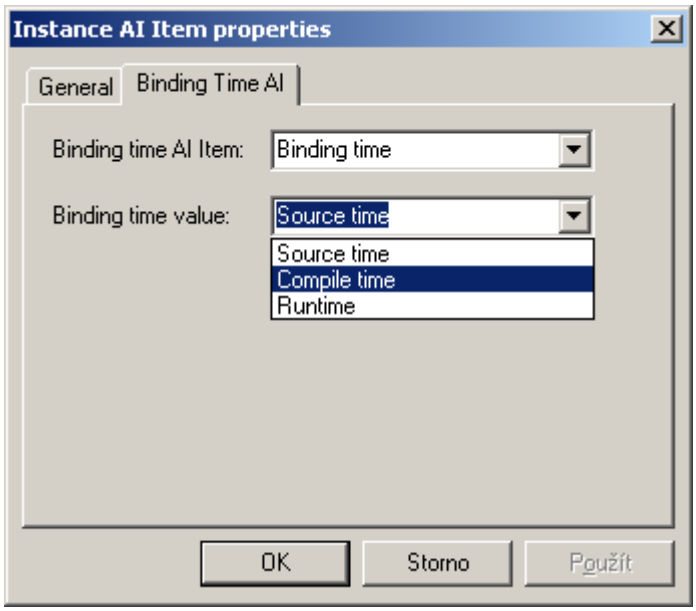
A.4 Editovanie diagramu

Program ponúka možnosť vytvoriť diagram pre ľubovoľnú vlastnosť v modeli. Diagram je nožné vytvoriť pomocou tlačidla alebo pomocou menu. Po vytvorení je v diagrame len jediná vlastnosť. Podvlastnosti sa pridávajú naraz kliknutím na vlastnosť v diagrame a voľbou pridať potomkov.

V nastaveniach prídavných informácií je možné nastaviť, či sa má vedľa vlastnosti v diagrame zobrazovať aj hodnota danej prídavnej informácie. Viac v časti A.2.

A.5 Vytváranie inštancie konceptu

Inštanciu konceptu je možné vytvoriť vybraním konceptu a voľbou tlačidla . Okrem toho je možné inštancii nastaviť aj dobu tvorenia. Pred tým je nutné vhodne definovať prídavnú informáciu, ktorá bude niesť údaj o tom, v ktorej dobe je vlastnosť viazaná. Ďalej pod touto vlastnosťou nutné zadefinovať jednotlivé doby viazanie. Až po týchto krokoch je možné inštancii nastaviť, ktorú s prídavných informácií má považovať za dobu viazania. Taktiež potom treba určiť, v ktorom čase je inštancia vytváraná, viď Obr. 42.



Obr. 42 Voľba prídavnej informácie, ktorá rozhoduje o dobe tvorenia inštancie.

A.6 Zápis obmedzení a ich vyhodnotenie

Obmedzenia sú organizované do takzvaných sád obmedzení, aby bolo možné ich hierarchicky organizovať. Inštancia je overovaná voči všetkým sadám obmedzení. Novú sadu obmedzení je možné vytvoriť vybraním konceptu a zvolením tlačidla .

Obmedzenia sa zapisujú v tvare logických výrazov. Na vlastnosti je možné odvolávať sa dvoma spôsobmi. Prvý spôsob je uvedenie len mena vlastnosti. Tento spôsob možno použiť, ako má vlastnosť v danom koncepte jedinečné meno. V opačnom prípade je nutné zadať celé meno vlastnosti, teda uviesť cestu od mena konceptu až po vlastnosť oddelenú bodkami, napr. Concept . Feature 1. Okrem vlastností konceptu je možné odkazovať aj na vlastnosti referencovaných konceptov. V takomto prípade sa musí ale uviesť celé meno vlastnosti od pôvodného konceptu. Pravidlá samotné sa zapisujú do jedného riadku. Prázdne riadky sú ignorované.

V kontextovom menu sa nachádza aj možnosť overenia inštancie konceptu. Inštancia je overená voči všetkým sadám obmedzení svojho konceptu. Výsledkom vyhodnotenia obmedzenia môžu byť štyri stavy:

1. Syntaktická chyba v zápise obmedzenia
2. Obmedzenie vyhodnotené ako splnené
3. Obmedzenie vyhodnotené ako nesplnené
4. Správa o tom, že obmedzenie nemôže byť vyhodnotené, lebo sa odkazuje na vlastnosti, ktorých doba viazania je neskoršia ako doba viazania konceptu.

Okrem toho sú vyhodnotené všetky sady obmedzení pre koncepty, ktoré sú referencované v danej inštancii.