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

Bc. Ľuboš Zelinka

Metamodel modelovania prípadov použitia a jeho profilovanie

Diplomová práca

Vedúci diplomovej práce:
Ing. Valentino Vranić, PhD.

máj, 2009

Anotácia

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

Autor: Bc. Ľuboš Zelinka
Diplomový projekt: Metamodel modelovania prípadov použitia a jeho profilovanie
Vedenie diplomového projektu: Ing. Valentino Vranić, PhD.
máj 2009

Prípady použitia sú často využívanou technikou na dokumentovanie požiadaviek. Používajú sa hlavne na zachytenie funkcionálnych požiadaviek softvérových systémov a modelovanie podnikových procesov. Napriek ich dôležitosti, podpora modelovania prípadov použitia v existujúcich nástrojoch nie je v mnohých prípadoch adekvátna. Cieľom tohto projektu bolo navrhnutie vhodného spôsobu zaznamenávania a udržiavania opisov prípadov použitia. Na základe analýzy opisov prípadov použitia, rôznych používaných notácií a podpory modelovania prípadov použitia v nástrojoch bol vytvorený metamodel pre modelovanie prípadov použitia. Výsledný metamodel je odvodený z UML konceptu pre modelovanie prípadov použitia a rozšírený o typické časti textových opisov prípadov použitia. Navyše môže byť konfigurovaný prostredníctvom množiny viac ako dvadsiatich nastavení, ktoré môžu byť použité na vytváranie profilov. Výsledky tejto práce boli implementované do prototypu modelovo orientovaného nástroja pre modelovanie prípadov použitia.

Annotation

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

Author: Bc. Ľuboš Zelinka
Diploma Thesis: Use case modeling metamodel and its profiling
Supervisor: Ing. Valentino Vranić, PhD.
2009, May

Use cases are a frequently used technique for documenting requirements. They are mainly used for capturing functional requirements of software systems and for business process modeling. Despite their importance, the support for use case modeling in existing tools is in many cases not adequate. The goal of this project was to design an appropriate solution for writing and maintaining use case description. Based on the analysis of use case descriptions, different used notations and tool support for use case modeling a metamodel for use case modeling was created. The resulting metamodel is derived from the UML concept for use case modeling and enhanced with typical parts of textual use case descriptions. In addition it can be configured using a set of more than twenty options, which can be used to create profiles. The results of this work were implemented into a model based use case modeling tool prototype.

Čestné prehlásenie

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím publikácií uvedených v zozname použitej literatúry.

Máj 2009

Poďakovanie

Touto cestou vyslovujem poďakovanie Ing. Valentinovi Vranićovi za pomoc, odborné vedenie, cenné rady a pripomienky pri vypracovaní mojej diplomovej práce.

Máj 2009

1 Úvod

Zaznamenávanie, pochopenie a reprezentácia požiadaviek zákazníka na systém je kritickou úlohou softvérového inžinierstva. Z veľkého množstva techník, ktoré boli navrhnuté za týmto účelom, sa dominantnou technikou používanou v praxi stalo modelovanie prípadov použitia. Používa sa hlavne na dokumentovanie funkcionálnych požiadaviek na systém, ktoré sú dôležité pre všetky zainteresované osoby a tvoria podstatnú časť špecifikácie požiadaviek.

Funkcionálne požiadavky na softvér vyjadrujú, čo by mal určitý softvérový systém robiť pre jeho používateľov. Často sa však tieto požiadavky javia abstraktné a nehmotné, hlavne pre softvérových vývojárov, čo môže spôsobiť viacero problémov [4]. Podľa viacerých štúdií The Standish Group vykonaných v rôznych obdobiach viac ako tretina softvérových projektov bolo predĺžených, prípadne zrušených, kvôli nedostatku záujmu zo strany používateľa a neúplným alebo meniacim sa požiadavkám a špecifikácii [5,6]. Práve riziko predĺženia či zrušenia projektu kvôli neúplným alebo meniacim sa požiadavkám je možné čiastočne zmenšiť vhodným systematickým dokumentovaním požiadaviek, ktoré umožňuje modelovanie prípadov použitia.

Modelovanie prípadov použitia sa stalo populárnym prostriedkom dokumentovania funkcionálnych požiadaviek na softvér z viacerých dôvodov. Asi najvýznamnejším dôvodom bol ľahko pochopiteľný opis požiadaviek v prirodzenom jazyku, pretože nie všetky osoby zainteresované do projektu by boli schopné porozumieť pseudokódu alebo diagramom. S týmto dôvodom úzko súvisí aj ďalší a to, že ľudia, ktorí požiadavky na softvér dokumentovali, nepotrebovali poznať žiaden špeciálny jazyk. Okrem opisu uznávaného zainteresovanými osobami, boli prípady použitia aj dobre sledovateľné. V každej fáze vývoja softvéru bolo možné zistiť, či vytváraný systém spĺňa požiadavky zákazníka. Posledným významným dôvodom bola skutočnosť, že vlastne ani neexistovali vhodné alternatívy. Ich problémom bola buď ťažká čitateľnosť pre zainteresované osoby alebo slabá, prípadne žiadna, sledovateľnosť.

Prípady použitia opisujú, ako by mal systém reagovať na požiadavky používateľov. Každý prípad použitia má nejaký účel a je užitočný pre aspoň jedného používateľa systému. Základom prípadu použitia je postupnosťou akcií, ktoré sa vykonávajú pri interakcii medzi používateľom a systémom. To však nie je všetko. Napríklad pri opise výberu peňazí z bankomatu je potrebné tiež uviesť, za akých podmienok môže bankomat používateľovi peniaze vydať. Čo sa však stane ak v bankomate nie sú peniaze? Jedna z podmienok pre výber peňazí je porušená a systém musí vedieť, ako v tejto situácii zareagovať. Používateľ by určite nebol rád, keby si bankomat nechal kartu, lebo systém nevie, čo má urobiť. Je teda zrejmé, že prípady použitia neopisujú iba postupnosť akcií, ktorá vedie k úspešnému výslednému stavu, ale aj alternatívne postupnosti akcií, ktoré sa vykonajú na základe splnenia či nesplnenia určitých podmienok. Hovoríme, že podmienky redukujú nejednoznačnosť prípadov použitia [7].

Prípady použitia nie sú však len technikou pre dokumentovanie požiadaviek, ale aj technikou softvérového inžinierstva pre riadenie celého životného cyklu vývoja softvéru [8]. Tento prístup k vývoju softvéru bol nazvaný vývoj riadený prípadmi použitia (use case driven development) a v súčasnosti sa intenzívne používa. Na začiatku životného cyklu sa špecifikujú požiadavky identifikovaním prípadov použitia, definuje sa terminológia a doplnkové požiadavky na systém, ktoré nie sú zachytené v modeli prípadov použitia. V procese analýzy a návrhu sa z modelu prípadov použitia vytvorí analytický a návrhový model. Počas implementácie sa návrhový model stane špecifikáciou implementácie a jednotlivé navrhnuté triedy sa implementujú. Nakoniec sa vo fáze testovania použijú testovacie prípady, vytvorené na základe prípadov použitia, na otestovanie jednotlivých funkcií softvéru.

1.1 Motivácia a ciele práce

Podpora vývoja riadeného požiadavkami a vytváranie modelov prípadov použitia sa stali súčasťou prakticky každého CASE nástroja. Tieto nástroje však neposkytujú takú podporu modelovania prípadov použitia, ktorá by bola úmerná dôležitosti dokumentovania funkcionálnych požiadaviek. S každou novou verziou týchto nástrojov sa vylepšuje grafické rozhranie, uľahčuje sa kreslenie diagramov, ale na podporu textovej časti, ktorá je podstatou prípadov použitia sa akosi zabúda.

Problém spočíva hlavne v tom, že model prípadov použitia v CASE nástrojoch je založený na modelovacom jazyku UML, ktorý v rámci koncept pre modelovanie prípadov použitia definuje iba prvky grafickej notácie modelu a podstatu prípadov použitia, jeho opis, skrýva v entite s názvom prípad použitia. Nejedná sa však o nedokonalosť UML, ale skôr o abstrakciu, keďže existuje viacero odlišných spôsobov vytvárania opisu prípadov použitia, ktoré sa intenzívne využívajú v praxi. Nepríjemným dôsledkom vyplývajúcim z tejto skutočnosti je to, že vývojári CASE nástrojov sú nútený implementovať vlastný spôsob vytárania opisu prípadov použitia, ktorý vo väčšine prípadov poskytuje slabú podporu jeho zaznamenávania a udržiavania.

Cieľom tejto práce je vytvorenie metamodelu modelovania prípadov použitia založeného vychádzajúceho zo štruktúrovaného textového opisu prípadov použitia, ktorý je v praxi často preferovaný. Za účelom dosiahnutia tohto cieľa je potrebné analyzovať rôzne notácie a šablóny pre textový opisu prípadov použitia ako aj ich existujúcu podporu v nástrojoch umožňujúcich modelovanie prípadov použitia. Výsledkom by mala byť identifikácia dôležitých častí textového opisu prípadov použitia, ich štruktúry a vzťahov medzi nimi, ktoré budú v metamodeli vystupovať ako entity. Ako základ modelu je možné použiť koncept pre modelovanie prípadov použitia prezentovaný v UML [11] práve kvôli abstraktnej povahe vzhľadom na model prípadov použitia opísaný prostredníctvom textu.

Hlavnou myšlienkou je navrhnutie metamodelu modelovania prípadov použitia pre opis v podobe štruktúrovaného textu tak, aby bol zovšeobecnením viacerých populárnych textových notácií, ktoré bude možné spätne reprezentovať prostredníctvom profilov tvorených súborom nastavení aplikovaných na vytvorený metamodel. Myšlienka zovšeobecnenia notácií vychádza zo skutočnosti, že notácie používané pri textovom opise prípadov použitia sú podobné, pretože vychádzajú z rovnakého základu.

1.2 Štruktúra dokumentu

Práca je rozdelená na deväť častí. V prvej časti je úvod do problematiky a motivácia. V druhej časti je vysvetlený pojem a jeho využitie pri vývoji softvéru a modelovaní podnikových procesov. Tretia časť obsahuje pohľad na základné časti opisu prípadov použitia, ktoré sa používajú v praxi. Štvrtá časť je venovaná analýze existujúcej podpory pre modelovanie prípadov použitia. V piatej časti je uvedený návrh metamodelu modelovania prípadov použitia na základe existujúcich prístupov. Šiesta časť sa zaoberá možnosťami profilovania navrhnutého metamodelu modelovania prípadov použitia. V siedmej časti sa nachádza návrh nástroja pre modelovanie prípadov použitia založený na navrhnutom metamodeli a jeho možnostiach profilovania. Ôsma časť obsahuje opis dôležitých častí implementácie prototypu a záverečná časť s číslom deväť obsahuje zhodnotenie problematiky a dosiahnutých výsledkov práce.

2 Využitie prípadov použitia

V súčasnosti sú prípady použitia najčastejšie spájané UML modelmi. Veľa ľudí si pod prípadmi použitia ihneď predstaví diagram s elipsami, postavičkami a rôznymi šípkami, ktoré znázorňujú určité vzťahy. Prípady použitia však vznikli skôr ako samotné UML a boli opisované prevažne textovou formou. Vznik prípadov použitia sa spája s menom Ivar Jacobson, ktorý ich vymyslel v roku 1967, no vtedy ešte termín prípad použitia neexistoval [1]. Verejnosti predstavil prípady použitia až v roku 1987 [2]. V roku 1992 predstavil Ivar Jacobson základné koncepty modelovania prípadov použitia [3]. V tom čase sa už forma prípadov použitia dosť podobala na tú, ktorá sa používa dnes [1].

Prípad použitia opisuje postupnosť akcií vykonávaných systémom, ktorých výsledok poskytuje určitému účastníkovi nejakú hodnotu. Postupnosť akcií je vždy iniciovaná účastníkom, ktorý má určitý cieľ a ten je uspokojený práve vtedy, keď postupnosť akcií skončí úspechom. Samozrejme to neznamená, že prípady použitia opisujú iba postupnosť akcii, ktorá vedie k úspešnému koncu. Často systém z rôznych dôvodov zlyhá pri plnení cieľa účastníka a je potrebné, aby systém vedel takéto situácie riešiť. Preto opis prípadov použitia obsahuje napríklad ošetrenie chýb alebo opis špeciálneho chovania sa systému. Práve sústredenie sa na zlyhanie cieľov a odpovede na zlyhanie sú dva dôvody, prečo sú prípady použitia vhodné na opis systémov [9].

2.1 Systémový prípad použitia

Asi najväčšou výzvou pre vývojárov je zdieľanie rovnakej vízie o výslednom produkte so zákazníkom. Vízia býva vyjadrená vo forme požiadaviek zákazníka na výsledný produkt. Požiadavky sú vlastnosti, schopnosti a správanie, ktoré by mal systém mať. Rozdeľujú sa na funkcionálne a nefunkcionálne podľa toho, či ide o funkcie výsledného systému alebo jeho vlastnosti. Na pochopenie problému a dosiahnutia dohody so zákazníkom je potrebné všetky požiadavky na systém vhodne dokumentovať a aj udržiavať, pretože sa môžu meniť. Pomocou prípadov použitia je možné zdokumentovať funkcionálne požiadavky na systém, lebo funkcionálne požiadavky charakterizujú správanie výsledného produktu, ktoré je možné opísať postupnosťou akcií. Takýmto prípadom použitia nesú prívlastok systémové.

Systémové prípady použitia sú prípady použitia vytváraného softvérového systému a postupnosť akcií opisuje interakciu medzi účastníkom a vytváraným systémom. Účastníkom môže byť používateľ systému, iný systém alebo externá udalosť, ako napríklad dosiahnutie určitého času. Systémové prípady použitia nie sú funkcie napriek tomu, že dokumentujú funkcionálne požiadavky na systém, ale skôr opisujú ako účastník môže použiť systém, aby dosiahol niečo užitočné. Pri ich dokumentovaní sa zvyčajne nepoužíva implementačno-špecifický jazyk, aby sa zabránilo akýmkoľvek predstavám alebo domnienkam o realizácii prípadov použitia.

Opis systémových prípadov použitia býva predovšetkým zapísaný v prirodzenom jazyku s použitím terminológie danej oblasti napriek tomu, aby ho pochopili všetci zainteresovaní. Môže mať rôzne podoby z hľadiska obsahu aj formálnosti. Obsah systémového prípadu použitia pozostáva hlavne z názvu, účastníkov a scenárov. Okrem toho obsahuje často vstupné a výstupné podmienky, prípadne cieľ, spúšťače a rôzne ďalšie informácie. Všetky časti môžu byť zapísané ako neštruktúrovaný text, no taký druh opisu je moc neprehľadný a často sa nepoužíva. Namiesto toho sa používa štruktúrovaný opis rozdelený na spomínané časti.

2.2 Biznis prípad použitia

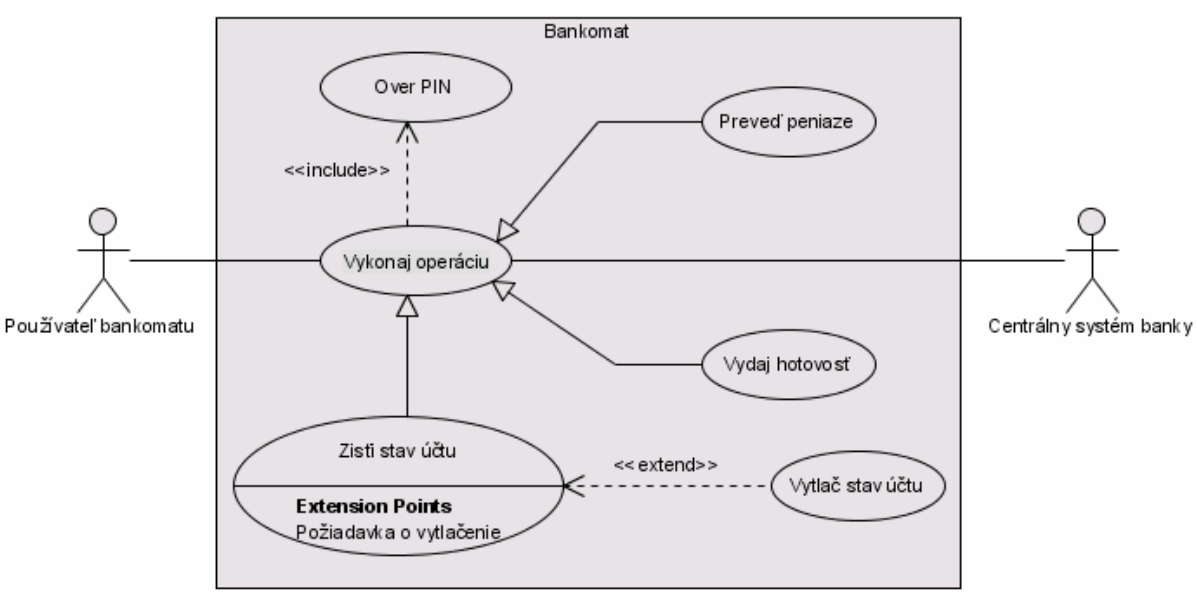
Prípady použitia vznikli pre potreby dokumentovania funkcionálnych požiadaviek vytváraného systému. Neskôr sa však do istej miery uplatnili aj pri dokumentovaní podnikových procesov, ktoré reprezentujú štruktúru a hlavne správanie sa organizácie. Štruktúra je definovaná entitami a reláciami medzi nimi, zatiaľ čo správanie pozostáva z procesov, udalostí a pravidiel. Cieľom modelovania podnikových procesov je ich vylepšenie a zautomatizovanie. V biznis modeli založenom na prípadoch je štruktúra vyjadrená pomocou účastníkov a správanie pomocou samotných prípadov použitia, ktoré sa nazývajú biznis prípady použitia.

Biznis prípad použitia opisuje biznis procesy, ktoré dokumentuje ako postupnosť akcií poskytujúcu nejakú hodnotu biznis účastníkovi [10]. Postupnosť akcií biznis prípadu použitia reprezentuje automatizáciu podnikových procesov ako celku alebo ich častí, počas ktorej sú dokumenty, informácie alebo úlohy presúvané od jedného účastníka k druhému pre ďalšie spracovanie zodpovedajúce definovanej organizačnej štruktúre. Biznis účastníkom môže byť zákazník, dodávateľ, partner alebo iná zainteresovaná osoba, ktorá je v interakcii s organizáciou. Každý účastník v interakcii s organizáciou má určité ciele, ktoré chce interakciou s organizáciou dosiahnuť. Na základe týchto cieľov je možné identifikovať prípady použitia, pričom je možné pozerať sa na organizáciu ako na čiernu skrinku. Netreba sa teda zaujímať ako organizácia reaguje na požiadavky účastníkov, ale iba o to, že reaguje správne.

Opis biznis prípadov použitia je veľmi podobný opisu systémových prípadov použitia. Je zvyčajne štruktúrovaný a prakticky celú štruktúru textového opisu systémových prípadov použitia je možné aplikovať aj pri opise biznis prípadov použitia. Jediné rozdiely sú v samotnom obsahu a to hlavne pri opise účastníkoch a scenároch. Účastníkom v biznis prípadoch použitia spravidla nemôže byť systém. Systém môže byť účastníkom iba na úrovni modelovania vytváraného systému a preto keď ho chceme uviesť v biznis modeli, musíme v ňom ako hráča uviesť organizáciu, ktorej patrí. Scenáre, ako bolo už spomenuté, opisujú biznis procesy namiesto interakcií a akciami sú teda aktivity procesu definovaného biznis prípadom použitia.

3 Reprezentácia a podpora prípadov použitia

Problém podpory prípadov použitia existuje prakticky od zaužívania samotných prípadov použitia v praxi. Mnohé dnešné nástroje využívajú na modelovanie prípadov použitia diagram prípadov použitia, ktorý je súčasťou UML, a sústreďujú sa skôr na podporu kreslenia týchto diagramov ako na zaznamenávanie a údržbu textovej časti jednotlivých prípadov použitia. Preskúmaním príkladu diagramu prípadov použitia (pozri *Obr. 1*) je možné prísť k záveru, že diagramami nemožno úplne opísať prípady použitia. Z diagramu sa dajú vyčítať jednotlivé prípady použitia a systém, ku ktorému sa vzťahujú, vzťahy medzi prípadmi použitia, body rozšírenia prípadov použitia a zainteresovaných. Nemožno však vyčítať, čo sa má presne vykonať a za akých podmienok.



Obr. 1: Model prípadov použitia pre bankomat

Textom je možné opísať nielen to, čo zobrazuje diagram prípadov použitia, ale aj to, čo zobraziť nedokáže. Najdôležitejšou časťou textového opisu prípadu použitia, ktorá sa nedá modelovať diagramom prípadov použitia, je správania systému. Textový opis prípadov použitia môže teda existovať aj bez diagramu prípadov použitia, avšak diagram bez textového opisu prípadov použitia existovať nemôže. Znamená to, že diagramy prípadov použitia sú nepotrebné? Názory na túto otázku sú rôzne. Spôsobené je to hlavne vzťahmi medzi prípadmi použitia, ktoré ešte stále pôsobia zmätočne a často prinášajú viac škody ako úžitku. Z tohto dôvodu sa niektorí metodológovia zaoberajúci sa prípadmi použitia, ako napríklad Alistair Cockburn [9], bránia používaniu diagramu prípadov použitia. Napriek tomu diagram prípadov použitia prehľadne zobrazuje zovšeobecnené funkcionálne požiadavky na systém alebo jeho časti, čiže určite nie je úplne zbytočný. Okrem toho môže kreslenie diagramu slúžiť ako pomôcka pri identifikovaní zainteresovaných, prípadov použitia, vzťahov medzi prípadmi použitia a zainteresovanými a predmetu jednotlivých prípadov použitia.

Hlavným prostriedkom pre úplné opísanie prípadov použitia je z uvedených dôvodov text. V nástrojoch na modelovanie prípadov použitia sa často stáva, že jedinou podporou textovej časti prípadov použitia je iba textové pole nazývané opis prípadu použitia, kde si autor prípadu použitia môže napísať vlastný text. Tento prístup sa môže na prvý pohľad zdať v poriadku, veď predsa

softvérová firma si môže stanoviť vlastný formát opisu, ktorý by autori prípadov použitia mali dodržiavať, no písane a údržba takého opisu by boli časovo náročné.

Textový opis prípadov použitia môže byť formálny alebo neformálny. Neformálny opis je obyčajný text bez číslovania, ktorý môže byť ďalej rozdelený na toky udalostí. Využíva sa najčastejšie v agilnom modelovaní. Formálny opis býva oproti neformálnemu číslovaný, štruktúrovaný a prostredníctvom neho možno detailnejšie zachytiť funkčné požiadavky na vyvíjaný softvér. Navyše notácie používané vo formálnych opisoch prípadov použitia možno zachytiť v tvare metamodelu a preto budú v tejto časti práce ďalej analyzované.

Štruktúra opisu prípadov použitia má viac ako päťdesiatročnú históriu [2]. V šesťdesiatych rokoch sa však ešte nehovorilo o prípadoch použitia a písali sa iba rôzne neformálne scenáre použitia, ktoré poskytovali základnú myšlienku fungovania vyvíjaného systému. Práve Ivar Jacobson bol tým človekom, ktorý začal písať tieto scenáre použitia pri definovaní architektúry systému AXE vyvíjaného firmou Ericsson [2]. Od tých čias sa štruktúra opisu neustále menila. Na začiatku deväťdesiatych rokov bol súčasťou opisu napríklad účel prípadu použitia, tok riadenia, základný tok udalostí, alternatívne toky udalostí, podmienky a účastníci. Neskôr pribudli k opisu zainteresovaní, ich záujmy a ciele, čím vznikol základ štruktúry opisu prípadov použitia tak, ako sa používa dnes.

3.1 Názov prípadu použitia

Názov prípadu použitia je dôležitou súčasťou opisu prípadu použitia. Vhodný názov napovedá, čo sa dosiahne interakciou medzi účastníkom a systémom a preto je treba názov vybrať opatrne. Názov prípadu použitia by mal byť krátky, výstižný, unikátny a predovšetkým aktívny. Aktívny názov je napríklad vyber hotovosť a implikuje nejakú akciu, zatiaľ čo pasívny názov, ako napríklad výber hotovosti, môže znieť ako funkcia alebo funkčná oblasť v rámci organizácie. Pasívne názvy sa využívajú skôr na pomenovanie skupiny prípadov použitia.

3.2 Stručný opis

Názov prípadu použitia, napriek tomu, že je výstižný, neposkytne čitateľovi základný opis prípadu použitia. Na tento účel slúži stručný opis (brief description), ktorý sa občas nazýva cieľ prípadu použitia. Tento opis hovorí o účele prípadu použitia a jeho hodnote pre účastníkov a zainteresované osoby, čím vlastne odôvodňuje jeho existenciu [7]. To znamená, že jeho obsahom by mali byť zainteresované osoby, poskytovaná hodnota a krátky opis, ako systém túto hodnotu dosiahne. Podobne ako názov, aj stručný opis prípadu použitia musí byť jasný a jednoznačný.

3.3 Predmet

Pod predmetom (subject) prípadu použitia sa dá rozumieť navrhovaný systém, na ktorý sa prípad použitia vzťahuje [11]. Pri určovaní predmetu návrhu je veľmi dôležité, aby sa na ňom zhodli všetky zainteresované osoby. Problém spočíva v tom, že predmet návrhu v modeli prípadov použitia určujú tí ľudia, ktorí ho vytvárajú a ostatné zainteresované osoby musia vedieť predmet návrhu jednoznačne prečítať. Riešenie tohto problému spočíva v zaradení predmetu návrhu do opisu každého prípadu použitia.

Predmet návrhu sa v diagrame prípadov použitia označuje ako štvoruholník, v ktorom sa nachádza všetko, čo do daného predmetu návrhu patrí. Pohľadom na diagram prípadov použitia pre bankomat (pozri *Obr. 1*) je možné vidieť, že všetky prípady použitia v tomto diagrame majú rovnaký predmet a to bankomat. Keby bol bankomat modelovaný ako podsystém nejakého väčšieho systém, stále by

bol predmetom všetkých prípadov použitia v spomínanom diagrame bankomat, lebo predmet musí byť čo najšpecifickejší a musí byť iba jeden kvôli tomu, že musí byť jednoznačný.

3.4 Zainteresovaní a účastníci

Zainteresovaný (stakeholder) je subjekt, ktorý sa určitým spôsobom zaujíma o správanie sa prípadu použitia, ale nemusí byť v interakcii s daným systémom. Týmto subjektom môže byť napríklad majiteľ systému, oddelenie manažmentu kvality a podobne. Uvádzanie zainteresovaných a ich záujmov v opise prípadov použitia môže viesť k zlepšeniu ich kvality, pretože záujmy zainteresovaných predstavujú biznis pravidlá, ktoré musí výsledný systém dodržiavať. Zlepšenie kvality opisu prípadu použitia spočíva v zahrnutí biznis pravidiel vyplývajúcich zo záujmov zainteresovaných do opisu prípadu použitia.

Účastníci (actors) sú externé entity, ktoré sú v interakcii so systémom [4]. Účastníkom môže byť buď používateľ systému, prípadne iný systém alebo podsystém. Dôležitým znakom účastníkov je to, že využívajú funkcie a služby modelovaného systému. Každý účastník môže byť spojený s viacerými prípadmi použitia systému a každý prípad použitia môže byť spojený s viacerými účastníkmi. Účastníka, pre ktorého systém vyvíjame, nazývame hlavný účastník (primary actor). Hlavný účastník využíva systém na dosiahnutie svojich cieľov. Často je teda subjektom, ktorý aktivuje príslušný prípad použitia napríklad stlačením klávesy, poslaním správy alebo iným spôsobom interakcie so systémom. Niekedy je však potrebuje systém na dosiahnutie cieľov hlavných účastníkov iných účastníkov, ktorých nazývame vedľajší účastníci (secondary actor). Vedľajší účastníci sú externí účastníci, ktorí poskytujú služby danému systému. Vedľajším účastníkom môže byť napríklad tlačiareň, webová služba, telefonický operátor a podobne.

Pri opise prípadu použitia nie je prakticky možné vymenovať účastníkov na základe ich skutočných mien. Účastníci však majú často spoločné ciele, na základe ktorých je možné účastníkov agregovať do rolí, ktoré zastávajú z pohľadu systému ako napríklad administrátor, manažér alebo zákazník. Opis účastníkov resp. rolí je veľmi jednoduchý a väčšinou sa skladá z mena a profilu. Meno je charakteristický názov účastníka, zvyčajne podstatné meno, ktoré nemusí byť spojené so skutočným menom účastníka. Profil je krátky textový opis, ktorý charakterizuje účastníka. Môže obsahovať napríklad zručnosti používateľa systému. Pomáha hlavne návrhárom, ktorí na základe neho môžu prispôsobiť správanie sa systému alebo grafické používateľské rozhranie pre daných účastníkov.

V diagrame prípadov použitia sa zobrazujú účastníci ako pomenované postavičky. Je to viac menej zaužívané zobrazenie UML stereotypu ‹‹Actor››. V diagrame prípadov použitia pre bankomat (pozri *Obr. 1*) je možné vidieť, že účastníkmi v tomto systéme sú používateľ bankomatu a centrálny systém banky. To, čo ale nemožno nevidieť, je profil a dokonca ani typ účastníka, teda či ide o hlavného účastníka alebo vedľajšieho. Profil sa teoreticky dá zobraziť v diagrame pomocou poznámky, avšak takým spôsobom vytvorený diagram by vyzeral neprehľadne a je zrejmé, že takáto možnosť je prakticky nepoužiteľná. Problém typov účastníkov má v diagrame prípadov použitia riešenie a to pomocou orientovaných asociácií.

3.5 Vzťah medzi účastníkmi a prípadmi použitia

Skutočnosť, že sa účastník zúčastňuje na prípade použitia sa v diagrame prípadov použitia označuje asociáciou medzi účastníkom a prípadom použitia. Asociácia sa v UML značí čiarou a indikuje špecifickú interakciu medzi účastníkom sa systémom. Sieť všetkých asociácií medzi účastníkmi a prípadmi použitia v rámci určitého systému poskytuje obraz o interakcii systému s jednotlivými

účastníkmi. Interakciou medzi systémom a účastníkmi je väčšinou komunikácia a preto sa týmto asociáciám hovorí aj komunikačné asociácie.

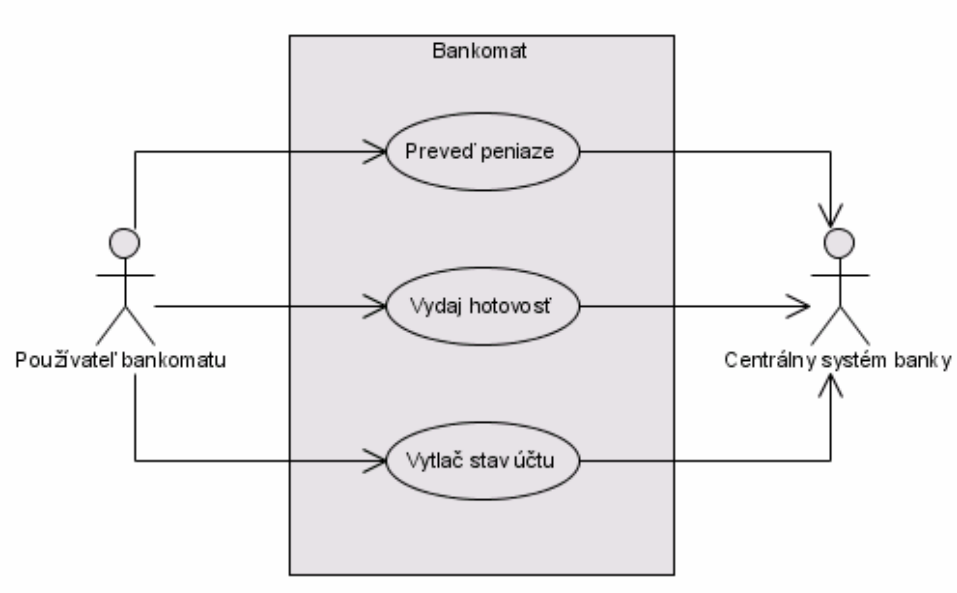
Komunikačná asociácia reprezentuje dialóg medzi účastníkom a systémom [7]. Asociácia však neurčuje tok dát komunikácie, čas komunikácie alebo spôsob komunikácie, ale naznačuje iba to, že nejaká interakcia účastníka so systémom môže nastať a často aj nastane. Teoreticky by sa mohlo stať, že interakcia účastníka a systému nikdy nenastane, napríklad keď bankomat nikto nikdy nepoužije.

Komunikácia medzi účastníkom a systémom je väčšinou obojstranná. Účastník aj systém majú pre komunikáciu viacero dôvodov [7]:

Tab. 1: Dôvody komunikácie účastníka a systému

Účastník	Systém
Inicializovanie prípadu použitia.	Oznámenie niečoho, o čom by mal účastník vedieť.
Získanie údajov zo systému.	Spýtanie sa účastníka pri dôležitých rozhodnutiach
Zmena údajov v systéme.	Postúpenie zodpovednosti používateľovi
Oznámenie niečoho, o čom by mal systém vedieť.	

Hlavný aj vedľajší účastník nie sú statické typy účastníkov vzhľadom na systém, s ktorým sú účastníci v interakcii, ale závisia na prípade použitia. To znamená, že ku každému prípadu použitia nestačí pripojiť asociáciami účastníkov, ale treba uviesť aj typ každého účastníka spojeného s určitým prípadom. Z obrázka, na ktorom je diagram pre bankomat objednávok (pozri *Obr. 1*), určenie typov účastníkov pre každý prípad použitia nie je zrejmé. Notácia pre diagram prípadov použitia však umožňuje tento problém vyriešiť pomocou orientovaných asociácií.



Obr. 2: Zjednodušený model prípadov použitia pre bankomat

Na obrázku 2 je naznačený veľmi zjednodušený diagram prípadov použitia pre operácie bankomatu. Tento diagram explicitne určuje typy účastníkov pre jednotlivé prípady použitia pomocou orientácie jednotlivých asociácií. Asociácia, ktorá začína pri účastníkovi naznačuje, že tento účastník je pre pripojený prípad použitia hlavný účastník. To znamená, že používateľ

bankomatu je hlavný účastník pre všetky tri prípady použitia a centrálny systém banky je vedľajší účastník taktiež pre všetky tri prípady použitia.

Na prvý pohľad by sa mohlo zdať, že šípky v diagrame naznačujú nejaký tok údajov resp. smer komunikácie. Samozrejme to nie je pravda, čo je jasné pri pohľade na skutočné fungovanie bankomatu. Napríklad keď pri prihlasovaní zadá používateľ PIN, tak dostane od systému odpoveď či je PIN správny alebo nie. PIN overí centrálny systém banky na základe požiadavky bankomatu a odošle mu odpoveď, ktorú bankomat zasa sprostredkuje používateľovi. Je teda zrejmé, že používateľ bankomatu vystupuje v rámci tohto prípadu použitia v roli hlavného účastníka a centrálny systém banky v roli vedľajšieho účastníka. Napriek tomu komunikácia medzi systémom a účastníkmi prebieha v oboch smeroch. V UML je použitie orientovaných asociácií voliteľné. Takáto notácia bola vymyslená kvôli ujasneniu typov účastníkov priamo v diagrame, no bez poznania jej interpretácie nie je táto notácia až taká jasná.

UML notácia pre diagram prípadov použitia umožňuje aj určenie násobnosti asociácií podobne ako napríklad pri diagrame tried. Násobnosť asociácie hovorí, s koľkými inštanciami prípadu použitia môže jedna inštancia účastníka komunikovať súčasne a naopak [12]. Inštancia prípadu použitia je jedno konkrétne použitie opísané v prípade použitia, vykonaním jedného cesty akcií cez všetky sekvencie akcií opísaných v prípade použitia [12]. Pre prípad použitia prihlásenie z diagramu prípadov použitia pre bankomat by násobnosť asociácií vyzerala nasledovne:

Obr. 3: Násobnosť asociácií v diagrame prípadov použitia

Asociáciu možno zapísať aj pomocou textu v opise prípadu použitia, keďže tento vzťah medzi účastníkmi a prípadmi použitia existoval predtým ako vzniklo samotné UML. Zápis je veľmi jednoduchý a ľahko pochopiteľný, pretože je založený na vymenovaní hlavných a vedľajších účastníkov. Je prakticky ekvivalentný použitiu orientovaných asociácii, ale nehovorí nič o násobnosti. Násobnosť sa však väčšinou prejaví v ďalšom opise prípadu použitia a preto ju netreba explicitne zapisovať.

3.6 Vstupné podmienky

Vstupné podmienky (preconditions) prípadu použitia opisujú stav systému, pri ktorom sa prípad použitia jednoducho povedané naštartuje. Ich cieľom je nastavenie kontextu pre čitateľa opisu prípadu použitia [12]. Vstupná podmienka nie je opis nejakej udalosti, ale skôr výrok, ktorý hovorí o tom, kedy je možné prípad použitia aplikovať. Z tohto dôvodu vstupná podmienka nestačí na spustenie prípadu použitia a prípad použitia musí byť iniciovaný hlavným účastníkom, ktorý môže prípad použitia spustiť iba vtedy, keď sú všetky vstupné podmienky pre daný prípad použitia splnené.

Výraz vstupná podmienka môže vzbudiť dojem, že systém bude túto podmienku kontrolovať pred spustením prípadu použitia. Stav systému je vo väčšine prípadov výsledkom vykonaných prípadov použitia, čiže vykonané prípady použitia nastavujú vstupné podmienky pre ďalšie prípady použitia. Všetky akcie systému vrátane kontroly podmienok musia byť zahrnuté v tokoch akcií prípadov použitia.

3.7 Výstupné podmienky

Výstupné podmienky (postconditions) sú podobné vstupným podmienkam, ale hovoria čitateľovi o stave, v ktorom sa systém bude nachádzať po skončení prípadu použitia [12]. Dokopy hovoria vstupné a výstupné podmienky o tom, čo prípad použitia robí bez toho, aby hovorili ako. Preto tvoria vhodný základ pre vytváranie testov pomocou techniky čiernej skrinky.

Výstupné podmienky opisujú všetky koncové stavy prípadu použitia, vrátane chybných stavov, zatiaľ čo vstupné podmienky opisujú stav, v ktorom všetky toky udalostí môžu byť iniciované. Z toho vyplýva, že na začiatku vykonávania prípadu použitia bude splnená aspoň jedna vstupná podmienka a na konci vykonávania prípadu použitia bude splnená aspoň jedna výstupná podmienka. Vzťah medzi nimi potom možno opísať aj tak, že ak je splnená vstupná podmienka, potom po vykonaní prípadu použitia bude splnená výstupná podmienka. Dôležitou vlastnosťou výstupných podmienok je teda nezávislosť od zvoleného alternatívneho toku udalostí. Samozrejme vytvorenie takýchto podmienok nemusí byť a často ani nie je triviálne, hlavne keď má prípad použitia viacero alternatívnych tokov udalostí.

Pri vstupných podmienkach bolo povedané, že stav je vo väčšine prípadov výsledkom vykonaných prípadov použitia a práve stav vykonaných prípadov použitia je charakterizovaný výstupnými podmienkami. To znamená, že pomocou vstupných a výstupných podmienok je možné vytvoriť postupnosti prípadov použitia. Napríklad v diagrame prípadov použitia pre bankomat je zrejmé, že peniaze môže používateľ vybrať až potom, čo sa prihlási resp. zadá správny PIN a práve túto skutočnosť majú odrážať vstupné a výstupné podmienky.

3.8 Scenáre a toky udalostí

Scenáre (scenarios) opisujú každý možný výsledok pokusu dosiahnuť cieľ prípadu použitia [13]. Každý scenár je postupnosťou interakcií medzi účastníkom a systémom. Interakcia začína nejakou spúšťacou akciou a pokračuje pokým nie je cieľ dosiahnutý alebo opustený [14]. Z toho vyplýva, že scenáre neslúžia len na opis postupnosti akcií, ktoré vedú k dosiahnutiu cieľa, ale aj takých, kde dosiahnutie cieľa z rôznych dôvodov zlyhá. Dokumentovanie scenárov je užitočné z viacerých dôvodov [7]:

- Scenáre budú zodpovedať testovacím prípadom a budú dôležitým zdrojom informácii pre testovanie systému.
- Scenáre hovoria o tom, ako bude systém fungovať v praxi a teda aj ako bude systém používaný, čo ich robí užitočnými pre vytváranie dokumentácie.
- Scenáre sú užitočné v procese analýzy a návrhu, lebo pomáhajú vývojárom rozmýšľať o tom, ako bude systém používaný.

Scenár je inštancia prípadu použitia, ktorá reprezentuje jednu logickú cestu realizácie alebo vykonávania konceptuálneho prípadu použitia [13]. Interakcie medzi účastníkom a systémom z hľadiska štruktúry textového opisu prípadu použitia sa nazývajú kroky. Kroky by mali opisovať jednoduché akcie a jasne identifikovať účastníka alebo systém, ktorý akciu vykonáva. Každý krok môže okrem interakcie opisovať aj validáciu za účelom ochrany záujmov zainteresovaných a vnútornú zmenu na uspokojenie zainteresovaných [9].

Postupnosti krokov vytvárajú toky udalostí, aby bol opis prípadu použitia lepšie pochopiteľný. Rozdiel medzi tokom udalostí a scenárom však spočíva v tom, že tok udalostí nemusí opísať celé vykonávanie prípadu použitia od začiatku do konca. Toky udalostí možno reprezentovať napríklad neformálnym textom, diagramom činností alebo štruktúrovaným textom, ktorý je často preferovaný, pretože je dobre čitateľný a ľahko pochopiteľný, ak je správne napísaný. V UML je

tok udalostí braný ako jedna vlastnosť a prípad použitia hovorí málo o tom, ako by mal byť štruktúrovaný [7]. Toky udalostí sa bežne rozdeľujú na základné a alternatívne.

3.8.1 Základný tok udalostí

Základný tok udalostí (basic flow of events) opisuje normálnu postupnosť krokov pri vykonávaní prípadu použitia [7]. Je najdôležitejšou a nevyhnutnou časťou opisu prípadu použitia, pretože reprezentuje interakcie medzi účastníkom a systémom za ideálnych podmienok, teda bez alternatív, chýb a výnimiek. Každý opis prípadu použitia by mal obsahovať minimálne jeden takýto typický scenár, lebo reprezentuje nejakú základnú funkcionalitu systému. Prípad použitia však môže obsahovať viacero základných tokov udalostí, keď je možné prípad použitia iniciovať rôznymi spôsobmi [8]. Problém je však v tom, že prípad použitia má určitý cieľ pričom hlavný tok je prostriedkom jeho dosiahnutia a teda existenciou viacerých hlavných tokov nemusí byť pravý cieľ prípadu použitia zrejmý, pokiaľ nie je explicitne uvedený. Metodológovia zaoberajúci sa prípadmi použitia vo všeobecnosti radia vyhnúť sa prípadom použitia s viacerými hlavnými tokmi, pretože sa tým ohrozuje ich pochopiteľnosť.

Keď sa píšu scenáre prípadu použitia, mali by vždy začínať základným tokmi, lebo ich možno ľahko identifikovať. V nich sa potom hľadajú miesta, kde môže prípad použitia zlyhať alebo kde existuje nejaká alternatíva, ktorá môže viesť k úspešnému koncu. Tie sa potom rozvinú do ďalších tokov udalostí tak, aby všetky nájdené toky udalostí vrátane toho základného tvorili úplný opis prípadu použitia z hľadiska scenárov. Aby sa nemuseli vypisovať kroky alternatívnych tokov úplne od začiatku a tým aj niekoľko krát opakovať niektoré kroky, je vhodné jednotlivé kroky základného toku udalostí číslovať. Potom stačí označiť miesto, kde alternatívny tok začína a kde končí a vypísať jeho kroky.

3.8.2 Alternatívny tok udalostí

Základný tok udalostí je len jeden z možných scenárov prípadu použitia. Zvyšné toky sa nazývajú alternatívne toky udalostí (alternative flows of events). Alternatívny tok udalostí zahŕňa voliteľné, výnimočné a alternatívne správanie vo vzťahu k iným tokom udalostí [7]. Je vždy závislý na podmienke kontrolovanej v inom toku udalostí, inak by nemohol byť alternatívny.

Prvý krok alternatívneho toku udalostí určuje miesto a podmienku aktivácie tohto alternatívneho toku udalostí. Často sa stáva, že v jednom mieste existujú viaceré alternatívy. Tie sa však líšia v podmienke aktivácie, na základe ktorej sa potom vyberie alternatíva. Niektorý krok alternatívneho toku udalostí zasa môže určiť miesto, kde bude účastník pokračovať v toku udalostí. Miestom pokračovania môže byť miesto, v ktorom bol tok udalostí spustený alebo iné miesto v toku udalostí, pokiaľ prípad použitia nekončí [7]. Podmienka aktivácie sa zvyčajne označuje číslom miesta základného toku udalostí, ku ktorému sa pridáva písmeno „a“ ako alternatíva. Kroky alternatívneho toku sa potom číslujú pridaním čísiel k existujúcemu číslu kroku.

Špeciálnym typom alternatívneho toku udalostí je taký, kde podmienkou aktivácie je nejaký výnimočný stav. Tento typ alternatívneho toku udalostí je najčastejšie sa vyskytujúci alternatívny tok, lebo opisuje ošetrenie chyby. Nazýva sa aj chybový tok udalostí (exception flow of events). Chybové toky udalostí sa často oddeľujú od ostatných alternatívnych tokov, aby sa zlepšila prehľadnosť opisu prípadu použitia.

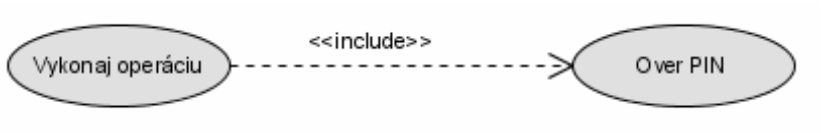
3.9 Scenáre a vzťahy medzi prípadmi použitia

Keď sa modeluje systém pomocou prípadov použitia často sa stáva, že rovnaké postupnosti krokov sa vyskytujú v rôznych prípadoch použitia počas identifikácie scenárov a bolo by vhodné zhrnúť

ich do jedného prípadu použitia, aby sa odstránila redundanciu. Inokedy sa dajú nájsť alternatívne toky udalostí, ktoré by mohli byť zapísané ako samostatné prípady použitia alebo je výhodné pridať novú funkcionalitu na uspokojenie nových alebo existujúcich potrieb používateľa bez toho, aby sa príliš zasahovalo do existujúceho modelu prípadov použitia. Pre obe tieto situácie existuje riešenie, ktoré však zahŕňa využitie vzťahov medzi prípadmi použitia. UML špecifikuje tri vzťahy medzi prípadmi použitia: zahrnutie, rozšírenie, zovšeobecnenie. Každý z nich má ekvivalentnú reprezentáciu v textovom opise prípadu použitia.

3.9.1 Vzťah zahrnutia

Špecifikácia UML definuje vzťah zahrnutia (include relationship) a vzťah medzi prípadmi použitia, kde jeden z nich obsahuje správanie definované v inom prípade použitia [11]. Vzťah zhrnutia je analogický volaniu medzi objektmi v programe. Slúži na znovu používanie existujúcich prípadov použitia alebo na extrakciu rovnakých častí opisu prípadov použitia. V oboch prípadoch ide o zjednodušenie opisu prípadov použitia, aby bolo možné ľahšie udržiavať model prípadov použitia. Správanie zahrnutého prípadu použitia sa vkladá do správania daného prípadu použitia. Keďže hlavným účelom tohto vzťahu je znovu používanie spoločných častí, to čo zostane v základnom prípade použitia, nie je samostatný prípad použitia a závisí na zahrnutých prípadoch použitia. Jednou z najčastejších chýb je používanie tohto vzťahu na funkčnú dekompozíciu.



Obr. 4: UML notácia vzťahu zahrnutia

Vzťah zahrnutia sa v UML značí pomocou šípky so stereotypom ‹‹include›› smerujúcej k zahrnutému prípadu použitia (pozri *Obr. 4*). V textovom opise prípadu použitia sa zahrnutie prípadu použitia prejavuje tak, že niektorý krok obsahuje názov zahrnutého prípadu použitia.

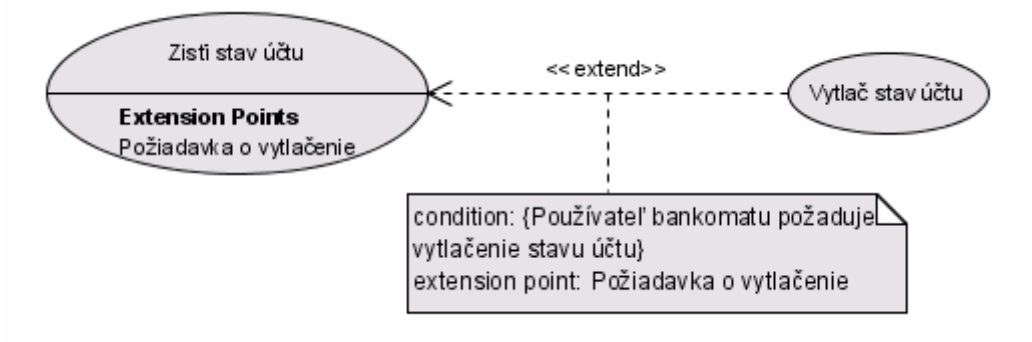
3.9.2 Vzťah rozšírenia

Podľa špecifikácie UML vzťah rozšírenia (extend relationship) špecifikuje ako a kedy môže byť správanie sa definované v rozširujúcom prípade použitia vložené do správania definovaného v rozširovanom prípade použitia [11]. Jednoducho povedané, pomocou vzťahu rozšírenia je možné rozšíriť správanie systému opísané v určitom prípade použitia. Rozširovaný prípad použitia musí byť nezávislý na prípade použitia, ktoré rozširujú ním opisované správanie sa systému. To znamená, že rozširovaný prípad použitia musí byť úplný aj bez rozširujúcich prípadov použitia. Týmto sa vzťah rozšírenia odlišuje od vzťahu zahrnutia.

Pomocou vzťahu rozšírenia možno modelovať voliteľné alebo výnimočné správanie podobne ako pomocou alternatívnych tokov udalostí. Ako bolo už spomenuté, niektoré alternatívne toky sa dajú zapísať ako samostatné prípady použitia a teda budú pôsobiť ako rozširujúce prípady použitia. Ostatné alternatívne toky sa takto nedajú zapísať hlavne preto, že sú závislé na podmienkach, tokoch udalostí alebo stave systému opísanom v príslušnom prípade použitia. Na druhej strane zapísanie takýchto alternatív ako rozširujúcich prípadov použitia nie je ani žiadané. Problémom by bolo príliš veľa prípadov použitia s krátkymi opismi a navzájom redundantnými časťami, ktoré by navyše vytvorili veľmi neprehľadný diagram.

Na označenie miest, kde je možné rozšírenie toku udalostí sa používajú body rozšírenia. Body rozšírenia (extension points) sú pomenované miesta v toku udalostí [7], kam je možné vložiť ďalšie správanie sa systému. Každý tok udalostí môže mať viacero bodov rozšírenia. Body rozšírenia

poskytujú ľahko čitateľný spôsob pre odvolávanie sa na určité miesto v opise prípadu použitia. Výhodou teda je, že sa netreba odvolávať na čísla, ktoré sa môžu meniť a netreba ani opisovať miesto rozšírenia slovne. Na druhej strane je nevýhodou ťažké hľadanie vhodného miesta a potreba pridávania bodov rozšírenia do prípadu použitia, ktorého opis by sa už nemal meniť.

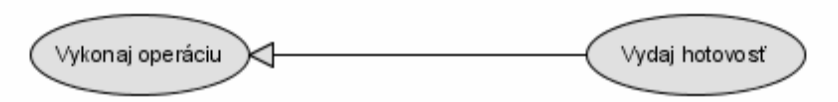


Obr. 5: UML notácia vzťahu rozšírenia

V UML sa vzťah rozšírenia značí pomocou šípky smerujúcej k rozširovanému prípadu použitia, ktorá má stereotyp ‹‹extend›› (pozri *Obr. 5*). Ako vidieť na obrázku, podmienka rozšírenia a bod rozšírenia, v ktorom prípad použitia rozširuje iný prípad použitia, sa v UML nedajú inak zaznačiť, len pomocou poznámky. V textovom opise prípadu použitia je vzťah rozšírenia reprezentovaný tak, že sa vytvorí rozširujúci prípad použitia, ktorý môže mať nielen základný tok udalostí, ale aj alternatívne. V ňom sa potom určí bod rozšírenia nachádzajúci sa v rozširovanom prípade a rozširujúci tok udalostí, ktorý sa pri rozšírení použije.

3.9.3 Vzťah zovšeobecnenia

Vzťah zovšeobecnenia (generalization relationship) je v UML špecifikácii definovaný ako taxonomický vzťah medzi všeobecným a špecifickým klasifikátorom, pričom špecifický klasifikátor dedí vlastnosti toho všeobecného [11]. Zovšeobecnie prípadov použitia sa dá aplikovať na prípady použitia, ktoré obsahujú podobné toky udalostí a hlavne pomáhajú dosiahnuť účastníkovi ten istý cieľ. Špecializovaný prípad použitia dedí od všeobecného prípadu použitia základný aj alternatívne toky udalostí.



Obr. 6: UML notácia pre vzťah zovšeobecnenie

V diagrame prípadov použitia sa kreslí vzťah zovšeobecnenia rovnako ako pri iných UML diagramoch (pozri *Obr. 6*). Ak sa zapisuje zovšeobecnenie v textovej forme postupuje sa tak, že sa vytvorí nový opis prípadu použitia a vyplnia sa iba tie časti opisu, ktoré sú špecializované. V prípade špecializácie tokov udalostí existujú dve často používané metódy:

- Prepíšu sa celé toky udalostí, ktoré majú byť špecializované [12].
- V tokoch udalostí všeobecného prípadu použitia sa vytvoria body rozšírenia, ktoré sa v špecializovanom prípade použitia použijú na doplnenie špecializovaných častí [7].

4 Podpora prípadov použitia v nástrojoch

V dnešnej dobe existuje veľa nástrojov na podporu modelovania prípadov použitia. Väčšinou ide o CASE nástroje, ktoré podporujú všetky diagramy UML špecifikácie a teda aj diagram prípadov použitia. Okrem nich však existujú aj nástroje, ktoré sú špeciálne vyvinuté pre podporu prípadov použitia. Tie sa často sústredia iba na textový opis prípadov použitia a diagramy prípadov použitia. Podpora modelovania prípadov použitia v jednotlivých CASE nástrojoch je veľmi podobná. V prípade špecializovaných nástrojov existuje skôr snaha poskytnúť čo najlepšiu podporu modelovania prípadov použitia a preto sa často výrazne odlišujú.

4.1 Názov, stručný opis a predmet

Stručný opis je vo všetkých nástrojoch na podporu modelovania prípadov použitia realizovaný prostredníctvom neformálneho textu, ktorý sa píše na nejaké vyhradené miesto. Mnoho nástrojov poskytuje na zadanie stručného opisu prípadu použitia textový editor podobný programu Microsoft Wordpad. Podobne je na tom aj podpora pre pomenovanie prípadov použitia. Podpora pre pomenovanie prípadov použitia často znamená iba zadanie mena prípadu použitia. Niektoré špecializované nástroje ešte navyše poskytujú automatické číslovanie prípadov použitia. Predmet prípadu použitia je taktiež často riešený iba zadaním textu do textového poľa. Na druhej strane však treba povedať, že nejaká špeciálna podpora pre túto časť opisu prípadu použitia nemá zmysel. Navyše možno predmet vyznačiť v diagrame prípadov použitia, ktorý je súčasťou takmer každého nástroja na podporu prípadov použitia.

4.2 Zainteresovaní a účastníci

Podpora zainteresovaných a účastníkov je v nástrojoch na podporu modelovania prípadov použitia väčšinou založená na oddelení účastníkov od prípadov použitia, tak ako to je v diagrame prípadov použitia. Každý účastník má svoje meno a slovný opis. Niektoré nástroje z opisu účastníka špeciálne vyčleňujú ciele a dokonca je možné sa stretnúť aj s možnosťou vytvorenia prípadov použitia na základe uvedených cieľov.

Často sa v nástrojoch pre modelovanie prípadov použitia nerozlišuje medzi hlavným a vedľajším účastníkom. V špecializovaných nástrojoch sa tento problém rieši tak, že sa pri každom prípade použitia účastníci vymenujú a určí sa, ktorý z nich je hlavný a ktorý vedľajší. Keďže účastníci sú oddelení od prípadov použitia, nástroje dokážu poskytnúť zoznam účastníkov, ktorých je možné v prípade použitia zaradiť medzi hlavných alebo vedľajších účastníkov. Pre zainteresovaných neexistuje nijaká špeciálna podpora okrem vyhradeného miesta pre napísanie vlastného textu.

4.3 Toky udalostí a vzťahy medzi prípadmi použitia

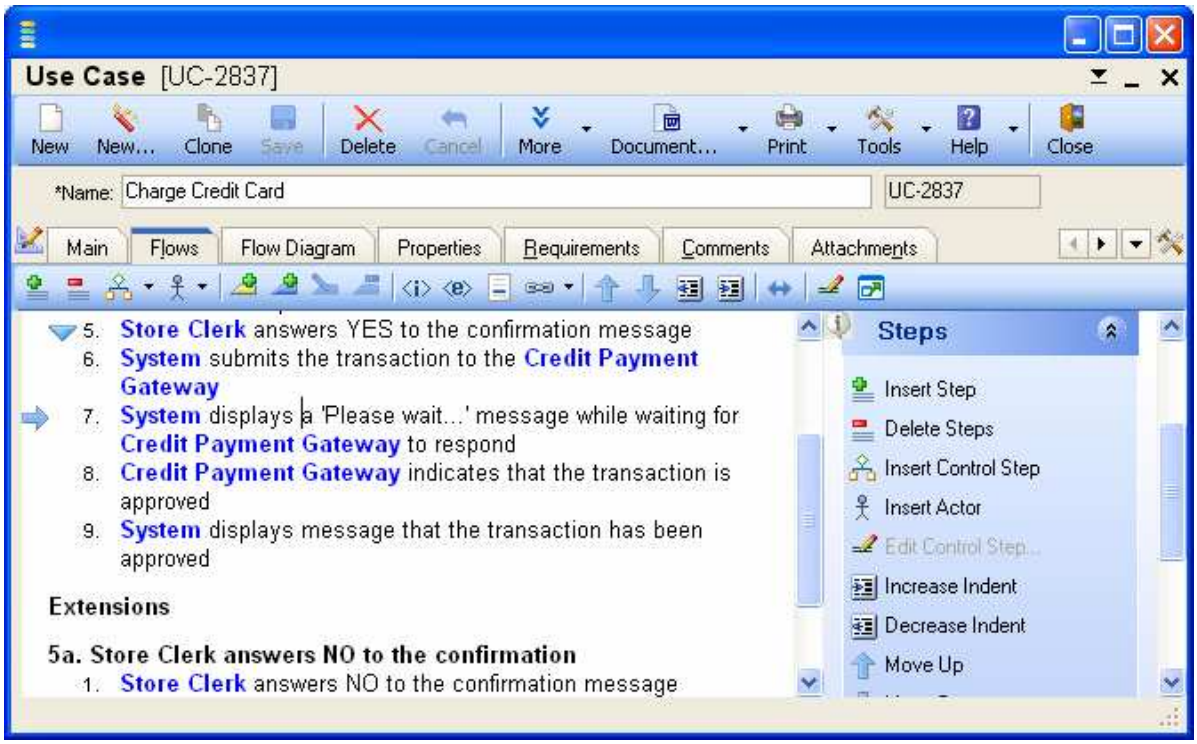
Toky udalostí sú najdôležitejšou časťou prípadov použitia a ich podpora v nástrojoch na modelovanie prípadov použitia je naozaj rozmanitá. V jednotlivých nástrojoch je vidieť rôzne prístupy, ktoré je možné zaradiť do troch kategórii: textovo orientované nástroje, šablónovo orientované nástroje a modelovo orientované nástroje.

Textovo orientované nástroje majú celý opis prípadov použitia zapísaný pomocou obyčajného alebo formátovaného textu v jednom alebo niekoľkých textových poliach. Nástroje, ktoré používajú formátovaný text podporujú napríklad číslovanie, odrážky, rôzne farby písma, štýly písma a podobne. Napriek tomu, že je v týchto nástrojoch možné vytvoriť dobre čitateľné textové opisy prípadov použitia, neposkytujú absolútne žiadnu podporu ich udržiavania. Do tejto kategórie

nástrojov je možné zaradiť napríklad ArgoUML, Poseidon for UML a IBM Rational Software Architect.

V šablónovo orientovaných nástrojoch je textový opis prípadov použitia založený na statickej alebo dynamickej šablóne, pomocou ktorej sa vytvára štruktúrovaný text. Šablóna sa zvyčajne skladá z tokov udalostí a jednoduchých častí textového opisu ako napríklad stručný opis, vstupné podmienky, výstupné podmienky a iné podobné časti textového opisu prípadov použitia. V prípade dynamických šablón je možné prispôsobenie šablón pridaním nových alebo odstránením existujúcich tokov udalostí a jednoduchých častí textového opisu. Rovnakým spôsobom vznikajú aj nové šablóny, ktoré je potom možné v danom nástroj používať. Aj keď udržiavanie textových opisov prípadov použitia v tých nástrojoch je stále problematické, ľahšie sa tieto opisy píšu a čítajú. Predstaviteľmi tejto kategórie nástrojov sú Visual Paradigm alebo Enterprise Architect.

V modelovo orientovaných nástrojoch vychádzajú textové opisy prípadov použitia zo špecifického modelu. Nástroje tejto kategórie poskytujú sofistikované textové editory, ktoré priamo menia štruktúru opisu prípadov použitia podľa príslušného modelu. Napríklad každý krok je súčasťou určitého toku udalostí, čo umožňuje používateľovi nástroja zmeniť poradie krokov s tým, že ostatné kroky v toku udalostí sa prečíslujú automaticky. Automaticky sa samozrejme upravia aj odkazy na kroky v alternatívnych tokoch udalostí a bodoch rozšírenia, pretože súvisiace časti sú v modeli prepojené. Modelovo orientované nástroje poskytujú najlepšiu podporu modelovania prípadov použitia, hlavne dobré udržiavanie textových opisov prípadov použitia. Do tejto kategórie nástrojov je možné zaradiť napríklad Visual Use Case a CaseComplete.



Obr. 7: Úprava tokov udalostí v nástroji Visual Use Case

4.4 Vstupné a výstupné podmienky

Vstupné a výstupné podmienky sú podporované takmer každým nástrojom na podporu prípadov použitia, pričom existujú dva také základné prístupy:

1. Vstupné a výstupné podmienky sú súhrnne nazývané obmedzenia, ktoré môžu byť určitého typu. Každé obmedzenie má okrem typu meno a opis, v ktorom sa uvedie podmienka.
2. V opise prípadu použitia sú vyhradené textové polia pre vstupné a pre výstupné podmienky, kam je možné podmienky napísať.

4.5 Ostatné prvky podpory prípadov použitia

Všetky predchádzajúce možnosti podpory prípadov použitia sa týkali vytvárania opisu prípadov použitia. Mnoho nástrojov však poskytuje ďalšie možnosti podpory opisu prípadov použitia ako napríklad:

Manažment verzií prípadov použitia – S manažmentom verzií prípadov použitia sa je možné stretnúť pri viacerých nástrojoch, či ide o CASE nástroje alebo špecializované nástroje na modelovanie prípadov použitia. Manažment verzií prípadov použitia však nie je nič iné ako zaznamenávanie zmien v prípadoch použitia pri vytváraní modelu prípadov použitia. Niektoré nástroje iba zvyšujú číslo verzie prípadu použitia, prípadne zaznamenávajú komentár autora o zmene. Iné nástroje dokážu zaznamenávať úplne všetky zmeny tak, akoby išlo o manažment verzií zdrojových kódov a teda môžu realizovať operácie ako porovnanie verzií alebo návrat na niektorú z predchádzajúcich verzií prípadu použitia.

Export dokumentov použitím rôznych šablón – Takmer každý lepší nástroj na modelovanie prípadov použitia obsahuje export modelu prípadov použitia vo forme dokumentu. Poskytujú rôzne formáty exportu, ktorých obsah má určitú vopred stanovenú štruktúru. Ak nástroj obsahuje manažment verzií na začiatku dokumentu je typicky história dokumentu. Po histórii nasleduje obsah dokumentu a potom nasledujú jednotlivé časti dokumentu ako napríklad účel dokumentu, účastníci prípadov použitia, opis prípadov použitia, otvorené problémy a iné.

Import prípadov použitia z dokumentov – Import prípadov použitia z dokumentov nie je moc populárny, pretože závisí od štruktúry a formátu importovaného dokumentu. V prípade, že nástroj nepozná štruktúru zadaného dokumentu, nemôže ho importovať a preto je import prípadov použitia často obmedzený na exportované dokumenty daného nástroja. Samozrejme existujú aj nástroje, ktoré dokážu importovať prípady použitia z dokumentov exportovaných v iných nástrojoch. Tie sa však musia prispôsobovať zmenám v štruktúre dokumentov exportovaných v iných nástrojoch, čo býva často náročné.

Kolaborácia pri vytváraní opisov prípadov použitia – Kolaborácia je jednou z najdôležitejších súčastí nástrojov na modelovanie vo všeobecnosti a preto je podporovaná takmer každým nástrojom. Dôvodom je to, že na vytváraní modelov pracuje viac ľudí a je nemysliteľné, aby každý z nich pracoval na lokálnej kópii modelov. Kolaborácia je podporovaná buď existenciou špecializovaného servera alebo iba prístupom na samostatnú databázu. V oboch prípadoch sa používa na modelovanie klient, ktorý sa pripája buď na server alebo na databázu.

História zmien prípadov použitia – História zmien súvisí s kolaboráciou a manažmentom verzií. Je to populárna vlastnosť širokej množiny nástrojov na modelovanie prípadov použitia. Z histórie môžeme minimálne vyčítať kto, kedy a akú zmenu urobil v prípade použitia resp. modeli prípadov použitia.

Slovník pojmov a ich vyznačovanie v texte – Slovník pojmov je dôležitou súčasťou modelu prípadov použitia a preto je často súčasťou nástrojov na modelovanie prípadov použitia. Jeho dôležitosť spočíva v, že slúži na uchovávanie definícií pojmov používaných v oblasť, v rámci

ktorej sa vyvíja určitý softvér a tým pomáha osobám, ktoré nie sú do danej oblasti zainteresované, porozumieť opisu prípadov použitia. Niektoré nástroje dokážu aj označovať slová v slovníku priamo v texte, čím uľahčujú zaznamenávanie prípadov použitia.

Publikovanie opisov prípadov použitia na webe – V prípade publikovania opisov prípadov použitia ide vlastne o export vo formáte html okamžite uložený na internetový server. Iba niektoré nástroje podporujú takúto formu exportu. Server býva buď integrovaný priamo v nástroji alebo nástroj poskytuje možnosť exportu modelu prípadov použitia a automatickej konfigurácie internetového servera, ktorý musí byť nainštalovaný používateľom.

Kontrola pravopisu – Kontrolu pravopisu obsahujú takmer všetky nástroje, aj tie ktoré poskytujú slabú podporu prípadov použitia. Je to spôsobené tým, že veľká väčšina nástrojov používa na zaznamenanie prípadov použitia textový editor. Kontrola pravopisu je samozrejme veľmi dôležitá, keďže výsledný dokument je určený všetky pre zainteresované osoby.

Pripájanie súborov k prípadom použitia – Pripájanie súborov k prípadom použitia je možné použiť na viaceré účely, napríklad na pripájanie dokumentov so zozbieranými požiadavkami, pripájanie obrázkov, pripájanie dokumentov na internete a podobne. Túto funkcionalitu poskytuje mnoho nástrojov, keďže je jednoduchá, efektívna a ľahko sa implementuje.

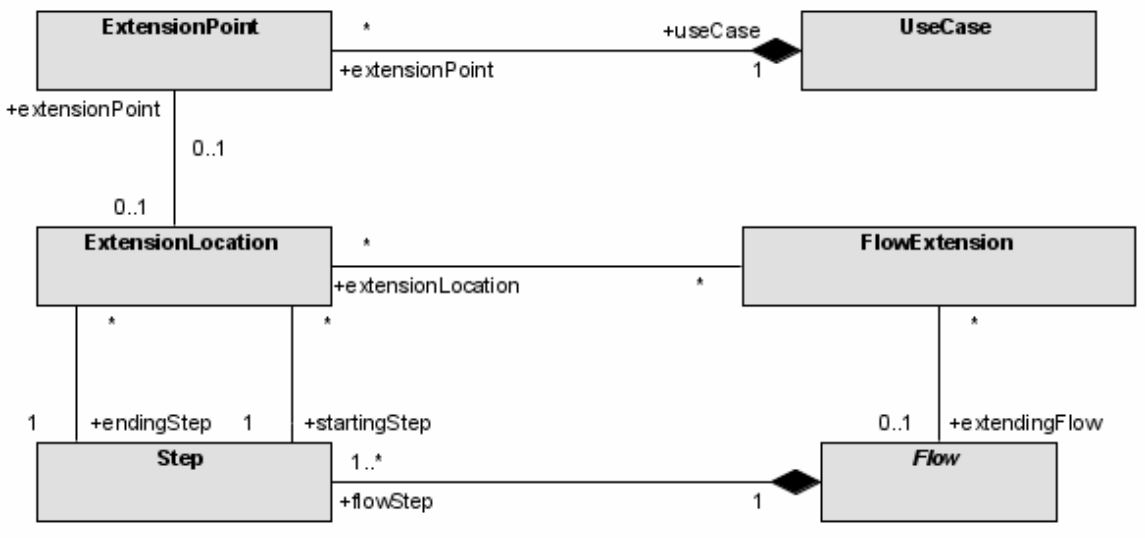
5 Metamodel modelovania prípadov použitia

Modelovanie prípadov použitia pomocou štruktúrovaného textu má v súčasnosti veľmi veľa podôb. Metodológovia zaoberajúci sa prípadmi použitia sa síce snažia držať základnej štruktúry, ktorá bola prezentovaná v analýze, ale takmer každý z nich má svoj vlastný štýl opisu prípadov použitia vychádzajúci z jeho praktických skúseností. Rozdiely existujú hlavne pri zápise vzťahov medzi prípadmi použitia a s tým súvisiaca reprezentácia tokov udalostí a bodov rozšírenia, pretože problém ich správnej praktickej aplikácie existuje už od ich zavedenia niekedy medzi rokmi 1987 až 1992 [1].

S cieľom zovšeobecnenia modelovania prípadov použitia bol vytvorený metamodel modelovania prípadov použitia, ktorý je založený na metamodeli modelovania prípadov použitia jazyka UML. Pri jeho tvorbe sa postupovalo tak, že metamodel modelovania prípadov použitia jazyka UML bol doplnený o entity tvoriace textový opis prípadov použitia a vzťahy medzi nimi tak, aby vzniknutý metamodel podporoval viaceré notácie textového opisu prípadov použitia. Metamodel vznikol na základe výberu najvšeobecnejších alebo kombinácie viacerých možností reprezentácie pre jednotlivé problematické časti štruktúry textového opisu prípadov použitia, ktoré sú známe z literatúry.

5.1 Reprezentácia bodov rozšírenia

Body rozšírenia sa podľa UML vždy definujú v rozširovanom prípade použitia. V textovom opise prípadov použitia má zvyčajne každý bod rozšírenia svoj názov a pozíciu v toku udalostí. Pozícia môže byť určená buď jedným alebo dvomi krokmi. Ak je pozícia určená dvomi krokmi ide o bod rozšírenia, ktorý pokrýva postupnosť krokov medzi počiatočným a konečným krokom [8]. Prípad použitia s takýmto bodom rozšírenia je možné rozšíriť v každom kroku zahrnutom v bode rozšírenia a k rozšíreniu prichádza najmä na základe podmienky, ktorá je súčasťou vzťahu rozšírenia medzi dvoma prípadmi použitia.



Obr. 8: Navrhnutá reprezentácia bodov rozšírenia

V navrhnutom metamodeli je bod rozšírenia spojený cez miesto rozšírenia (ExtensionLocation) s dvoma krokmi (Step), aby bolo možné reprezentovať body rozšírenia (ExtensionPoint), ktoré pokrývajú postupnosť viacerých krokov. Tým sa samozrejme nevylučuje možnosť, že počiatočný

a koncový krok bude rovnaký a teda pôjde o bod rozšírenia, ktorého pozícia v toku udalostí je určená jedným bodom. Výhodou takejto reprezentácie bodov rozšírenia je práve možnosť definovania jednokrokových aj viackrokových bodov rozšírenia. Prepojenie cez miesto rozšírenia navyše umožňuje vytvárať rozšírenia prípadov použitia bez bodov rozšírenia iba na základe krokov.

V praxi sa možno stretnúť s prístupom, kde pozícia bodu rozšírenia môže byť definovaná iba jedným krokom. Napriek tomu, že tento prístup je prakticky zahrnutý v navrhnutom spôsobe reprezentácie bodov rozšírenia, poukazuje na určitý problém navrhnutého spôsobu. Problém sa dá ilustrovať na funkciách bankomatu, ktoré majú byť vyjadrené ako rozširujúce prípady použitia. Hlavný tok rozširovaného prípadu použitia sa dal napísať napríklad takto:

Hlavný tok:

1. Používateľ vloží bankomatovú kartu.
2. Systém si vyžiada PIN kód.
3. Používateľ zadá PIN kód.
4. Systém overí používateľom zadaný PIN kód.
5. Používateľ zvolí funkciu bankomatu.

{Vykonanie funkcie bankomatu}

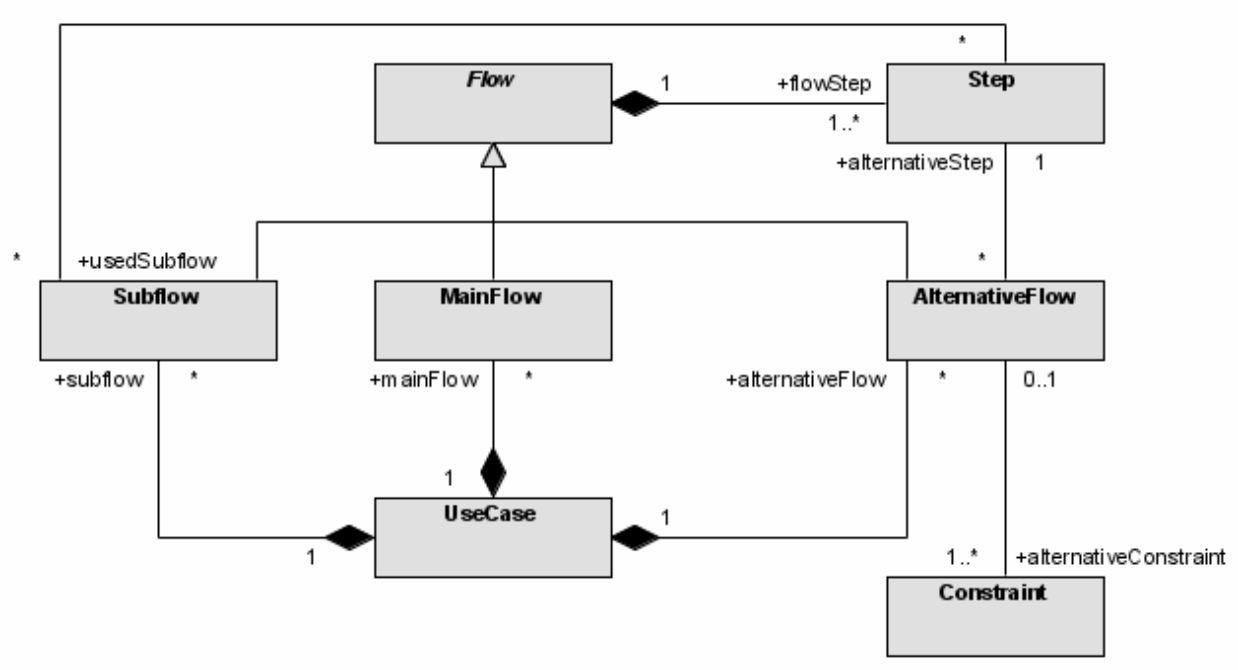
6. Používateľ ukončí prácu s bankomatom.
7. Systém vráti používateľovi bankomatovú kartu.

Uvedený príklad obsahuje bod rozšírenia, ktorý je použitý ako krok toku udalostí, ale nie je očíslovaný. Už na pohľad je zrejmé, ako sa budú jednotlivé kroky prípadu použitia, vrátane rozšírení, vykonávať. Najprv sa vykonajú kroky 1 až 5, potom dôjde k vykonaniu krokov z rozširujúceho prípadu použitia a nakoniec vykonaniu krokov 6 a 7. Na zapísanie tohto bodu rozšírenia v navrhnutej reprezentácii bodov rozšírenia je potrebné vybrať krok, ktorý bude miestom rozšírenia. Problém však spočíva v tom, že sa ponúkajú dve možnosti a to krok 5 a krok 6.

Na vyriešenie identifikovaného problému existuje viacero možností. Napríklad je možné implicitne predpokladať, že v prípade bodu rozšírenia sa najprv vykoná krok toku udalostí a potom dôjde k rozšíreniu prípadu použitia. Takéto riešenie však neprispieva k cieľom tejto práce. Lepšie riešenie je ponúknutie možnosti definovania správania sa bodu rozšírenia na základe vzťahu rozšírenia medzi dvoma prípadmi použitia.

5.2 Reprezentácia tokov udalostí

Toky udalostí sú podstatnou súčasťou textovej reprezentácie prípadov použitia. V navrhnutom metamodeli vychádza reprezentácia tokov udalostí zo všeobecne uznávanej predstavy, že ide o postupnosť krokov. Základom reprezentácie je teda krok, ktorý môže v navrhnutej reprezentácii obsahovať textový opis akcie vykonanej účastníkom prípadu použitia, podmienku, bod opätovného spojenia (rejoin point), názov zahrnutého prípadu použitia a ďalšie iné možnosti alebo ich kombináciu. Dôvodom zvolenia takejto voľnej reprezentácie krokov je to, že nie je potrebné v metamodeli presne vymedzovať syntax krokov. Metamodel navyše neobsahuje ani spôsob zoradenia krokov do toku udalostí, pretože na úrovni metamodelu stačí vedieť, že tok udalostí sa skladá z krokov.



Obr. 9: Navrhnutá reprezentácia tokov udalostí

V navrhnutom modeli vystupujú tri typy tokov udalostí (Flow) a to hlavný tok (MainFlow), alternatívny tok (AlternativeFlow) a podtok (Subflow). Hlavný a alternatívny tok udalostí už boli spomenuté v analýze textového opisu prípadov použitia. Alternatívny tok sa od hlavného toku odlišuje v navrhnutej reprezentácii v tom, že obsahuje podmienku aktivácie (Constraint) a krok, v ktorom sa je možné alternatívny tok vykonať. Podmienka a krok toku udalostí sa veľmi podobajú reprezentáciu bodu rozšírenia uvedenú v predchádzajúcej časti. V skutočnosti existujú aj prístupy k textovej reprezentácii prípadov použitia, ktoré používajú body rozšírenia v alternatívnych tokoch udalostí [7].

Pri alternatívnych tokoch sa často spomína aj krok, ktorý môže určiť krok v hlavnom toku udalostí, kde bude interakcia pokračovať. Tento krok sa nazýva bod opätovného spojenia. Všeobecne platí, že ak bod opätovného spojenia neexistuje v toku udalostí, interakcia bude pokračovať v bode, ktorý je definovaný v alternatívnom toku. Využitím bodov opätovného spájania je možné rozdeliť alternatívne toky na bežne používané typy [15], ktorými sa táto práca nezaoberá podrobne, pretože ide skôr o praktické využívanie textovej reprezentácie prípadov použitia. Z toho dôvodu ani navrhnutý metamodel neobsahuje špeciálne rozdelenie na typy alternatív a ponúka iba všeobecnú možnosť ich reprezentácie prostredníctvom kroku, ktorý môže obsahovať aj bod opätovného spojenia.

Okrem dvoch spomínaných tokov vystupuje v navrhnutom metamodeli ešte tretí, ktorý bol nazvaný podtok. Podtok je pomenovaný tok udalostí, ktorý nie je hlavným ani alternatívnym tokom. Typicky sa využíva na opis niektorých menej podstatných postupností krokov, ak je opis prípadu príliš veľký. To znamená, že umožňuje zjednodušiť opis prípadu použitia, čím sa stane lepšie čitateľný, ale stále bude rovnako detailný. Podtoky sa ďalej môžu použiť aj na zníženie redundancie niektorých častí opisu, keď sa rovnaké sekvencie krokov opakujú v hlavnom toku alebo alternatívnych tokoch. Typickým prípadom využitia podtoku na zníženie redundancie je napríklad prihlásenie používateľa do systému:

Podtok: Prihlásenie používateľa

1. Systém si vyžiada PIN kód.

2. Používateľ zadá PIN kód.
3. Systém overí používateľom zadaný PIN kód.

Hlavný tok prípadu použitia z príkladu uvedeného pri návrhu reprezentácie bodov rozšírenia by s využitím podtoku pre prihlásenie používateľa mohol vyzerať nasledovne:

Hlavný tok:
1. Používateľ vloží bankomatovú kartu.
2. Vykonanie podtoku **Prihlásenie používateľa**.
3. Používateľ zvolí funkciu bankomatu.

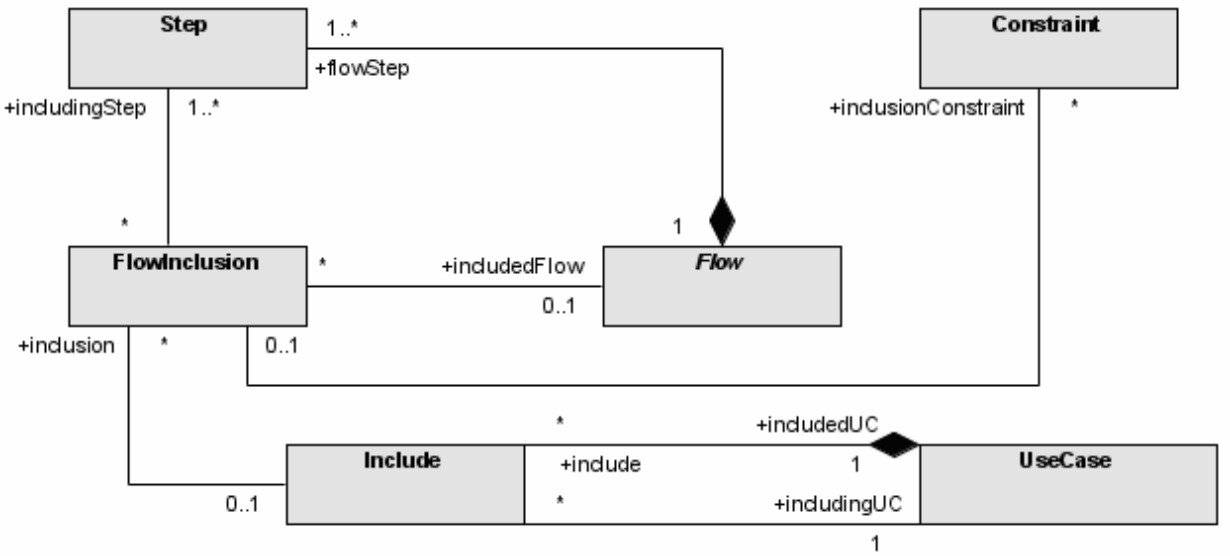
{Vykonanie funkcie bankomatu}

4. Používateľ ukončí prácu s bankomatom.
5. Systém vráti používateľovi bankomatovú kartu.

Podtoky sú navrhnutom metamodeli reprezentované tak, že sa dajú využiť v každom type tokov udalostí a teda aj samotných podtokoch. Cieľom takéhoto prístupu je dosiahnutie čo najväčšej voľnosti pri ich používaní. Ako je vidieť na príklade hlavného toku prípadu použitia, podtoky sa v tokoch udalostí používajú podobne ako vzťah zahrnutia. Táto skutočnosť však nie je náhoda, ale skôr dôsledok toho, že pri vzťahu zahrnutia dochádza často k znovupoužitiu určitého správania alebo k extrakcii rovnakých častí viacerých prípadov použitia. Podobne to funguje aj pri podtokoch, ale s tým rozdielom, že k znovupoužitiu alebo extrakcii dochádza na úrovni jedného prípadu použitia.

5.3 Reprezentácia vzťahov medzi prípadmi použitia

Vzťahy medzi prípadmi použitia sú rovnako dôležité ako toky udalostí, pretože vlastne vytvárajú opis funkcionálnych požiadaviek špecifickou kombináciou viacerých prípadov použitia resp. kombináciou viacerých tokov udalostí. Pôvodnou predstavou o prípadoch použitia bolo, že každý prípad použitia musí byť spojený s účastníkom, ktorý ho spúšťa. Táto predstava však veľmi obmedzovala možnosť použitia vzťahov medzi prípadmi použitia, pretože nájsť napríklad spoločné správanie systému, ktoré by zároveň bolo prípadom použitia bolo veľmi ťažké. Z tohto dôvodu existujú v súčasnosti okrem prípadov použitia aj fragmenty prípadov použitia, ktoré však podľa Jacobsona nemožno považovať za prípady použitia [1]. Napriek tomu sa tieto fragmenty prípadov použitia modelujú v UML rovnako ako prípady použitia. V diagrame ich možno identifikovať podľa toho, že nemajú účastníka a sú spojené s iným prípadom použitia vzťahom zahrnutia alebo rozšírenia.



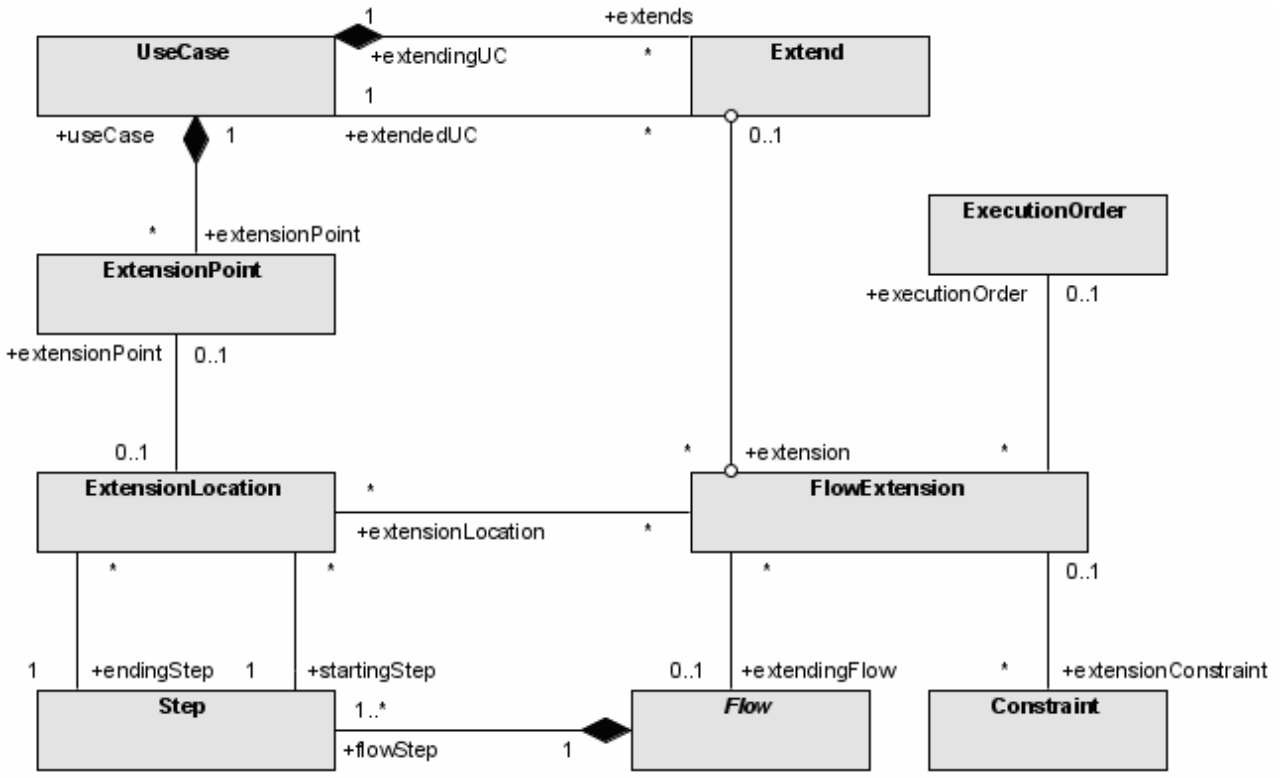
Obr. 10: Navrhnutá reprezentácia vzťahu zahrnutia

Zavedením pojmu fragment prípadu použitia sa uvoľnila štruktúra opisu takýchto fragmentov a postupne vznikli rôzne návrhy opisu. Zaujímavou, aj keď nie prekvapujúcou, skutočnosťou je to, že fragmenty prípadov použitia, ktoré boli vytvorené iba pre potreby vzťahu zahrnutia mali vo väčšine návrhov úplne rovnakú štruktúru opisu ako samotné prípady použitia okrem účastníkov, ktorý sa v štruktúre opisu nenachádzali. Základnou predstavou bolo, že pri zahrnutí sa hlavný tok udalostí zahrnutého fragmentu prípadu použitia sa vloží do hlavného toku prípadu použitia a alternatívne toky sa zjednotia. Výhodou takéhoto prístupu je to, že pri vzťahu zahrnutia netreba žiadne špeciálne označenie toku udalostí, ktorý sa má vložiť do prípadu použitia a navyše je možné do opisu fragmentu prípadu použitia pridať účastníka a dostať prípad použitia. Na druhej strane je však takýto prístup príliš obmedzujúci.

V navrhnutej reprezentácii je vzťah zahrnutia (Include) reprezentovaný tak, že sa nachádza medzi dvoma prípadmi použitia (UseCase), ktoré môžu byť aj fragmentmi, pretože prípad použitia v metamodeli nemusí byť spojený s účastníkom (pozri *Obr. 12*). Okrem toho môže vzťah zahrnutia obsahovať viacero zahrnutých tokov udalostí (FlowInclusion) v rôznych krokoch. Každé zahrnutie toku môže mať vlastné podmienky, za ktorých k zahrnutiu toku dôjde. Výhodou takéhoto prístupu je možnosť reprezentovať všetky kombinácie medzi zahrnutiami tokov a vzťahmi zahrnutia medzi dvomi prípadmi použitia. V praxi sa však používajú často iba dve kombinácie a to [12]:

1. Medzi dvomi prípadmi použitia je jeden vzťah zahrnutia, ktorý obsahuje všetky zahrnutia tokov.
2. Medzi dvomi prípadmi použitia je toľko vzťahov zahrnutia, koľko je zahrnutí tokov a každý vzťah zahrnutia obsahuje jedno zahrnutie toku.

Pri fragmentoch, ktoré vznikali v dôsledku vzťahu rozšírenia, bola situácia dosť odlišná ak pri vzťahu zahrnutia. Dôvodom bola potreba rozšírenia jedného prípadu použitia na viacerých miestach jedným alebo viacerými rôznymi tokmi udalostí, čo by nebolo možné urobiť fragmentom so štruktúrou opisu, ktorý bol spomenutý pri zahrnutí. Populárnou sa stala štruktúra opisu fragmentov prípadov použitia, ktorá obsahovala voľné pomenované toky udalostí a tie mohli byť použité pri rozšírení prípadov použitia. Jediným rozdielom oproti tokom udalostí, ktoré boli navrhnuté v reprezentácii prípadov použitia je to, že obsahujú aj samotný vzťah rozšírenia a teda každý tok má určené pri akej podmienke, do ktorého prípadu použitia, na ktoré miesto sa má vložiť. Rozširujúci fragment je totiž jediným miestom, kde je možné tieto informácie umiestniť, pretože základný prípad použitia nesmie obsahovať informácie o rozšírení [12].



Obr. 11: Navrhnutá reprezentácia vzťahu rozšírenia

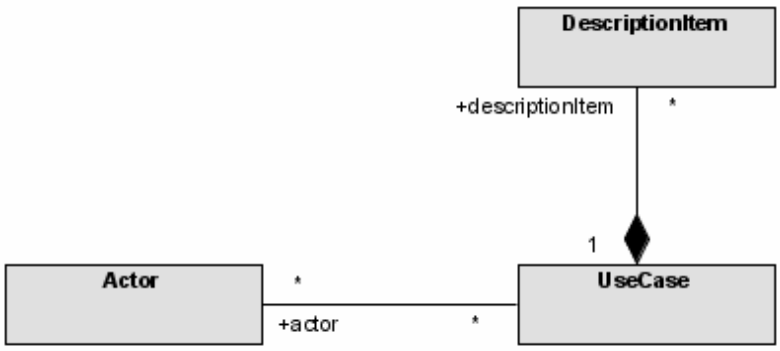
V navrhnutej reprezentácii vzťahu rozšírenia (Extend) sa viaceré podmienky a miesta rozšírenia vzťahujú na jedno rozšírenie tokom (FlowExtension), ktorých môže vzťah rozšírenia obsahovať viacero. Tým, že vzťah rozšírenia oddelený od toku je možné v rozšíreniach tokom použiť všetky navrhnuté typy tokov bez toho, aby bolo potrebné vytvoriť ďalší typ. Dôsledkom takejto reprezentácie je aj možnosť použitia toho istého toku na rozšírenie jedného prípadu použitia na viacerých miestach alebo viacerých prípadov použitia. Navyše je na rozšírenie možné vybrať ktorýkoľvek tok udalostí či už ide o hlavný tok, alternatívny tok alebo podtok.

Pri návrhu reprezentácie bodov rozšírenia bol identifikovaný jeden problém, ktorý sa prejavuje aj pri vzťahu rozšírenia. Problém spočíva v tom, že pri bodoch rozšírenia definovaných pomocou krokov toku udalostí neexistuje explicitný spôsob presného určenia miesta rozšírenia toku udalostí. V navrhnutom metamodeli je riešenie tohto problému reprezentované poradím vykonávania (ExecutionOrder) rozšírení tokom udalostí vzhľadom na miesto rozšírenia. Toto riešenie sa podoba na mechanizmus videní, ktoré sa používajú v aspektovo-orientovanom programovaní. V aspektovo-orientovanom programovaní sa videnia vykonávajú pred (before), po (after) alebo namiesto (around) bodu opätovného spájania [8]. Aplikáciou tohto prístupu na miesta rozšírenia by bolo možné určiť, či sa vkladaný toku udalostí vkladá pred, po alebo namiesto kroku v toku udalostí.

Na prvý pohľad by sa mohlo zdať, že by bolo najlepšie definovať body rozšírenia spolu s explicitným určením miesta vloženia rozširujúceho toku udalostí. Problémom je však to, že takto definovaný bod rozšírenia je príliš obmedzujúci, pretože všetky toky vložené na miesto určené takýmto bodom rozšírenia by sa vkladali rovnako, napríklad pred, po alebo namiesto kroku v toku udalostí. V prípade, že by bolo potrebné použiť všetky možnosti poradia vykonávania pre nejaké miesto v toku udalostí, museli by sa definovať rôzne body rozšírenia pre to isté miesto rozšírenia pre všetky možnosti poradia vykonávania. Používanie explicitného definovania presného miesta na vloženie krokov toku udalostí sa preto v literatúre typicky vzťahuje na rozšírenie [8,12] a tento prístup je použitý aj v navrhnutom metamodeli.

5.4 Reprezentácia ostatných častí opisu prípadov použitia

Pod ostatnými časťami je možné rozumieť hlavne účastníkov a štruktúrou jednoduché časti opisu prípadov použitia. V navrhnutom metamodeli môže byť každý účastník (Actor) spojený s viacerými prípadmi použitia systému a každý prípad použitia môže byť spojený s viacerými účastníkmi tak, ako to vyplýva z analýzy textového opisu prípadov použitia. Zvyšné časti opisu prípadov použitia (DescriptionItem) sú reprezentované podobne ako kroky v tokoch udalostí, čiže nie je podstatný ich obsah. Ide teda o modelom podporovaný spôsob dotvorenia opisu prípadov použitia.



Obr. 12: Návrh ostatných častí opisu prípadov použitia

5.5 Alternatívne toky a vzťah rozšírenia

Alternatívny tok a vzťah rozšírenia sa používajú na rozšírenie toku udalostí iným tokom udalostí. Oba spôsoby rozšírenia sa veľmi podobajú podobne ako podtok a vzťah zahrnutia. Rozšírenie je totiž v oboch prípadoch určené miestom, podmienkou a rozširujúcim tokom. Táto skutočnosť vedie často k problémom v praktickej aplikácii alternatívnych tokov a vzťahu rozšírenia, pretože niekedy sa ponúka možnosť použiť oba spôsoby rozšírenia toku udalostí a vtedy je potrebné vybrať si jeden zo spôsobov.

Existenciu tohto problému identifikoval aj Gunnar Övergaard [12] a uviedol, že je chybou modelovať alternatívne toky vzťahmi rozšírenia extrakciou správania do fragmentu prípadu použitia. Jeho vysvetlenie je založené na tom, že keď sa zoberie alternatívny tok a vytvorí fragment prípadu použitia so vzťahom rozšírenia, základný prípad použitia, kde sa alternatívny tok použitia nachádzal, nebude úplný bez rozšírenia, pretože alternatívny tok bude chýbať. Prípad použitia totiž musí obsahovať hlavný tok a všetky podstatné alternatívne toky, aby bol prípad použitia úplný.

Trochu iný pohľad na tento problém môžeme nájsť u Kurta Bittnera [7], ktorý pripúšťa možnosť modelovania alternatívneho toku prostredníctvom vzťahu rozšírenia, ale dodáva, že nie všetky alternatívne toky je možné zapísať v tvare fragmentov prípadov použitia. Pri alternatívnych tokoch bol spomenutý bod opätovného spojenia, ktorý môže ukončiť vykonávanie alternatívneho toku udalostí a určiť krok v hlavnom toku, ktorým bude vykonávanie pokračovať. Vzťah rozšírenia však niečo takéto neobsahuje a pozná iba bod rozšírenia, kde sa vkladá tok udalostí ako celok. Z toho je zrejmé, že alternatívne toky, ktoré obsahujú návrat na iný krok ako je v nich definovaný nie je možné len tak extrahovať do fragmentu prípadu použitia. Niektoré alternatívne toky, ktoré opisujú ošetrenie chybových stavov taktiež nie je možné zapísať prostredníctvom rozširujúceho prípadu použitia aj keby sa po vykonaní vracali na rovnaké miesto v hlavnom toku, pretože sú potrebné z hľadiska úplnosti prípadu použitia. Existujú však aj alternatívne toky udalostí, ktoré opisujú voliteľné správanie a práve tieto toky je možné reprezentovať rozširujúcimi prípadmi použitia, pretože aj prostredníctvom vzťahu rozšírenia sa modeluje voliteľné správanie [7].

Otázka používania alternatívnych tokov a vzťahov rozšírenia však príliš neovplyvňuje navrhnutú reprezentáciu, iba poukazuje na to, že je potrebné aby, navrhnutá reprezentácia podporovala prechod od alternatívnych tokov k vzťahu rozšírenia. Z toho dôvodu je potrebné, aby sa aj v alternatívnom toku uvažovalo o poradí vykonávania napriek tomu, že v metamodeli to nie je naznačené.

6 Profilovanie modelu prípadov použitia

Existencia veľkého počtu notácií pre prípady použitia vyjadrené prostredníctvom štruktúrovaného textu vedie k tomu, že si autori modelov vyberú jednu z týchto reprezentácií, zvyčajne obsiahnutú v softvéri, ktorý na modelovanie používajú. Problémom je však to, že väčšina súčasne dostupného softvéru pracuje nad statickou reprezentáciou prípadov použitia, ktorá sa nemusí byť vhodná pre všetky softvérové projekty. Dôsledkom toho je, že rôzni ľudia, ktorí vytvárajú jeden model prípadov použitia, používajú vlastné spôsoby opisu prípadov použitia, aby prekonali nedostatky používaného nástroja, čo často vedie problémom s konzistenciou modelu prípadov použitia

Cieľom navrhnutého metamodelu modelovania prípadov použitia a jeho profilovania je poukázanie na možné zlepšenie načrtnutej situácie. Profilovaním sa dá totiž dosiahnuť možnosť konfigurácie textovej reprezentácie prípadov použitia, ktorú by bolo možné meniť v závislosti od softvérového projektu alebo smerníc používaných v organizácii a zároveň minimalizovať možnosť zadávania voľný neštruktúrovaný text, ktorý zvádza používateľa nástroja na modelovanie prípadov použitia k používaniu vlastného spôsobu zaznamenávania prípadov použitia v textovej forme.

Nad navrhnutým metamodelom boli identifikované viaceré možnosti konfigurácie častí opisu prípadov použitia, z ktorý by sa mohli vytvárať jednotlivé profily:

- používanie jednokrokových bodov rozšírenia
- používanie dvojkrokových bodov rozšírenia
- používanie aspoň jedného hlavný tok udalostí
- používanie viacerých hlavné toky udalostí
- používanie podtokov
- používanie podtokov v hlavných tokoch
- používanie podtokov v alternatívnych tokoch
- používanie podtokov v podtokoch
- používanie alternatívnych tokov
- používanie alternatívnych tokov v hlavných tokoch
- používanie alternatívnych tokov v alternatívnych tokoch
- používanie alternatívnych tokov v podtokoch
- používanie vzťahu rozšírenia
- používanie viacerých miest rozšírení vo vzťahu rozšírenia
- používanie rozšírenia určitým tokom
- používanie podmienok pre rozšírenie tokom
- používanie poradia vykonávania pre rozšírenie tokom
- používanie vzťahu zahrnutia
- používanie podmienok pre zahrnutie toku
- používanie zahrnutia určitého toku
- ďalšie časti opisu prípadov použitia
- špeciálne kroky

Uvedené možnosti identifikované v navrhnutom metamodeli prípadov použitia sú všetky možnosti, ktoré by mohli byť samo o sebe použiteľné, avšak niektoré kombinácie by určite nefungovali spolu. To však nebráni uvažovaniu o nich v rámci profilovania, pretože na úrovni aplikácie, ktorá by takéto profilovanie používala, je možné používateľa na túto skutočnosť upozorniť. Bola teda zvolená skôr cesta voľnosti profilovania pred povolením iba určitých kombinácií, o ktorých známe, že môžu spolu dobre fungovať pri opise prípadov použitia.

Navrhnuté možnosti, ktoré by mohli byť súčasťou profilu, je potrebné overiť a to aplikovaním na niektoré existujúce prístupy k modelovaniu prípadov použitia známe z preštudovanej literatúry, aby sa preukázalo, že profilovaním je možné reprezentovať aj v praxi používané notácie pre textový opis prípadov použitia. Vybrané boli dve dosť odlišné notácie a to Jackobsonova a Cockburnova.

6.1 Jackobsonova notácia

Ivara Jacobsona sme spomínali vo vzťahu k vzniku prípadov použitia a jeho prístup výrazne vplýva na modelovanie prípadov použitia v praxi. Z toho dôvodu bol jeho prístup k modelovaniu prípadov použitia bližšie analyzovaný nielen z hľadiska profilovania, ale aj z hľadiska všetkého, čo bolo v tejto práci dosiaľ napísané, hlavne v rámci návrhu metamodelu modelovania prípadov použitia. Za účelom zostavenia profilu pre jeho prístup bol podrobnejšie rozobraný príklad modelovania vzťahu rozšírenia z knihy, kde vystupuje ako spoluautor [8]:

Prípad použitia: Rezervuj izbu

Hlavné toky:

B1. Rezervuj izbu

1. Zákazník zvolí rezerváciu izby.
2. Systém zobrazí typy izieb a ich ceny.
3. Zákazník vykoná **Over cenu izby**.
4. Zákazník si podá rezerváciu na vybranú izbu.
5. Systém zníži v databáze počet izieb špecifikovaného typu prípustných pre rezerváciu.
6. Systém vytvorí novú rezerváciu so zadanými údajmi.
7. Systém zobrazí potvrdenie rezervácie a inštrukcie pre ubytovanie.

Alternatívne toky:

A1. Duplicitná rezervácia

Ak v kroku 5 základného toku existuje identická rezervácia v systéme (rovnaké meno, email, počiatočný a konečný dátum), systém zobrazí existujúcu rezerváciu a opýta sa zákazníka či chce pokračovať s novou rezerváciou.

1. Ak zákazník chce pokračovať, systém pokračuje v rezervácii a prípad použitia pokračuje.
2. Ak chce zákazník označiť novú registráciu ako duplicitnú, prípad použitia skončí.

Podtoky:

S1. Over cenu izby

1. Zákazník vyberie požadovanú typ izby a uvedie dobu pobytu.
2. Systém vypočíta cenu za špecifikovanú dobu.

Body rozšírenia:

E1. Obnovenie dostupnosti izby

Bod rozšírenia obnovenie dostupnosti izby nastáva v kroku 5 základného toku.

Prípad použitia: Spracuj čakací zoznam

Rozširujúce toky:

EF1. Rad na izbu

Tento rozširujúci tok nastáva v bode rozšírenia obnovenie dostupnosti izby v prípade použitia rezervácia izby keď nie sú dostupné žiadne izby vybraného typu

1. Systém vytvorí čakajúcu rezerváciu s unikátnym identifikátorom pre vybraný typ izby.
2. Systém vloží čakajúcu rezerváciu na čakací zoznam.
3. Systém zobrazí unikátny identifikátor čakajúcej rezervácie zákazníkovi
4. Základný prípad použitia skončí.

V uvedených príkladoch vystupujú dva prípady použitia, Rezervuj izbu a Spracuj čakací zoznam. Z príkladov je možné vyčítať, že Jacobsonov prístup umožňuje viacero hlavných tokov a preto je hlavný tok označený ako B1. Ďalej sa dá vyčítať používanie podtokov, alternatívnych tokov, používanie jednokrokových bodov rozšírenia, používanie podmienky v alternatívnom toku a vzťahu rozšírenia a ďalšie iné. Rozširujúce toky udalostí možno považovať za podtoky a dodatočné informácie v nich za vzťah rozšírenia tak, ako to bolo uvedené pri reprezentácii vzťahov v rámci navrhnutého metamodelu. Na základe dôsledného preštudovania dostupnej literatúry od Ivara Jacobsona bol z navrhnutých možností zostavený profil pre jeho prístup k modelovaniu prípadov použitia.

Tab. 2: Profil pre notáciu Ivara Jacobsona

Možnosť	Hodnota
používanie jednokrokových bodov rozšírenia	áno
používanie dvojkrokových bodov rozšírenia	áno
používanie aspoň jedného hlavný tok udalostí	áno
používanie viacerých hlavné toky udalostí	áno
používanie podtokov	áno
používanie podtokov v hlavných tokoch	áno
používanie podtokov v alternatívnych tokoch	áno
používanie podtokov v podtokoch	áno
používanie alternatívnych tokov	áno
používanie alternatívnych tokov v hlavných tokoch	áno
používanie alternatívnych tokov v alternatívnych tokoch	áno
používanie alternatívnych tokov v podtokoch	áno
používanie vzťahu rozšírenia	áno
používanie viacerých miest rozšírení vo vzťahu rozšírenia	nie
používanie rozšírenia určitým tokom	áno
používanie podmienok pre rozšírenie tokom	áno
používanie poradia vykonávania pre rozšírenie tokom	áno
používanie vzťahu zahrnutia	áno
používanie podmienok pre zahrnutie toku	nie
používanie zahrnutia určitého toku	nie
ďalšie časti opisu prípadov použitia	krátky popis, špeciálne požiadavky
špeciálne kroky	prípad použitia skončí, prípad použitia pokračuje, názov zahrnutého prípadu použitia, bod opätovného spojenia

6.2 Cockburnova notácia

Alistair Cockburn je priekopníkom modelovania prípadov použitia pomocou textového opisu prípadov použitia. V svojej, pre túto prácu najvýznamnejšej, knihe [9] poskytuje čitateľovi prehľad prakticky celej štruktúry textového opisu prípadov použitia a na množstve príkladov z praxe vysvetľuje koncept modelovania prípadov použitia bez použitia diagramov. Opäť bolo využitých niekoľko príkladov častí opisu prípadov použitia [9], aby bolo možné zostaviť profil pre jeho prístup k modelovaniu prípadov použitia:

Prípad použitia: Uprav dokument
Oblasť: Wapp
Úroveň: Používateľský cieľ
Spúšťač: Používateľ spustí aplikáciu.
Vstupná podmienka: Žiadna.

Hlavný úspešný scenár:

1. Používateľ otvorí dokument za účelom úpravy.
2. Používateľ vloží a upraví text.

...
Používateľ uloží dokument a ukončí aplikáciu.

Prípad použitia: Skontroluj pravopisu
Oblasť: Wapp
Úroveň: Podfunkcia
Spúšťač: Hocikedy v úprave dokumentu, keď je dokument otvorený a používateľ zvolí kontrolu pravopisu.
Vstupná podmienka: Dokument je otvorený.

Hlavný úspešný scenár:

... atď. ...

Cockburnov prístup k modelovaniu prípadov použitia je špecifický, pretože je zástanca čisto textovej reprezentácie a podľa neho je zaoberanie sa kreslením diagramov zbytočným mrhaním energiou [9]. Jeho prístup sa odlišuje od iných prístupov, ktoré používajú na modelovanie prípadov použitia diagramy aj text, hlavne v modelovaní vzťahov medzi prípadmi použitia. V dvoch príkladoch, ktoré boli uvedené vyššie, sú časti rozširovaného a rozširujúceho prípadu použitia. Zaujímavá je v nich reprezentácia vzťahu rozšírenia, kde sa rozširuje prípad použitia Uprav dokument implicitne hlavným tokom prípadu použitia kontrola pravopisu, pričom podmienka a miesto rozšírenia je uvedená v spúšťači. Spúšťač (trigger) prípadu použitia je impulz alebo udalosť, ktorá iniciuje prípad použitia [4]. Miesto rozšírenia je reprezentované v uvedenom príklade slovami „hocikedy v úprave dokumentu“, čo je ekvivalentné dvojkrokovému bodu rozšírenia, ktorý určuje oblasť medzi prvým až posledným krokom. Problém je však v tom, že nemožno povedať, že ide o bod rozšírenia v tom zmysle, ako je definovaný v UML alebo navrhnutom metamodeli, čiže miesto v správaní rozširovaného prípadu použitia, kde toto správanie môže byť rozšírené správaním iného prípadu použitia, tak ako to je špecifikované vzťahom rozšírenia [11]. Najlepším spôsobom reprezentácie bude teda zakázanie používania vzťahu rozšírenia a využitie iných možností modelu v podobe častí opisu prípadov použitia.

Prípad použitia: Skontroluj pravopis

Hlavný úspešný scenár:

...

3. Systém prechádza cez dokument a overuje každé slovo v slovníku.
4. Systém odhalí pravopisnú chybu, vyznačí slovo a ponúkne výber alternatívy používateľovi.
5. Používateľ vyberie jednu možnosť na nahradenie slova. Systém nahradí vyznačené slovo používateľským výberom nahradenia.

...

Rozšírenia:

...

4a. Systém neodhalí žiadne pravopisné chyby v celom dokumente.
    4a1 Systém notifikuje používateľa, prípad použitia skončí.
5a. Používateľ sa rozhodne ponechať originálne slovo.
    5a1. Systém ponechá slovo a pokračuje.
5b. Používateľ napíše nové slovo, ktoré nie je v zozname.
    5b1. Systém opäť overí pravopis, návrat na krok 3.

...

Alternatívne toky, podtoky a vzťah zahrnutia majú rovnakú reprezentáciu ako v Jacobsonovej notácii, pričom alternatívne toky a podtoky majú iba inú formálnu reprezentáciu, ktorú je možné vidieť v druhej dvojici príkladov uvedenej vyššie. V Cockburnovom prístupe je alternatívny tok je nepomenovaný a označený číslom kroku, v ktorom môže alternatívny krok rozšíriť hlavný tok udalostí. Ak sa v jednom kroku hlavného toku udalostí vyskytuje viac alternatívnych tokov, pripája sa k označeniu rozlišovací znak. Podmienka je uvedená pri označení alternatívy a ďalej nasledujú samotné kroky toku udalostí. Cockburn pripúšťa aj existenciu alternatívnych tokov alternatívnych tokoch a podobne. Podtoky formálne reprezentuje prostredníctvom podrobnejšieho číslovania, podobne, ako keď sa číslujú vnorené časti textu. Profil pozostávajúci z navrhnutých možností pre Cockburnov prístup je uvedený v tabuľke nižšie.

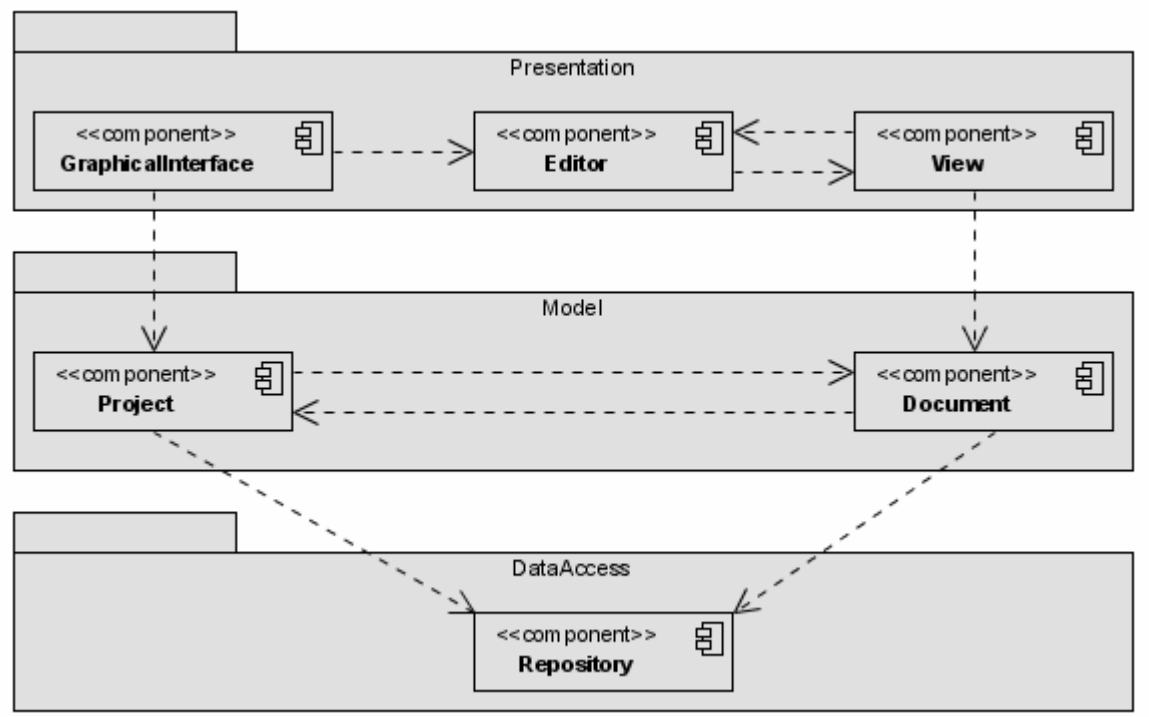
Tab. 3: Profil pre notáciu Alistaira Cockburna

Možnosť	Hodnota
používanie jednokrokových bodov rozšírenia	nie
používanie dvojkrokových bodov rozšírenia	nie
používanie aspoň jedného hlavný tok udalostí	áno
používanie viacerých hlavné toky udalostí	nie
používanie podtokov	áno
používanie podtokov v hlavných tokoch	áno
používanie podtokov v alternatívnych tokoch	áno
používanie podtokov v podtokoch	áno
používanie alternatívnych tokov	áno
používanie alternatívnych tokov v hlavných tokoch	áno
používanie alternatívnych tokov v alternatívnych tokoch	áno
používanie alternatívnych tokov v podtokoch	áno
používanie vzťahu rozšírenia	nie
používanie viacerých miest rozšírení vo vzťahu rozšírenia	nie
používanie rozšírenia určitým tokom	nie
používanie podmienok pre rozšírenie tokom	nie
používanie poradia vykonávania pre rozšírenie tokom	nie
používanie vzťahu zahrnutia	áno
používanie podmienok pre zahrnutie toku	nie
používanie zahrnutia určitého toku	nie
ďalšie časti opisu prípadov použitia	predmet, úroveň, spúšťač, minimálna garancia, garancia úspechu, kontext použitia, zainteresovaný účastníci a záujmy, zoznam technologických a dátových variácií
špeciálne kroky	prípad použitia skončí, prípad použitia pokračuje, názov zahrnutého prípadu použitia, bod opätovného spojenia

7 Návrh nástroja na modelovanie prípadov použitia

Na vypracovanie analýzy podpory modelovania prípadov použitia boli otestované viaceré populárne nástroje na modelovanie prípadov použitia s cieľom ich kategorizácie a identifikovania ich slabých aj silných stránok. Získané poznatky boli zužitkované pri návrhu jednoduchého nástroja na modelovanie prípadov použitia založeného na navrhnutom metamodeli a jeho profilovaní, ktorý by mohol byť použitý aj v praxi. Návrh pozostáva z návrhu grafického rozhrania, návrhu logického modelu údajov a návrhu editora, ktorý je tvorí podstatnú časť návrhu nástroja a je preto ďalej rozčlenený. Špecifickou črtou návrhu editora je navyše jeho implementačne nezávislý charakter a teda sa editor dá implementovať podľa tohto návrhu napríklad v programovacích jazykoch C++, C# alebo Java.

7.1 Architektúra nástroja



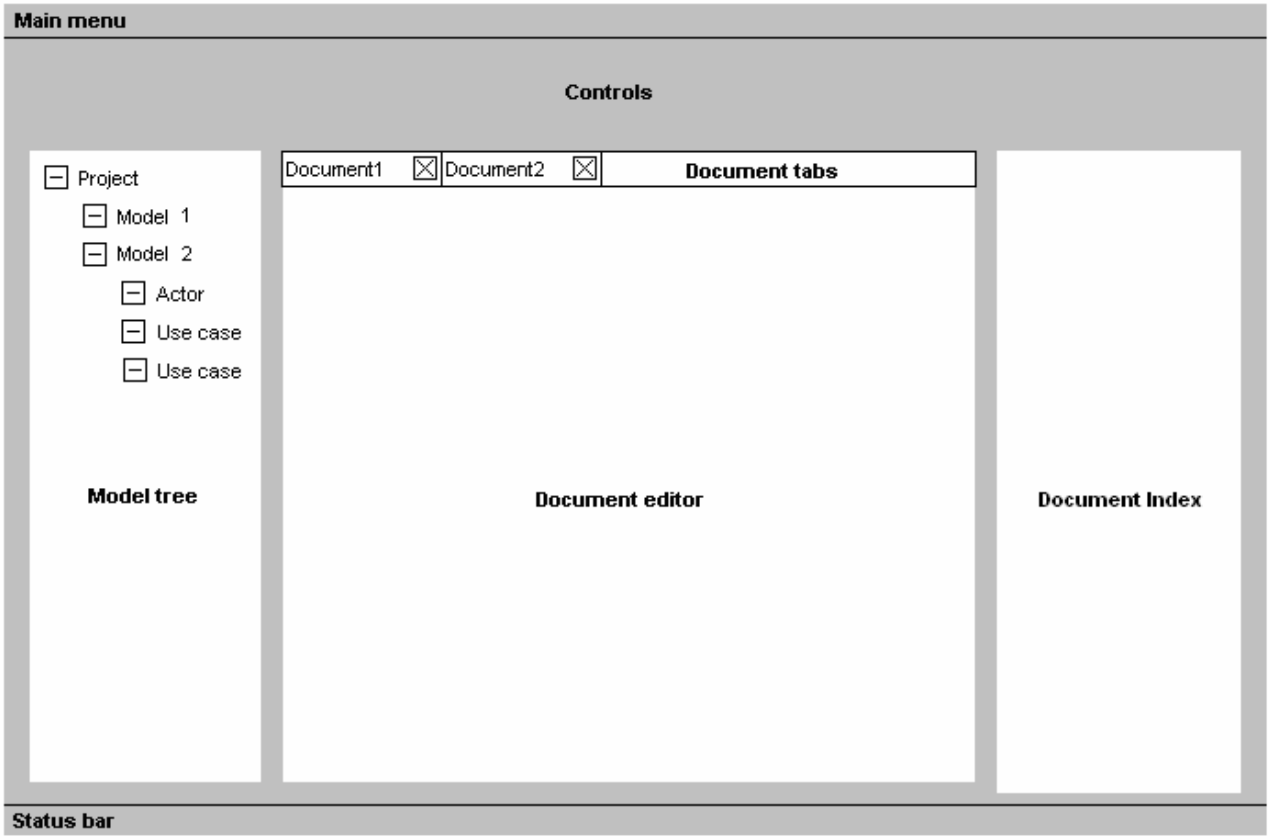
Obr. 13: Návrh architektúry

Na obrázku 13 je návrh architektúry nástroja na modelovanie prípadov použitia. Architektúra je založená na MVC, kde model je projekt(Project) a dokument(Document), view je editor a pohľad a controller je zahrnutý v grafickom rozhraní(GraphicInterface). Grafické rozhranie, editor(Editor) a pohľad(View) tvoria prezentačnú vrstvu celej architektúry, pretože spolu zobrazujú celý model a zmeny v ňom. Editor a pohľad zobrazujú dokument, zatiaľ čo grafické rozhranie projekt a jeho ďalšie časti. Pohľad využíva funkcie editor na vykresľovanie dokumentu na plochu editora. Na ukladanie častí modelu do databázy slúži repozitár(Repository), ktorý zabezpečuje komunikáciu s databázou

7.2 Grafické rozhranie

V predchádzajúcej časti bola navrhnutá architektúra nástroja. Podľa návrhu by malo grafické rozhranie obsahovať editor dokumentov a mal zobrazovať projekt a jeho súčasti. Na vhodné

ovládanie grafického editora ako aj podliehajúceho opisu prípadu použitia je potrebné navrhnúť vhodné grafické rozhranie a spôsob ovládania jeho súčastí.

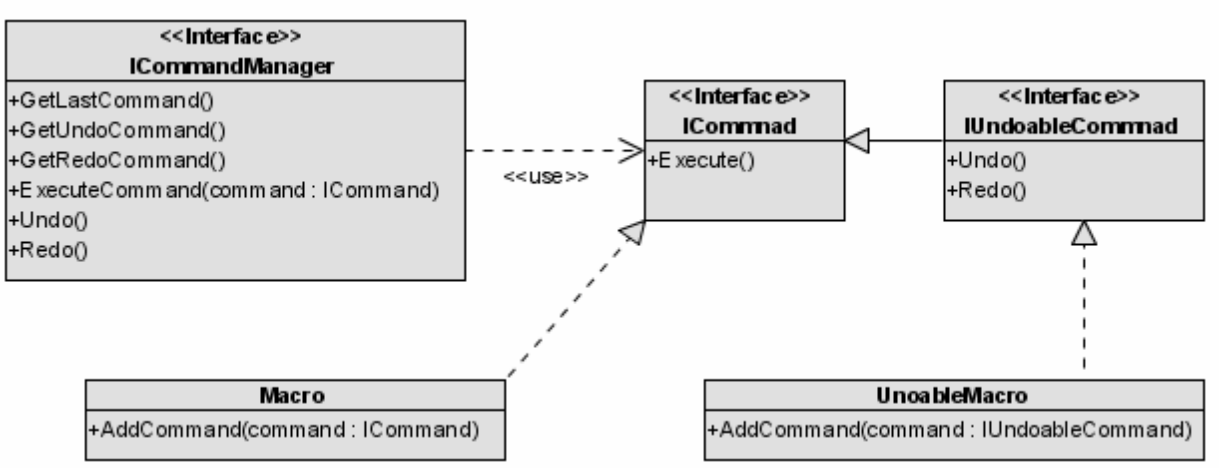


Obr. 14: Návrh grafického rozhrania

Základom grafického rozhrania je editor prípadov použitia, nad ktorým sú umiestnené záložky s otvorenými dokumentmi prípadov použitia. Rovnaký základ majú aj editory zdrojového kódu, ktoré týmto spôsobom ponúkajú používateľovi rýchly prístupu k nedávno upravovaným dokumentom. Takýto prístup je výhodné použiť aj v grafickom rozhraní editora modelu prípadov použitia, pretože sa určite osvedčí pri úprave dokumentov prípadov použitia, ktoré obsahujú vzťahy rozšírenia a zahrnutia.

Navrhnuté grafické rozhranie ďalej obsahuje strom projektu rozdeleného na modely obsahujúce účastníkov a prípady použitia, ktoré bude možné ľahko sprístupniť na editovanie. V spodnej časti grafického rozhrania sa bude zobrazovať pomocný text k riadiacim prvkom na úpravu dokumentu. Nad editorom je priestor, kde budú tieto riadiace prvky na úpravu dokumentu a úplne hore bude hlavné menu.

Z hľadiska ovládania je potrebné, aby aplikácia dokázala niektoré úpravy modelu prípadov použitia vrátiť do predchádzajúceho stavu a naopak. Ide napríklad o úpravy textu vkladanie alebo vymazávanie krokov a podobne. Ostatné akcie v rámci aplikácie ako uloženie dokumentu alebo nahranie dokumentu sa typicky vrátiť do predchádzajúceho stavu nedajú a preto k nim treba pristupovať inak. Z toho vyplýva, že aplikácia by mala podporovať dva typy akcií, ktoré nazveme vratné a nevratné. Na reprezentovanie oboch typov akcií sme navrhli prístup, ktorý je založený na návrhovom vzore príkaz (command).



Obr. 15: Návrh príkazov

Rozhranie ICommand je základom celého návrhu príkaz. Obsahuje iba metódu Execute, ktorej implementáciou by malo byť vykonanie príkazu. Od ICommand dedí rozhranie IUndoableCommand a trieda, ktorá toto rozhranie implementuje musí implementovať aj metódy Undo a Redo. Triedy Macro a UndoableMacro by sa mali používať pri zložených príkazoch. Obe implementujú AddCommand, ktorým sa pridávajú príkazy do reťaze príkazov. Metóda Undo triedy UndoableMacro vykonáva Undo metódy príkazov v opačnom poradí, v akom boli pridané.

Návrh ešte obsahuje rozhranie pre implementáciu manažéra príkazov ICommandManager. Manažér príkazov by mal vykonávať príkazy, udržiavať históriu vratných príkazov a správne podľa nej vykonávať operácie Undo a Redo.

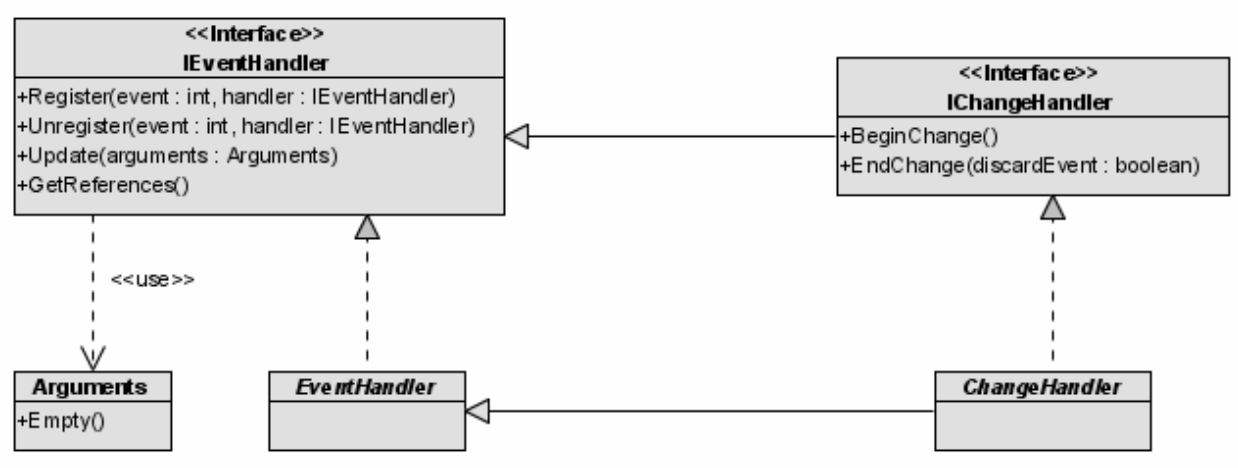
7.3 Editor opisu prípadov použitia

Editor opisu prípadov použitia bude základnou časťou nástroja, pretože prostredníctvom neho bude realizované celé modelovanie prípadov použitia Základnou vlastnosťou editora, ktorú musí editor spĺňať, je práca s textom vo forme krokov toku udalostí. Editor teda musí v každom okamihu vedieť poradie krokov, aby ich vedel samostatne číslovať a tak isto musí vedieť, ku ktorému toku udalostí kroky prislúchajú, pretože od toho sa môže odvíjať formát číslovania. Okrem toho by mal editor podporovať hlavne:

- formátovaný text, editovateľné objekty a textové referencie
- text určený iba na čítanie, aby bolo možné vyvárať napríklad needitovateľné názvy sekcií dokumentov prípadov použitia
- bežné textové operácie a operácie na prácu s krokmi a tokmi udalostí
- načítanie a uloženie vytvorených modelov prípadov použitia do databázy

7.3.1 Reakcie na zmeny

Jednou z najdôležitejších požiadaviek, ktoré by mal navrhovaný nástroj spĺňať je automatická údržba dokumentov prípadov použitia na základe zmien v údajov v modeli nad ktorým bude pracovať. Ako príklad je možné uviesť premenovanie účastníka, ktoré sa musí automaticky prejaviť v každom dokumente, kde je účastník uvedený. Z tohto dôvodu bolo potrebné navrhnúť vhodný mechanizmus udalostí.



Obr. 16: Návrh mechanizmu udalostí

Návrh vychádza z návrhového vzoru pozorovateľ (observer), ale ako je vidieť na obrázku, pozorovateľ je zároveň aj subjekt, aby sa mohli pozorovať navzájom objekty alebo aby mohli vytvoriť reťaz objektov, ktoré implementujú rozhranie IEventHandler. Pozorovatelia sa registrujú u subjektov spolu s identifikátorom udalosti, o ktorú sa zaujímajú . Špeciálnym typom subjektu by mal byť subjekt, ktorý implementuje rozhranie IChangeHandler. Subjekt by mal v tom prípade poskytovať na registráciu ďalšiu udalosť, ktorá reprezentuje všetky ostatné udalosti spolu. To znamená, že ak nastane normálna udalosť nastane aj špeciálna udalosť, ktorá však môže byť pozdržaná pomocou metódy BeginChange a opätovne uvoľnená pomocou metódy EndChange, pokiaľ sa nerozhodne, že udalosť treba zadržať úplne prostredníctvom discardEvent.

Špeciálna udalosť a jej pozdržanie slúžia nato, aby bolo možné riadiť, kedy sa všetky vykonané zmeny prejavia. Dôležité to je hlavne pri vykresľovaní dokumentu prípadu použitia, keď je potrebné vykonať napríklad prečíslovanie krokov. Pri prečíslovaní krokov by došlo vždy k prekresleniu obrazovky alebo jej časti zbytočne pri každej zmene, čo by mohlo na byť na starších počítačoch za určitých podmienok viditeľné voľným okom.

Na obrázku je ešte možné vidieť triedu Arguments, od ktorej sa dajú odvodiť triedy argumentov posielaných, keď nastane udalosť. Jej jediná metóda Empty by mala vrátiť objekt alebo hodnotu, ktorá reprezentuje prázdne argumenty. Okrem triedy Arguments toho sú na obrázku ešte abstraktné triedy EventHandler a ChangeHandler, ktoré implementujú príslušné rozhrania. Tieto triedy by mali slúžiť ako základ pre všetky triedy, ktoré budú buď pozorovateľmi alebo subjektmi. Dôvodom návrhu rozhraní aj abstraktných tried je to, že niektoré programovacie jazyky nepodporujú dedenie od viacerých tried. Navyše by mohlo byť pri prípadnom rozširovaní navrhnutého nástroja výhodne implementovať IChangeHandler tak, že by sa počas zadržania udalosti zbierali argumenty z normálnych udalostí, ktoré by sa po uvoľnení odoslali pozorovateľovi na spracovanie naraz. Pri takejto implementácii by nebolo potrebné rozdelenie udalostí, ale na druhej strane by bolo potrebné implementovať aj mechanizmus na spracovanie argumentov.

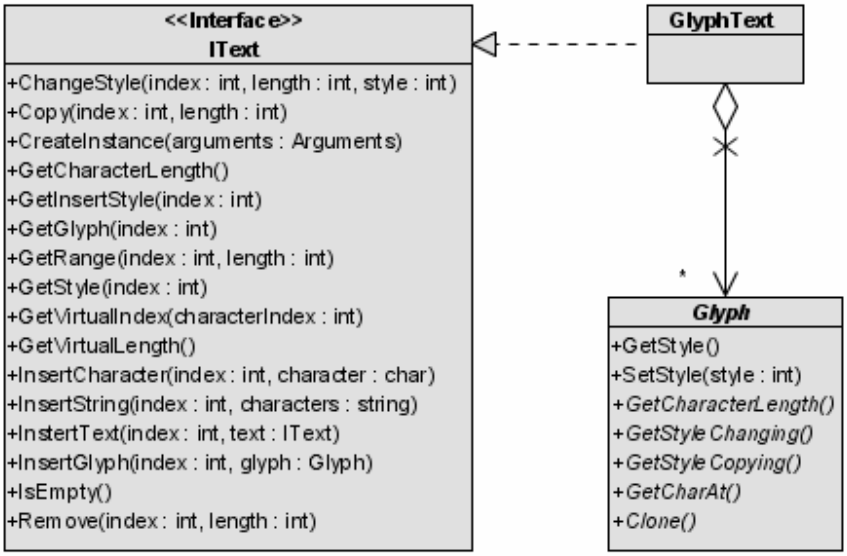
7.3.2 Reprezentácia formátovaného textu

Na reprezentáciu formátovaného textu s možnosťou vkladania špeciálnych objektov a textových referencií sa používajú dva prístupy hlavné prístupy. Prvým prístupom je vkladanie formátovania, objektov a referencií priamo do textu vo forme špeciálnych reťazcov. Ako príklad toho prístupu sa dajú uviesť formáty RTF, XML, HTML. Výhodou tohto prístupu je jednoduché prenášanie údajov po sieti, na médiách a podobne, pretože text aj formát je neustále spolu. Nevýhodou je však to, že editor pracujúci priamo nad touto reprezentáciou by musel pri každej zmene znova spracovať celý

text alebo jeho časť. Druhým prístupom k reprezentácii formátovaného textu je oddelenie textu od formátovania. To znamená, že bude existovať čistý text a formátovanie sa bude vzťahovať na časti bloky čistého textu. Výhoda tohto prístupu je v tom, že pri zmene formátu netreba zasahovať vôbec do textu a zároveň zmeny v texte vyvolajú iba lokálne úpravy vo formátovaní. Nevýhodou je však to, že pri takto reprezentovanom formátovanom text je pri zápise do databázy alebo do súboru potrebné konvertovať formátovanie na text napríklad vo formáte XML.

Pre textový editor bol zvolený práve druhý prístup k reprezentovaniu formátovaného textu, pretože je potrebné, aby editovanie, hlavne písanie textu, malo čo najmenšiu odozvu inak by implementovaný editor mohol blikať alebo pomaly vykresľovať písaný text. Extra čas, ktorý sa získa použitím tohto prístupu pri prekresľovaní obrazovky, môže byť neskôr požitý na rôzne doplnkové vlastnosti editora ako napríklad kontrolu pravopisu.

Problém reprezentácie formátovaného textu sa zvolením druhého prístupu rozpadol na dva ďalšie problémy a to reprezentáciu čistého textu a reprezentáciu formátovania, ktoré sú skôr problémami implementačnými a preto nie sú ďalej v tejto časti práce rozobrané. Napriek tomu, že bol zvolený jeden prístup, návrh je prispôsobený tak, aby bolo možné toto rozhodnutie jednoducho zmeniť v prípade potreby.



Obr. 17: Návrh reprezentácie formátovaného textu

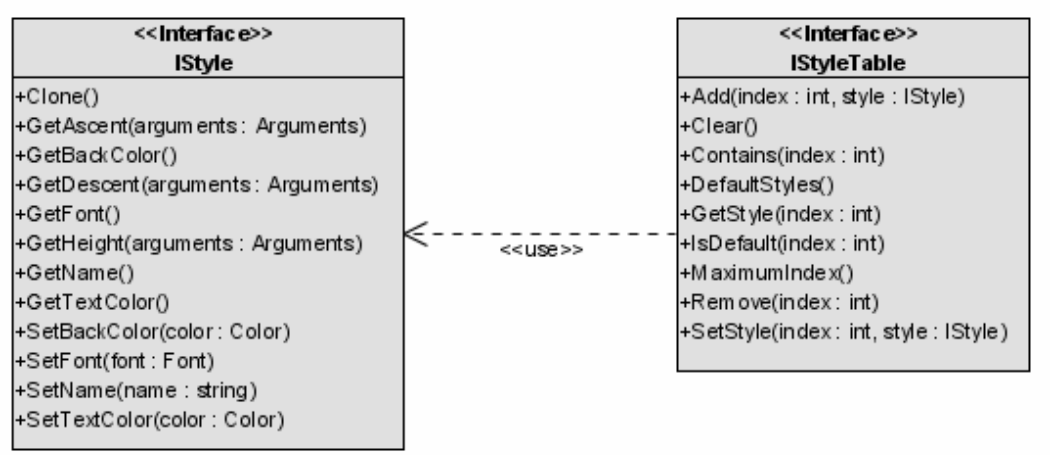
Základom návrhu textovej reprezentácie je abstraktná trieda Glyph, ktorá obsahuje iba dve nie abstraktné metódy GetStyle a SetStyle slúžiace na manipuláciu so štýlom textu. Štýl je na tejto úrovni iba celé číslo, ktoré je indexom pre tabuľku štýlov. Ostatné metódy sú abstraktné a teda musia byť odvodenými triedami implementované. Najdôležitejšie z týchto metód sú metódy GetCharAt a GetCharacterLength, pretože objekt triedy odvodenej od Glyph môže obsahovať aj skupinu znakov a prostredníctvom týchto metód by sa malo dať k jednotlivým znakom pristúpiť. Typickým príkladom využitia môže byť referencia, pri ktorej je žiaduce, aby bola reprezentovaná jedným objektom a nebolo potrebné pri jej zmene meniť viaceré objekty.

Formátovaný text je reprezentovaný postupnosťou objektov tried odvodených od Glyph. Trieda GlyphText implementuje rozhranie IText a teda predstavuje jednu možnú implementáciu formátovaného textu. Rozhranie IText obsahuje šestnásť metód z čoho väčšina slúži na manipuláciu s textom. Pre všetky metódy na editovanie formátovaného textu je dôležité, aby pracovali

s virtuálnym indexom, ktorý určuje pozíciu objektu triedy odvodenej od Glyph vo formátovanom texte. Na prevod znakového indexu na virtuálny index by mala slúžiť metóda GetVirtualIndex. Pri vkladaní neformátovaného textu do formátovaného prostredníctvom metód InsertCharacter a InsertString je potrebné zistiť štýl, ktorý má byť neformátovanému textu priradený. Na tento účel by mala slúžiť metóda GetInsertStyle, ktorá môže využiť metódu GetStyleCopying na zistenie, či môže alebo nemôže priradiť vkladanému textu formát objektu pred virtuálnym indexom. GetStyleChanging by zasa mala byť využívaná metódou ChangeStyle na zistenie, či je možná zmena štýlu objektu.

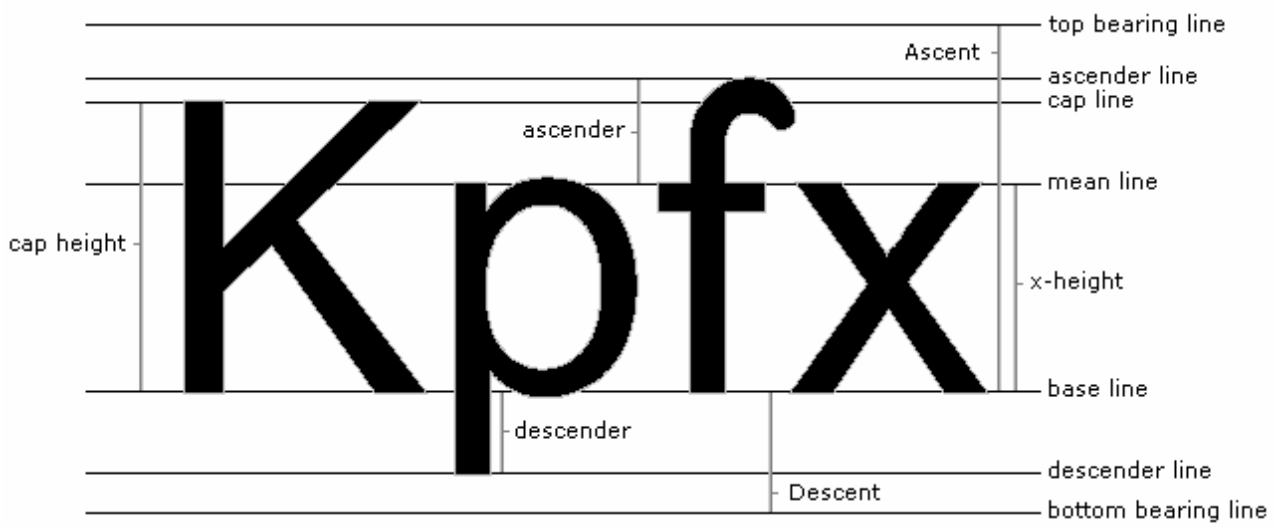
7.3.3 Štýly formátovaného textu

Štýl vo formátovanom texte určuje ako vizuálne má byť text vykreslený. V rámci formátovaného textu môže mať aj každé písmeno iný štýl a preto je potrebné zvoliť vhodnú reprezentáciu štýlov. Keďže bola zvolená reprezentácia formátovaného textu tak, aby bolo formátovanie oddelené od textu, najvhodnejšou reprezentáciou štýlov je index do tabuľky štýlov, kde sú uložené všetky rôzne štýly použité v dokumente. Výhodou tejto reprezentácie je aj to, že je možné mať statickú tabuľku štýlov, čo sa dá pri dokumentoch prípadov použitia v tom istom projekte využiť



Obr. 18: Návrh rozhraní pre reprezentáciu štýlov

Tabuľka štýlov by mala implementovať rozhranie IStyleTable. Rozhranie je navrhnuté tak, aby implementovaná tabuľa bola rozdelená ne predvolené a užívateľom definované štýly. Predvolené štýly by mali byť v tabuľke prítomné stále a preto je potrebné brať túto skutočnosť do úvahy pri implementácii metódy Remove. Pri implementácii metódy Add je zasa potrebné kontrolovať maximálny index, ktorý je možné použiť. Každý štýl musí byť uložený pod indexom, ktorý sa nesmie v tabuľke zmeniť.

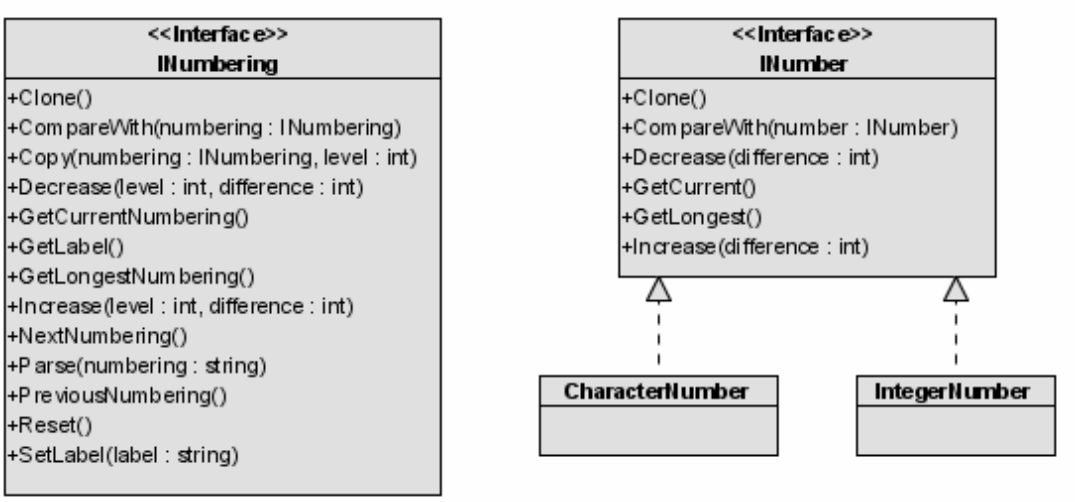


Obr. 19: Metriky druhu písma v operačnom systéme Windows

Každý štýl, teda objekt triedy implementujúcej rozhranie IStyle, by mal mať meno(GetName), druh písma(GetFont), farbu písma(GetTextColor) a farbu zvýraznenia(GetBackColor). Dôležitými metódami štýlu sú GetAscent, GetDescent a GetHeight, ktorými sa dajú získať základné metriky potrebné na vykreslenie riadkov textu obsahujúcich rôzne druhy písma. Tieto metriky môžu byť závisle od nastavenia plochy, na ktorú je potrebné text vykresliť a preto môžu mať všetky tri metódy argumenty.

7.3.4 Číslovanie

Mechanizmus číslovania je v navrhovanom nástroji potrebný pri číslovaní krokov a tokov udalostí. Na tento účel bolo navrhnuté viacúroňové číslovanie, kde každá úroveň môže byť iná. Základom je rozhranie INumber, ktoré určuje ako implementovať jednu úroveň číslovania. Základnými implementáciami tohto rozhrania sú triedy CharacterNumber a IntegerNumber teda číselná a znaková úroveň číslovania. Trieda reprezentujúca viacúrovňové číslovanie by mala implementovať rozhranie INumbering. Pri implementácii INumbering, konkrétne metódy GetCurrentNumbering, je možné rozhodnúť o tom aké úrovne číslovania a aké oddeľovače číslovania budú použité. Rozhranie poskytuje aj možnosť implementácie označenia pred číslovaním, čo je možné využiť na označenie typu toku udalostí. Pre vykreslenie číslovania je dôležitá metóda GetLongestNumbering, ktorá môže využiť metódu GetLongest úrovní číslovania, na vytvorenie najdlhšieho možného číslovania, aby sa napríklad text číslovaných krokov v tokoch udalostí dal zvisle zarovnať.



Obr. 20: Návrh reprezentácie číslovania

7.3.5 Dokument a riadky

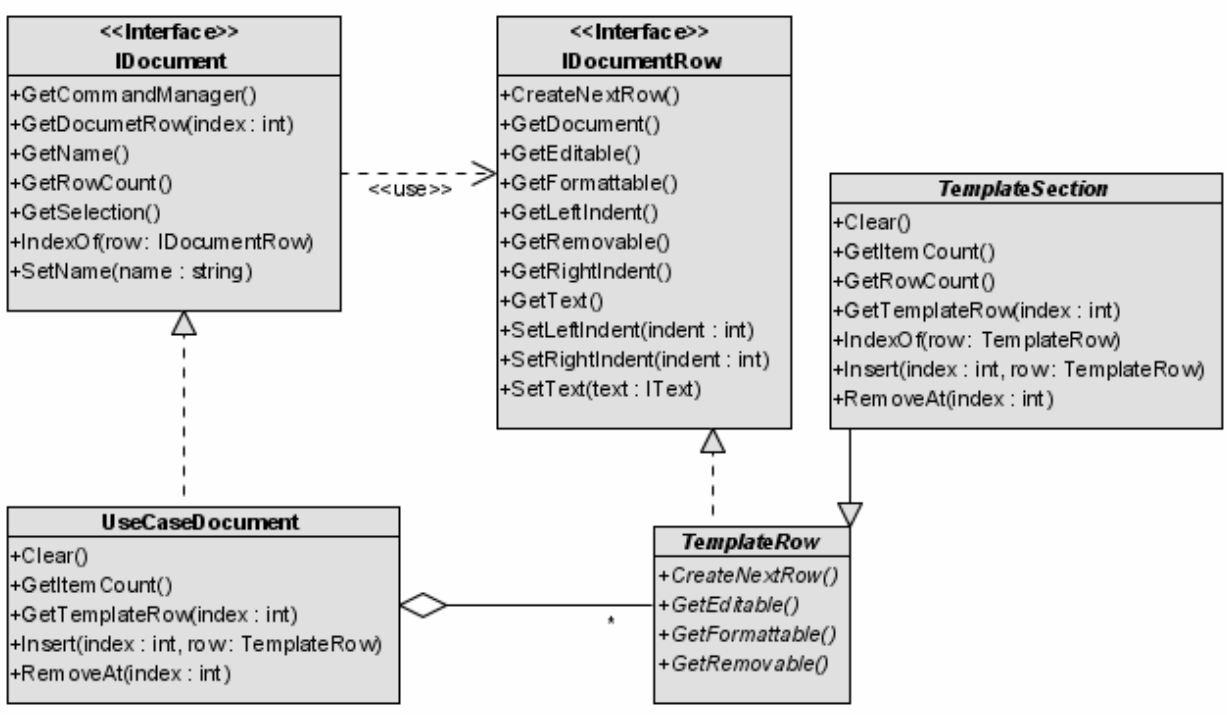
Každý prípad použitia bude v rámci editora reprezentovaný dokumentom prípadu použitia. Na realizáciu podpory automatického číslovania a prečíslovanie bolo potrebné navrhnúť vhodnú reprezentáciu dokumentu. Návrh reprezentácie dokumentu vychádza z toho, že kroky a názvy tokov neobsahujú nový riadok a teda ide vlastne o paragrafy textu, čo sa dá veľmi dobre využiť pri ich číslovaní a prečíslovaní.

0		Alternatívne toky
0		
0	A1.	Duplicitná rezervácia
0		Ak v kroku 5 základného toku existuje identická rezervácia v systéme (rovnaké meno, email, počiatočný a konečný dátum), systém zobrazí existujúcu rezerváciu a opýta sa zákazníka či chce pokračovať s novou rezerváciou.
1	1.	Ak zákazník chce pokračovať, systém pokračuje v rezervácii a prípad použitia pokračuje.
1	2.	Ak chce zákazník označiť novú registráciu ako duplicitnú, prípad použitia skončí.

Obr. 21: Reprezentácia dokumentu rozdelením na riadky

Reprezentácia dokumentu na obrázku 21 sa skladá z postupnosti riadkov, kde každý z nich obsahuje úroveň odsadenia, číslovanie a formátovaný text. Aj keď táto reprezentácia je pre dokument prípadov použitia veľmi dobrá rozdelenie dokumentu na riadky predstavuje aj určitý problém. Pri práci s viacerými riadkami je potrebné nielen získať označené riadky, ale aj označený text v nich, pretože riadok nemusí byť označený celý. Pri označení viacerých riadkov v dokumente platia nasledujúce tvrdenia:

- prvý označený riadok môže byť označený celý alebo jeho koncová
- posledný označený riadok môže byť označený celý alebo jeho počiatočná časť
- všetky riadky medzi prvým a posledným sú vždy celé označené



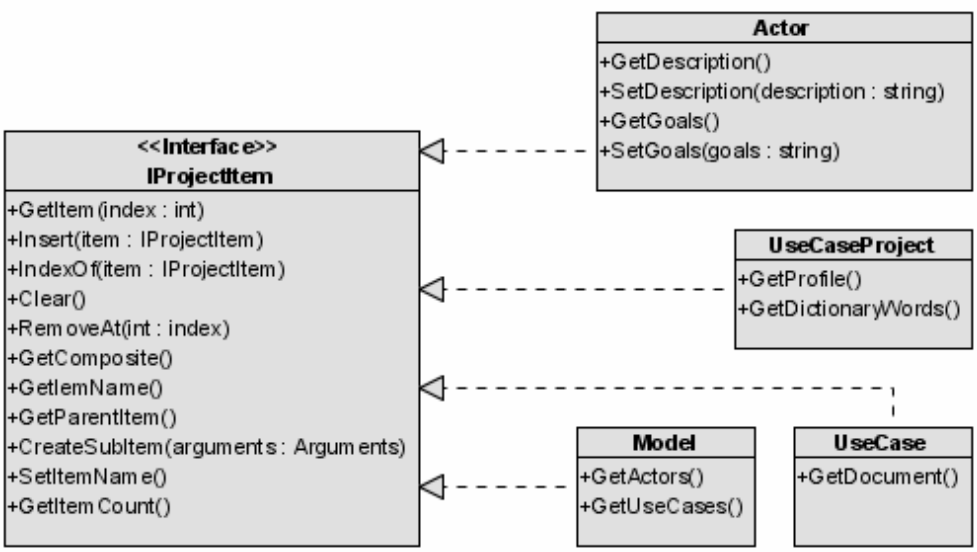
Obr. 22: Návrh reprezentácie dokumentu prípadov použitia

Navrhnutá reprezentácia dokumentu prípadov použitia je založená na dvoch rozhraniach IDocument a IDocumentRow, ktorých implementáciou možno vytvoriť základný dokument rozdelený na riadky. Pre dokument prípadov použitia to však nestačí a preto v navrhnutej reprezentácii vystupuje abstraktná trieda TemplateSection. Táto trieda je základom pre sekcie dokumentu, ktoré sú potrebné pri reprezentovaní napríklad tokov udalostí. Odvodená je od TemplateRow, pretože aj sekcia sa zobrazuje v dokumente vo forme nejakého nadpisu. TemplateRow je základným riadkom dokumentu prípadov použitia, od ktorého sú odvodené všetky ostatné riadky. Obsahuje tri abstraktné metódy GetEditable, GetFormattable a GetRemovable, ktoré určujú či je editovanie, formátovanie alebo odstránenie riadka pre používateľa povolené alebo zakázané. Štvrtou abstraktnou metódou je CreateNextRow, ktorá rozhoduje o tom aký riadok sa má vytvoriť po danom riadku a či vôbec sa má nejaký riadok vytvoriť.

Z návrhu vyplýva, že UseCaseDocument je postupnosť sekcií a riadkov alebo inak povedané UseCaseDocument je strom riadkov. Trieda UseCaseDocument však nie je zdedená od TemplateSection, pretože dokument nie je riadok aj keď za sekciu by sa mohol považovať. Obe triedy majú metódu na získanie všetkých riadkov GetRowCount ako aj metódu na získanie priamych potomkov GetItemCount, ktoré sú potrebné pri vykonávaní zmien v štruktúre dokumentu prípadov použitia.

7.3.6 Projekt a modely prípadov použitia

Dokument prípadov použitia reprezentuje v podstate jeden prípad použitia. Okrem prípadov použitia je potrebné navrhnúť reprezentáciu pre projekt, model prípadov použitia, účastníka. Model prípadov použitie je v tomto prípade iba súbor prípadov použitia a účastníkov, keďže vzťahy medzi prípadmi použitia navzájom sú zachytené v textovej podobe. Projekt je zasa súbor modelov, ktoré napríklad dokopy tvoria opis správania pre jeden systém.

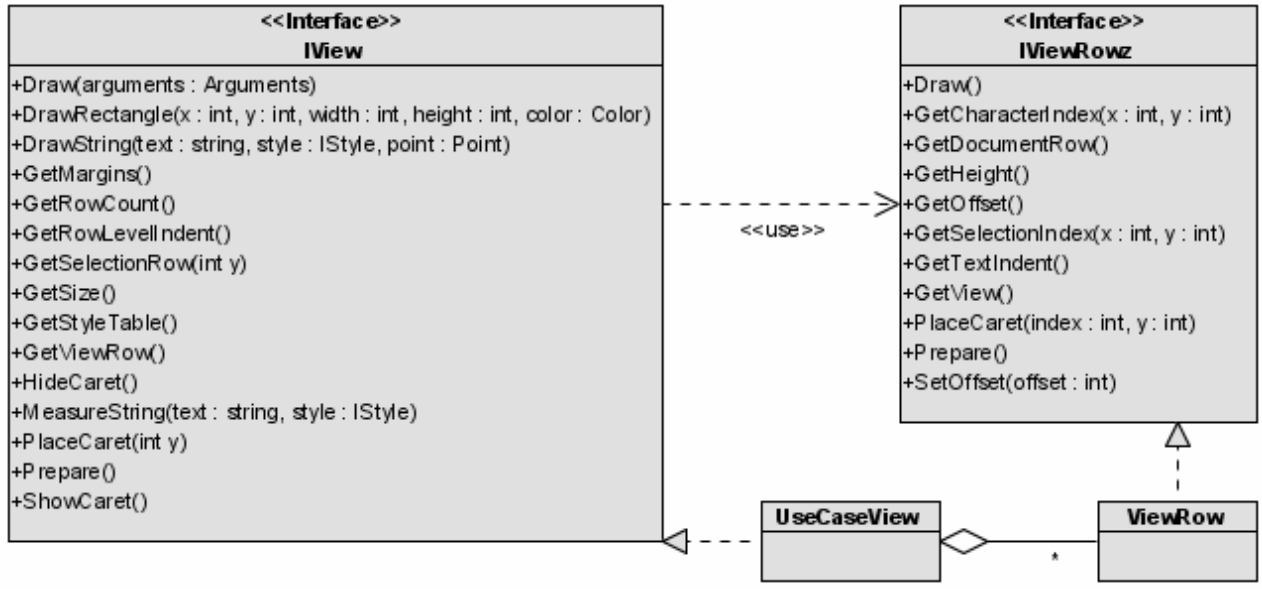


Obr. 23: Reprezentácia projektu

Projekt, ktorý obsahuje modely prípadov použitia obsahujúce prípady použitia a účastníkov, je možné reprezentovať pomocou stromu. Základom je rozhranie IProjectItem, ktoré implementujú všetky časti projektu. Vhodnou implementáciou CreateSubItem je možné vytvárať objekty prostredníctvom argumentov priamo v stromovej štruktúre. UseCaseProject má dve dôležité metódy GetProfile a GetDictionaryWords, ktoré slúžia na získanie prístup k profilu a k slovníku.

7.3.7 Pohľad a zobrazenie riadkov

Dokument, ktorý je rozdelený na riadky, je v podstate iba vnútorná reprezentácia úplne nezávislá na grafickom rozhraní. Na zobrazenie dokumentu a jeho riadkov bol navrhnutý pohľad, ktorý je taktiež rozdelený na riadky. Návrh spočíva v tom, že pohľad zobrazuje dokument tak, že riadky pohľadu zobrazujú riadky dokumentu.



Obr. 24:Návrh reprezentácie pohľadu

Základom reprezentácie pohľadu je rozhranie IView. Implementujúce triedy musia implementovať dve základné kresliace funkcie DrawString na vykresľovanie textu a DrawRectangle, ktorá je potrebná pri kreslení označeného textu. Okrem toho je potrebná aj implementácia metódy na

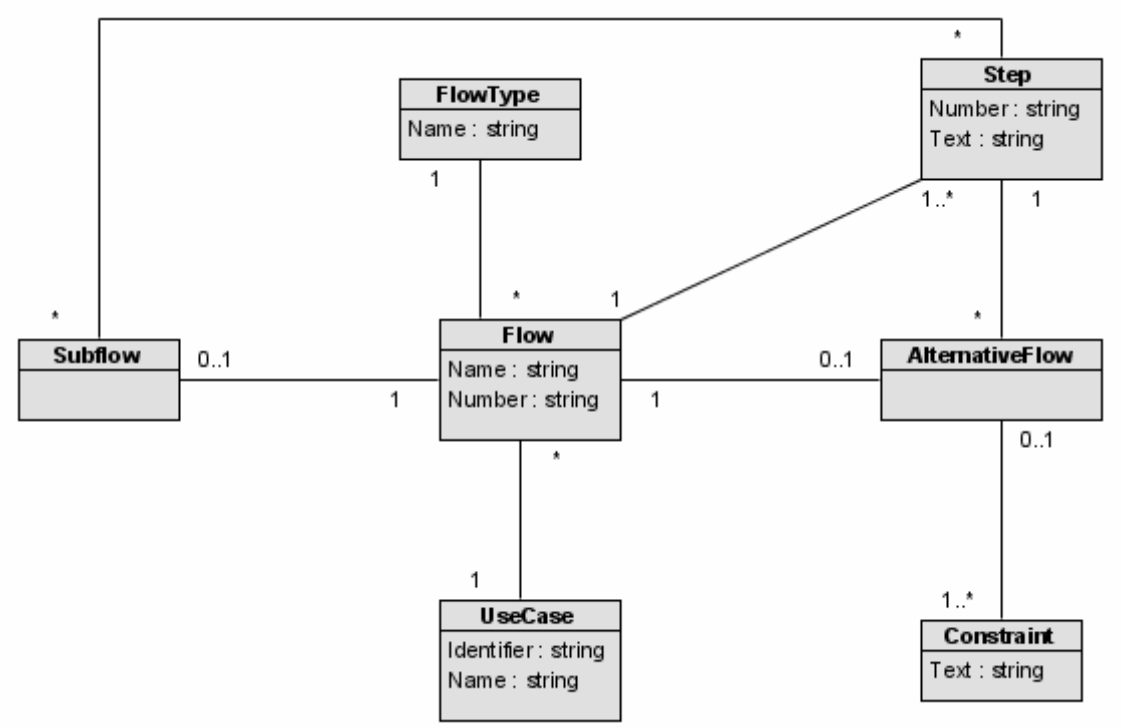
meranie reťazcov, aby ich bolo možné správne rozložiť na kresliacej ploche. Metódy ShowCaret, PlaceCaret a HideCaret by mali slúžiť na manipuláciu s kurzorom.

Pre riadky pohľadu bolo navrhnuté rozhranie IViewRow. Implementujúca trieda je zodpovedná za rozmiestnenie textu podľa nastavenia pohľadu a podľa riadka dokumentu. Keďže implementujúca trieda pozná algoritmus, bude vedieť aj previesť súradnice na znakový index. Získanie znakového indexu musí byť implementované v metóde GetSelectionIndex a GetCharacterIndex. GetSelectionIndex musí vrátiť znakový index aj keď súradnice sú mimo oblasti vykresleného textu, zatiaľ čo GetCharacterIndex musí vrátiť znakový index iba keď sú súradnice v rámci textu.

Vykresľovanie je rozvrhnuté na dve fázy. Vo fáze prípravy sa prostredníctvom metódy Prepare riadka pohľadu rozloží text, čo môže spôsobiť zmenu vo výške riadka a preto keď sú riadky pripravené pohľad ich opätovne rozloží prostredníctvom ich metódy SetOffset tak, ako majú podľa dokumentu nasledovať za sebou. V druhej fáze sa potom riadky vykreslia pomocou metódy Draw. Myšlienka rozdelenia na dve fázy spočíva v tom, že pokiaľ sa rozmiestňuje text a riadky, netreba prekresliť starý obsah kresliacej plochy alebo jej časti a prekreslenie sa bude zdať plynulejšie.

7.4 Logický model údajov

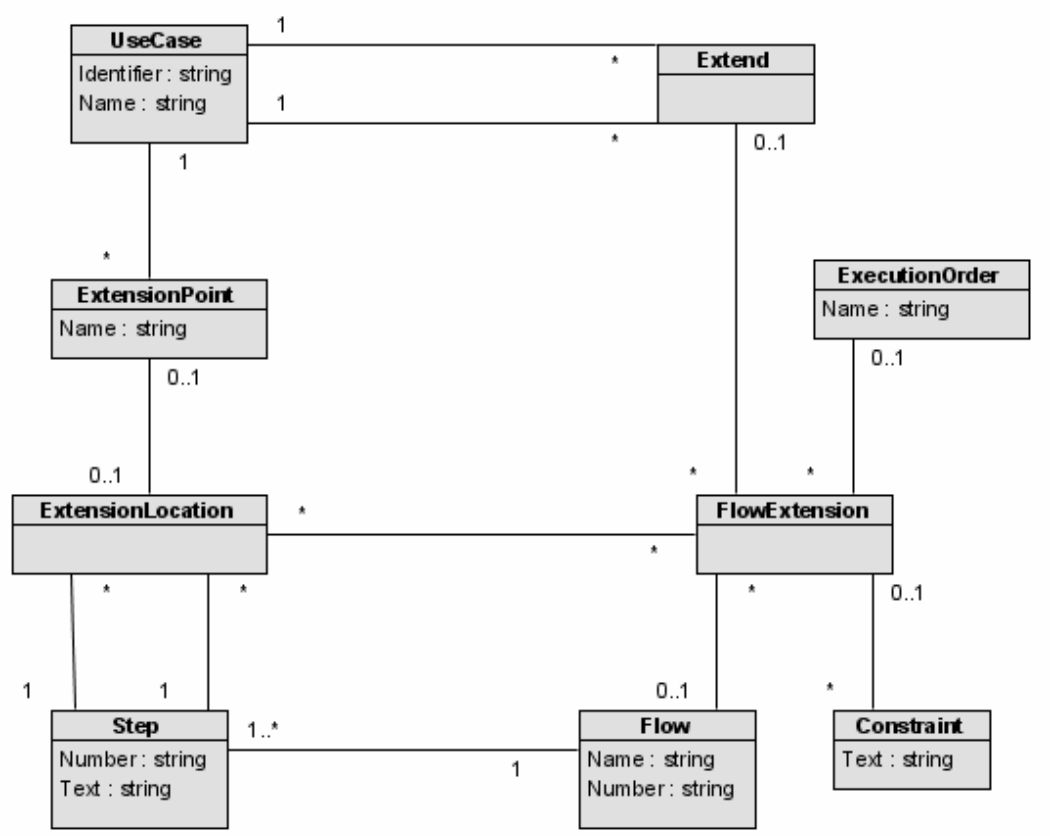
Dokument prípadu použitia je vlastne súbor údajov opisu prípadu použitia, ktoré bude používateľ nástroja prostredníctvom editora prípadov použitia upravovať. V časti 5 tejto práce bol navrhnutý metamodel modelovania prípadov použitia, ktorý je dobrým základom na vytvorenie logického modelu údajov. Z neho je možné získať základné údajové entity a vzťahy medzi nimi. Doplniť však treba ich atribúty, ktoré síce nie sú v metamodeli, ale mnohé z nich už boli spomenuté v rámci analýzy štruktúry opisu prípadov použitia.



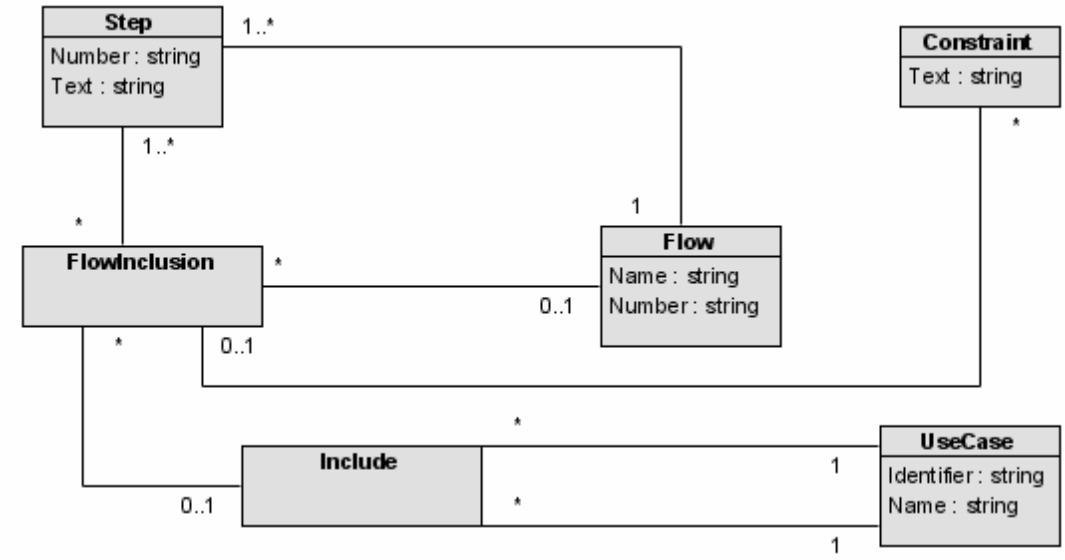
Obr. 25: Časť logického modelu pre toky udalostí

Na obrázku 25 je uvedená časť logického modelu pre toky udalostí. Oproti metamodelu je práve tu najväčšia zmena v vzťahoch medzi entitami kvôli zmene vzťahov zovšeobecnenia na asociácie. Hlavný tok udalostí je v logickom modeli reprezentovaný tokom udalostí, pretože obsahuje iba

kroky. Od ostatných tokov udalostí sa odlišuje prostredníctvom typu toku (FlowType). Každý tok má meno (Name) a číslo (Number) rovnako ako krok. Typ toku a prípad použitia obsahujú názvy, pričom prípad použitia má ešte identifikátor (Indentifier). Obmedzenie je reprezentovaná textom a preto má aj v logickom modeli iba atribút Text.



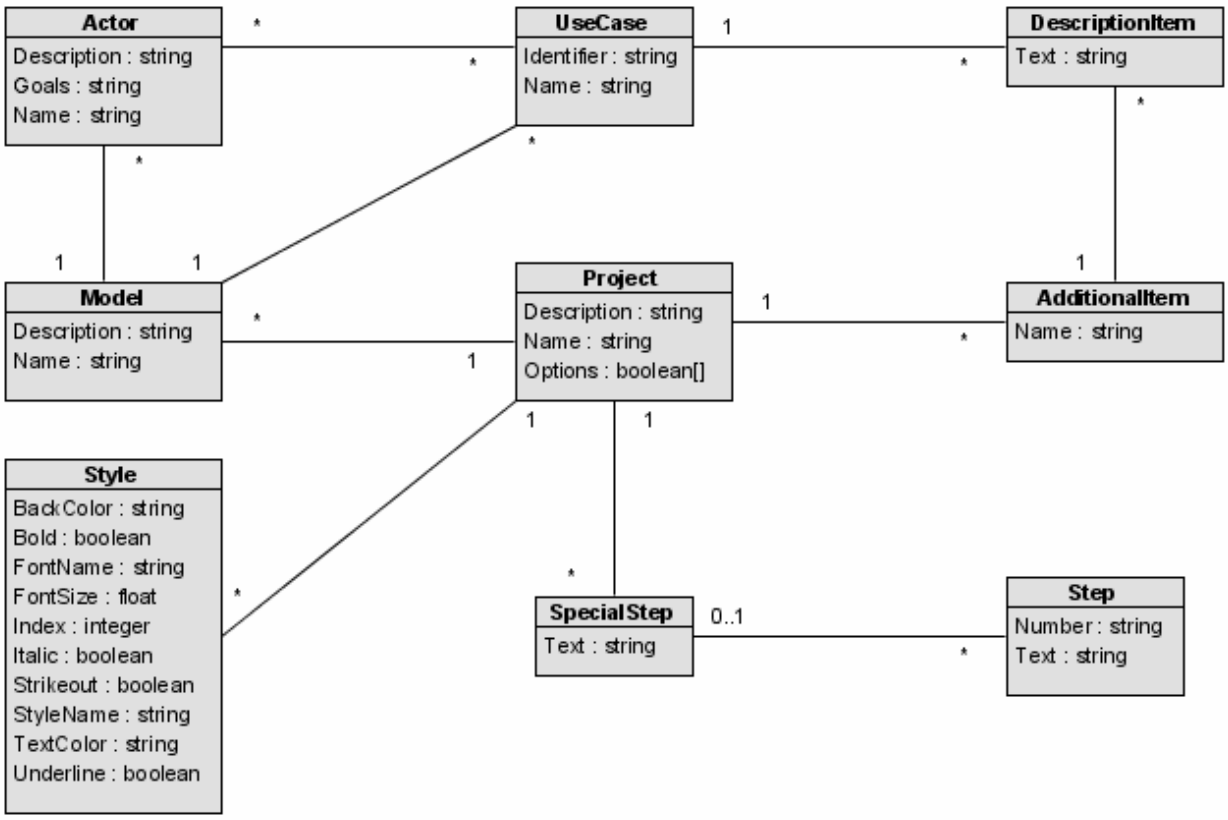
Obr. 26: Časť logického modelu pre bod rozšírenia a vzťah rozšírenia



Obr. 27: Časť logického modelu pre vzťah zahrnutia

V častiach logického modelu pre bod rozšírenia a vzťahy medzi prípadmi použitia pribudlo oproti časti logického modelu pre toky udalostí päť väzobných entít bez atribútov a to Extend,

ExtensionLocation, FlowExtension, Include a FlowInclusion. Okrem nich pribudli entity ExtensionPoint a ExecutionOrder, ktorých jedinou entitou je názov (Name).



Obr. 28: Časť logického modelu pre projekt a súvisiace časti

Základom poslednej časti logického modelu je projekt (Project). Projekt má atribúty meno, opis a nastavanie, ktoré sú reprezentované poľom booleovských hodnôt a predstavujú booleovské nastavenia v profile. Súčasťou profilu sú aj špeciálne kroky a ďalšie časti opisu reprezentované entitami SpecialStep a AdditionalItem, ktoré majú po jednom atribúte typu reťazec. S projektom je spojená aj entita Style reprezentujúca tabuľka štýlov, ktorá okrem bežných atribútov formátovaného textu obsahuje meno štýlu (StyleName) a index štýlu (Index). Poslednou entitou týkajúcou sa projektu je entita Model, ktorá reprezentuje model prípadov použitia. Model má atribúty meno (Name) a opis (Description) a môže obsahovať účastníkov (Actor) a prípady použitia (UseCase). Účastník má taktiež meno a opis a navyše aj ciele.

8 Implementácia

Prototyp bol implementovaný v jazyku C#, ktorý je veľmi dobrý na rýchle vytváranie aplikácii s grafickým rozhraním určených pre operačný systém Windows. Programy vytvorené v C# potrebujú mať nainštalovaný NET framework na počítači, na ktorom majú byť spustený. Výhodou je to, že program vytvorený v programovacím jazykom C# je prenositeľný medzi operačnými systémami, ktoré majú príslušný NET framework nainštalovaný. Podstatnou nevýhodou C# to, že programy v ňom vytvorené sú pomalšie ako napríklad programy vytvorené v C++, čo je v prípade tohto prototypu umocnené tým, že C# používa na kreslenie GDI+.

Vzhľadom na to, že ide o prototyp aplikácie, ktorá bola navrhnutá v časti 7, boli implementované iba nevyhnutné funkcie na overenie výsledkov práce. V prototype nie je implementované ukladanie vytvorených dokumentov do databázy a nie sú implementované ani viacriadkové operácie a vratné operácie.

8.1 Implementácia udalostí a automatické zmeny

Mechanizmus udalostí je implementovaný pomocou udalostí (event) v C#, ktoré fungujú akoby implementovali navrhnuté rozhranie IEventHandler a preto toto rozhranie nebolo potrebné použiť. Rozhranie IChangeHandler preto vyzerá tak, ako je zobrazené na obrázku 29.

```
public interface IChangeHandler
{
    event EventHandler ObjectChanged;

    void BeginChange();
    void EndChange( bool discardEvent );
    Delegate[] GetReferences();
}
```

Obr. 29: Rozhranie IChangeHandler v C#

Udalosť ObjectChanged možno považovať za špeciálnu udalosť, ktorá bola spomenutá v návrhu. Všetky triedy od textu až po projekt implementujú toto rozhranie. Každá normálna udalosť je definovaná jednou C# udalosťou ako to je znázornené na obrázku 30.

```
event EventHandler RowAdded;
event EventHandler RowChanged;
event EventHandler RowRemoved;
```

Obr. 30: Normálne udalosti dokumentu

Udalosti sú základom automatických zmien. V prototype sú implementované automatické zmeny tak, že vždy existuje iba jeden objekt, ktorý napríklad reprezentuje určitého účastníka, krok, tok udalostí alebo niečo iné a všetky súvisiace objekty tento objekt pozorujú. Tým sa dokážu automaticky prispôsobiť zmene, keď nastane. Naopak, keď má byť objekt napríklad odstránený a je potrebné upraviť ostatné objekty, nájdu sa jednoducho pomocou GetReferences.

8.2 Implementácia pohľadu

Dokument by mal byť úplne nezávislý od pohľadu a preto je pohľad implementovaný tak, že ukladá dvojice IDocumentRow, IViewRow do tabuľky s rozptýlenými kľúčmi sledovaním udalosti

dokumentu RowAdded a RowRemoved. Pri pridaní riadka do dokumentu sa vytvorí nový riadok pohľadu a pri odobratí riadka dokumentu sa riadok pohľadu z tabuľky zmaže. Každý riadok pohľadu od vytvorenia sleduje udalosti riadka dokumentu a okamžite sa prispôsobuje ešte predtým, ako dôjde k prekresleniu, aby sa zmeny vizuálne prejavili. Keď príde k vykresľovaniu dokumentu, tak pohľad prechádza riadky dokumentu a hľadá nim riadky pohľadu v tabuľke, ktoré potom vykresľuje.

Riadok pohľadu implementuje dôležitý algoritmus na rozloženie textu, kde sa riešia dva základné problémy:

1. Ako rozmiestniť text rôznej veľkosť na jeden riadok?
2. Ako rozmiestniť text tak aby sa nerozdeľovali slová pokiaľ sa inak nedá?

Prvý problém je vyriešený tak, že stačí získať maximálny ascent (pozri *Obr. 19*) v riadku, prostredníctvom ktorého sa dá vypočítať base line, ktorý bude platiť celý text na jednom riadku. Druhy problém je vyriešený tak, že sa postupne z textu odtrhávajú celé slová a celé medzery od začiatku textu do konca. Medzery sa ukladajú tak, že sa delia ak sa už do riadka nezmestia. Pri slovách ale nastávajú 3 prípady:

1. Slovo sa zmestí do riadka celé => slovo sa umiestni do riadka.
2. Slovo sa už do riadka nezmestí ale zmestí sa do nového riadka => slovo sa umiestni do nového riadka.
3. Slovo sa už do riadka nezmestí a nezmestí sa ani do nového => slovo sa rozdelí tak, aby sa jedna časť zmestila do daného riadka a so zvyškom sa znova testujú uvedené možnosti ale pre nový riadok.

9 Zhodnotenie a ďalšia práca

Hlavným cieľom tejto práce bolo navrhnutie vhodnejšieho spôsobu zaznamenávania a udržiavania textových opisov prípadov použitia, pretože mnoho UML a CASE nástrojov neposkytuje dostatočnú podporu v tejto oblasti, čo sa preukázalo aj pri analýze podpory nástrojov, kde väčšina týchto nástrojov patrila buď do textovo orientovanej alebo do šablónovo orientovanej kategórie. Na základe analýzy jednotlivých častí textových opisov prípadov použitia bol vytvorený metamodel pre modelovanie prípadov použitia. Základom bol koncept modelovania prípadov použitia, ktorý je súčasťou UML. Tento koncept neobsahoval takmer žiadne prvky vyskytujúce sa v textovom opise prípadov použitia a preto bol rozšírený o toky, kroky a ďalšie časti textového opisu.

Nad vytvoreným metamodelom boli definované možnosti profilovania, ktoré sú vo väčšine prípadov založené na obmedzení násobnosti relácií, vynechaní určitých častí metamodelu alebo obmedzení ich používania. Identifikovaných bolo 22 možností, ktorých rôzne kombinácie nastavení produkujú rôzne profily pre navrhnutý metamodel. S cieľom vyhodnotenia profilovania a aj vytvoreného metamodelu ako takého bol implementovaný prototyp nástroja na modelovanie prípadov použitia.

V ďalšej práci by bolo možné doimplementovať navrhnutý nástroj na modelovanie prípadov použitia. Ďalej by bolo vhodné preskúmať rozdiely medzi navrhnutým metamodelom a konceptom UML pre modelovanie prípadov použitia a pokúsiť sa tieto rozdiely prekonať, aby do konceptu bolo možné integrovať navrhnuté doplnkové časti. Zatiaľ boli do úvahy brané iba systémový prípady použitia a teda by bolo dobré pozrieť sa aj na biznis prípady použitia

Zoznam použitej literatúry

[1] Jacobson, I.: *Use Cases - Yesterday, Today and Tomorrow*. The Rational Edge, 2003.
[2] Cockburn, A.: Use cases, ten years later. STQE magazine, 2002.
[3] Jacobson, I. et al.: *Object-Oriented Software Engineering. A Use Case Driven Approach.* Addison-Wesley, 1992.
[4] Kulak, D., Guiney, E.: *Use Cases: Requirements in Context*. Addison-Wesley, ACM Press/Addison-Wesley Publishing Co., 2000.
[5] The Standish Group: *Chaos Report*. 2004. http://www.standishgroup.com/
[6] The Standish Group: *Chaos Report*. 2006. http://www.standishgroup.com/
[7] Bittner, K., Spence, I.: *Use Case Modeling*. Addison-Wesley Longman Publishing Co., Inc., 2002.
[8] Jacobson, I., Pan-Wei, Ng: *Aspect-Oriented Software Development with Use Cases(Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2004.
[9] Cockburn, A.: *Writing Effective Use Cases*. 1. vyd. Addison-Wesley Longman Publishing Co., Inc., 2000.
[10] Heumann, J.: *Introduction to Business Modeling using the Unified Modeling Language (UML)*. Rational Edge, 2001.
[11] Object Management Group: *Unified Modeling Language Specification*. verzia 2.1.2, 2007. http://www.omg.org/docs/formal/07-11-02.pdf
[12] Övergaard, G., Palmkvist, K.: *Use Cases: Patterns and Blueprints*. Addison-Wesley, Boston, MA, 2004.
[13] Pender, T.: *UML Bible*. 1. vyd. John Wiley & Sons, Inc., 2003.
[14] Cockburn, A.: *Structuring Use Cases with Goals*. Journal of Object-Oriented Programming, 1997.
[15] Metz, P., O'Brien, J., Weber, W.: *Specifying Use Case Interaction: Types of Alternative Courses*. Journal of Object-Oriented Programming, 2003

Príloha A: Príspevok na konferenciu ECBS-EERC 09

A UML Based Use Case Modeling Metamodel

L'uboš Zelinka and Valentino Vranić
Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technology
Slovak University of Technology,
Ilkovičova 3, 84216 Bratislava 4, Slovakia
zelinka04@student.fiit.stuba.sk, vranic@fiit.stuba.sk

Abstract

There is a variety of approaches to use case modeling. Under certain circumstances, the use of each one of these approaches may be justified. A consistent application of a particular approach requires the existence of its metamodel. Relevant approaches to use case modeling can be covered by a configurable use case modeling metamodel. Based on the analysis of different approaches to use case modeling, such a metamodel is proposed. Its configuration options are identified and the actual values for Jacobson's and Cockburn's notation presented and discussed.

1. Introduction

Use cases are widely used, but often degraded to be merely UML diagrams with only a few words of description. Being UML standardized gives a false impression of uniformness of this technique. Indeed, speaking of use case diagrams, we may note that although there are other graphical notations, in practice, this is truly a realm of UML. However, UML does not prescribe any notation for the use case description, which represents their true nature. There is a tremendous number of varieties in their textual description.

It is possible to discuss advantages of one approach to use case modeling over the other ones, but that would not bring any real value to modelers who have to use them as decisions that lead to adopting a particular approach might be beyond their control. They are interested in having their use case modeling as such—as well as consistency of the resulting models—supported by a tool. However, the tool support of a particular notation cannot possibly exist without making it clear what is, and what is not a part of the notation.

A model of the notation—i.e., a metamodel—is what is actually needed. The metamodel could cover several sufficiently related notations and be configured in to described each one of them. It appears that use case modeling notations are close enough to each other to allow for constructing a use case modeling metamodel, which is the topic of this paper.

Section 2 analyzes differences in understanding and application of the use case notation elements focusing mainly on Jacobson's and Cockburn's notation. Based on this analysis, Sect. 3 introduces a use case modeling metamodel. Section 4 presents options for the configuration of this metamodel and their values for Jacobson's and Cockburn's notation. Section 5 discusses related work. Section 6 brings conclusions and directions of further work.

2. Diversity in use case modeling

A use case describes a coherent functionality that provides some result of value to a user. As the term says, it is a case of a system use [1]. There are many different ways of describing use cases, but all of them have their roots in Jacobson's or Cockburn's notation. This section briefly explains the most prominent elements of use case description as such, and than points out the differences between Jacobson's and Cockburn's notation. The way the technique is used in practice is to a large extent influenced by the capabilities of available tools, so an overview of tool support is presented, too.

2.1. Use case description

A semiformal use case description is simply a natural language text structured using a text template which divides the text into logical parts. Even though there is no broadly accepted standard use case template, the existing templates are quite similar.

Name and *brief description* provide the reader with basic information about the use case. The use case name—sometimes called *title*—uniquely identifies the use case in

the use case model (or at least in its namespace if the model is partitioned). Use case names may be accompanied by identification numbers used to refer to them.

Actors are roles adopted by external entities that interact with the system directly [1]. Typically, actors are user roles, but systems, subsystems, or even time can all perform as actors. Each actor can participate in many use cases and each use case can embrace several actors. It is often distinguished between primary and secondary actors. Primary actors participate in a use case to satisfy their goals, while secondary actors help the system satisfy goals of primary actors.

Preconditions are a set of constraints that should be fulfilled before the use case starts. *Postconditions* are a set of constraints that would be fulfilled after the use case finishes if preconditions have been satisfied before it started. This is actually the design by contract [10], but we may encounter different, less restrictive understanding of preconditions and postcondition, putting them to a merely informative position [15], or a fully restricted view, where the very use case activation is presumed by fulfilling its precondition [1].

Flows of events—or simply just *flows* (known also as *scenarios*)— represent every possible outcome of an attempt to accomplish a use case goal [12]. A flow is a sequence of interactions between an actor and a system. The interactions start from the triggering action and continue until the goal is delivered or abandoned [8]. In the use case description the interactions are represented by steps. Flows are sometimes represented as prose, but usually they are represented as sequences of steps, or—more precisely—partial orderings of steps [4], as some steps simply don't fit into any ordering (such as "at any point, a user can cancel the activity"). Very often, it is distinguished between *main* (or *basic*) and *alternative* flows. A main flow describes the normal sequence of steps in the execution of a use case [2]. Usually, it represents the interaction between the actor and the system under ideal conditions without alternatives and exceptions. Alternative flows cover behavior that is of optional, exceptional, or truly alternate character in relation to another flow [2]. They are dependent on some condition occurring at an explicit point in another flow. Additionally, another category of flows can be distinguished: subflows, which are used to separate repetitive interaction from other flows [8].

Use case relationships are a part of the use case description even though they are not explicitly present in most of the use case templates. UML offers two standard relationships between use cases called *include* and *extend*. The include relationship defines that a use case contains the behavior defined in another use case [11]. The purpose of this relationship is to reuse existing behavior or extract identical behavior. The behavior of the included use case is simply inserted into the behavior described in the including use case. It is similar to a function call in a programming language, but the include relationship should not be used for a functional decomposition. The extend relationship is a relationship directed from the extending use case towards the use case being extended that specifies how and when the behavior defined in the extending use case can be inserted into the behavior defined in the use case being extended [11]. It is typically used to add optional or exceptional behavior without making changes to the behavior described in extended use case, which is similar to alternative flows.

The extend relationship is used in combination with *extension points*, which are named places in the flow of events where additional behavior can be inserted or attached [2]. Every flow of events can have multiple extension points. A common way to define an extension point in a use case description is inserting its name into the flow of events between two steps. It's a pretty straightforward way, but every extension point refers only to a single place in the flow of events, which can be limiting. Since steps in flows of events are usually numbered, it is possible to define the extension point by a step number. In addition it is possible to create extension points that occur over multiple steps between two step numbers.

2.2. Jacobson's notation

Jacobson's and Cockburn's use case modeling notation are well established and distinguished. Consider the example in Fig. 1 and 2. We can see that Jacobson's notation allows multiple main flows. The extension point is defined by a step number in a specific flow which suggest the possibility of using extension points over multiple steps. Beside alternative flows and subflows, Jacobson employs a special flow denoted as *extension flow*, but it can be seen just as an alternative flow defined in another use case, as confirmed in Jacobson's own writing [8].

2.3. Cockburn's notation

Cockburn is a strong proponent of a purely textual representation of use cases. In the example of extend relationship in Fig. 3, the *Check spelling* use case extends the *Edit a document* use case implicitly by its main flow. There is no explicit extension point either: the extension point is referred to descriptively in the trigger part of the description.

2.4. Tool support

While general UML modeling tools offer some support of use case modeling, there are also dedicated use case modeling tools with usually better coverage of use case description. The challenging areas of use case modeling support are flows and use case relationships because their changes

Use Case: Reserve Room
Basic Flows:
B1. Reserve Room
The use case begins when a customer wants to reserve a room.

1. The customer selects to reserve a room.
2. The system displays the types of rooms the hotel has and their rates.
3. The customer **Check Room Cost**.
4. The customer makes the reservation for the chosen room.
5. The system deducts from the database the number of rooms of the specified type available for reservation.
6. The system creates a new reservation with the given details.
7. The system displays the reservation confirmation number and check-in instructions.
8. The use case terminates.

Alternate Flows:
A1. Duplicate Submission If in step 5 of the basic flow there is an identical reservation in the system (same name, e-mail, and start and end dates), the system displays the existing reservation and asks the customer if he wants to proceed with the new reservation.

1. If the customer wants to continue, the system proceeds with the reservation, and the use case resumes.
2. If the customer indicates that the new reservation is a duplicate, the use case terminates.

Subflows:
S1. Check Room Cost

1. The customer selects his desired room type and indicates his period of stay.
2. The system computes the cost for the specified period.

Extension Points:
E1. Update Room Availability The Update Room Availability extension point occurs at step 5 of the Basic Flow.

Figure 1. A use case in Jacobson's notation (adopted from [2]).

Use Case: Handle Waiting List
Extension Flows:
EF1. Queue for Room This extension flow occurs at the extension point Update Room Availability in the Reserve Room use case when there are no Rooms of the selected type available.

1. The system creates a pending reservation with a unique identifier for the selected Room type.
2. The system puts the pending reservation into a waiting list.
3. The system displays the unique identifier of the pending reservation to the customer.
4. The base use case terminates.

Figure 2. An extension use case in Jacobson's notation (adopted from [2]).

Use Case: Edit a document
Primary actor: user
Scope: Wapp
Level: user goal
Trigger: User opens the application.
Precondition: none

Main success scenario:

1. User opens a document to edit.
2. User enters and modifies text. .....User saves document and exits application.

Use Case: Check spelling Primary actor: user
Scope: Wapp
Level: subfunction!
Precondition A document is open
Trigger: Anytime in **Edit a document** that the document is open and the user selects to run the spell checker.
Main success scenario: ...etc....

Figure 3. Use cases in Cockburns's notation (adopted from [4]).

affect the integrity of textual use case descriptions. Based on the creation of textual use case descriptions, it is possible to distinguish three categories of use case tools: text based, template based, and model based tools.

In the text based use case modeling tools, the use case description—including flows and use case relationships—is written as plain or formatted text into respective text boxes. Some tools in this category support formatted text, which makes the textual use case descriptions easier to read. The general problem of these tools is the lack of the support for the use case description maintenance. Examples of such tools include ArgoUML, Poseidon for UML, and IBM Rational Software Architect

In the template based use case modeling tools, a static or dynamic template is used to create use case descriptions. Templates basically partition the description text (plain or formatted). Common partitionings distinguish between flows and a range of simple description items, such as use case name, brief description, preconditions, postconditions, and other similar parts of the use case description that do not use numbering. In case of dynamic template support, the templates can be adapted by adding new or removing existing types of flows and description items to fit the user needs. While the maintainability of use case descriptions is still problematic even in such tools, they are at least easier to read and write. For example, Visual Paradigm and Enterprise Architect fall into this category.

In model based use case modeling tools, use case descriptions are based on a specific use case model. These tools usually contain sophisticated formatted text editors that directly manipulate the structured use case descriptions according to the use case model. For example, every step is a part of a specific flow, which allows the user to perform changes in the order of steps that result in automatic renumbering of affected steps and other parts of the use case description that depend on them like alternative flows or extension points. This is only one of many ways how tools of

this category help the user to ensure the integrity of use case descriptions. Visual Use Case and CaseComplete are examples of model based use case modeling tools.

3. Invoking a metamodel

The diversity in use case modeling can be concisely captured in a metamodel, which can serve to better understand this technique and as a basis for the development of configurable use case modeling tools. Since use case modeling is partially covered by the UML metamodel, we will adapt and extend it with the notions needed to cover textual part of use cases. Our aim was not to add to UML specification, but to develop a concise, standalone use case modeling metamodel. Integration into the UML metamodel is a part of our ongoing work.

3.1. Flows

Flows of events (`Flow`) are an essential part of the use case description (see Figure 4). Each flow consists of steps (`Step`). This is a generally accepted idea, yet actual step representation, including their ordering, may vary significantly and as such is beyond this metamodel. However, in our metamodel, we do recognize the possibility of existence of special kinds of steps (`TypedStep`) with the agreed meaning given by their type (`StepType`) which could be defined by its name, the list of parameters and the list of parameter names.

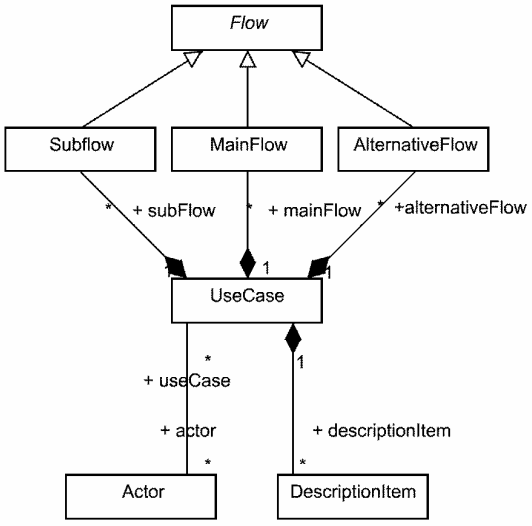


Figure 4. Flows.

Three types of flows can be recognized: main (basic) flow (`MainFlow`), subflow (`Subflow`), and alternative flow (`AlternativeFlow`). In general, a use case may have any number of main flows—even none—though some approaches require a main flow.

A use case without a main flow would represent a use case that could not be activated directly by an actor. Instead, it would be intended just for inclusion in other use cases or to extend them, in which case it should provide one or more subflows or alternative flows, respectively.

A use case may include preconditions and postconditions, which are a kind of a constraint (`Constraint`). Other parts of use case description vary significantly among approaches and even in a particular approach depending software analyst preferences, so they are just indicated as any kind of a description item (`DescriptionItem`).

In our metamodel, we made the participation relationship between `Actor` and `UseCase` explicit whereas in the UML metamodel it is given by the fact that these two metaclasses are derived from `BehavioredClassifier` which allows for them to be associated.

An alternative flow is activated, usually according to a constraint (`Constraint`), in a particular step (`Step`). In some approaches, it is possible to specify the execution order of the alternative flow with respect to the step affected by it usually before, after, or around it, i.e. with the full control upon the step (just like advices in aspect-oriented programming).

Any flow can have subflows. A use case with no flows can be an abstract use case intended to be specialized [1].

3.2. Relationships

In general, there are two types of use case relationships: include and extend. The UML metamodel recognizes both of them as a special kind of `DirectedRelationship` (which is the metaclass the general dependency is derived from, too, making them a close relative of it). However, some approaches ignore the extend relationship and, hypothetically, there could be approaches that wouldn't provide not even the include relationship.

The include relationship (see Figure 5) means an inclusion of a specific flow from another use case (`FlowInclusion`) in one or several steps of the including use case (`Step`). We opt for inclusion of a general flow (`Flow`), although it is unlikely that someone would want to include an alternative flow. The inclusion of a flow may be constrained (`Constraint`).

The inclusion of a flow (`FlowInclusion`) is possible even without the corresponding include relationship (zero multiplicity of `Include`), which covers flow activations of use case's own flows.

The extend relationship means an extension of one or several extension points (`ExtensionPoint`) of the use case being extended by a specific extension flow (`FlowExtension`). Some approaches allow only an al-

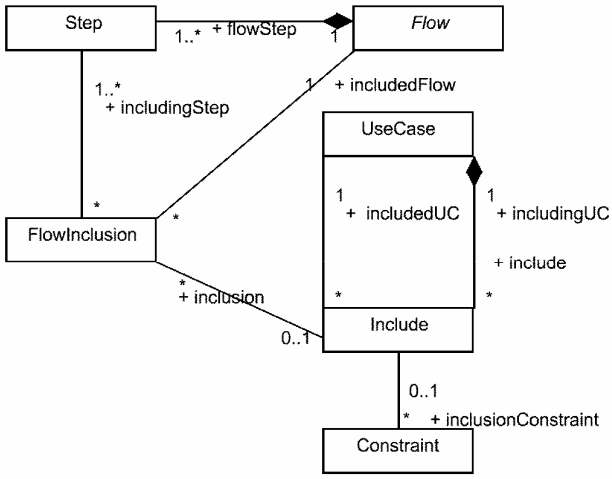


Figure 5. The include relationship.

ternative flow to serve as an extending flow, but this is not generally accepted, so our metamodel allows any kind of flow in this role.

Analogously to alternative flows—which actually act as extension flows in a single use case—some approaches allow to specify the execution order of the extension flow with respect to the extension point, usually before, after, or around it, i.e. with the full control upon the extension point (just like advices in aspect-oriented programming).

An extension point (`ExtensionPoint`) is merely a name of the step or a range of steps (`startingStep`–`endingStep`) represented by an extension location (`ExtensionLocation` exposed by the use case being extended. The extension of a flow may be constrained (`Constraint`). In the UML metamodel, there is at most one constraint for each extend relationship. Our metamodel allows several constraints for each extension flow.

As with the flow inclusion, the extension of a flow (`FlowExtension`) is possible even without the corresponding extend relationship (zero multiplicity of `Extend` and `ExtensionPoint` with respect to `ExtensionLocation`), which covers flow alterations of use case's own flows.

4. Metamodel configuration

The use case modeling metamodel proposed in the previous section can be configured to represent an established notation or simply to define a use case modeling that fits the needs of a particular organization. This can be done mostly by restricting multiplicities of associations in the metamodel. Such a restriction can represent any subset of values allowed by the original multiplicity, but only a total restriction to zero is of practical meaning.

Restricting multiplicities might be seen as a low-level metamodel configuration. To make the configuration easier, we can represent most of the configuration options as Boolean variables where true stands for the original multiplicity, and false for the zero multiplicity. Table 1 presents the list of the most important Boolean configuration options and their values in Jacobson's (J) and Cockburn's (C) notation, which we discussed in Sect. 2.2 and 2.3.

Table 1. The use case metamodel configuration options and their values in Jacobson's (J) and Cockburn's (C) notation.

Property	J	C
Single-Step Extension Points	Y	N
Range Extension Points	Y	N
Mandatory Main Flow	Y	Y
Multiple Main Flows	Y	N
Subflows	Y	Y
Subflows in Main Flows	Y	Y
Subflows in Alternative Flows	Y	Y
Subflows in Subflows	Y	Y
Alternative Flows	Y	Y
Alternative Flows in Main Flows	Y	Y
Alternative Flows in Subflows	N	Y
Alternative Flows in Alternative Flows	N	Y
Extension	Y	N
Multiple Extension Locations in an Extension	N	N
Extension by a Specific Flow	Y	N
Extension Flow Constraint	Y	N
Extension Flow Execution Order	Y	N
Inclusion	Y	Y
Inclusion Flow Constraint	N	N
Inclusion of a Specific Flow	N	N

Single-Step Extension Points means the possibility of having `startingStep` to be equal to `endingStep`. For *Range Extension Points*, `startingStep` must be able to differ from `endingStep`.

Mandatory Main Flow restricts the minimum multiplicity of `mainFlow` to 1, while no *Multiple Main Flows* would mean restricting its maximum to 1.

Options *Subflows*, *Alternative Flows*, *Inclusion*, and *Extension* have a meaning of the very presence of respective metaclasses.

If subflows are allowed, their inclusion can be restricted with respect to the type of the including flow by the following options: *Subflows in Main Flows*, *Subflows in Alternative Flows* and *Subflows in Subflows*. There are analogous options for alternative flows: *Alternative Flows in Main Flows*, *Alternative Flows in Subflows*, and *Al-*

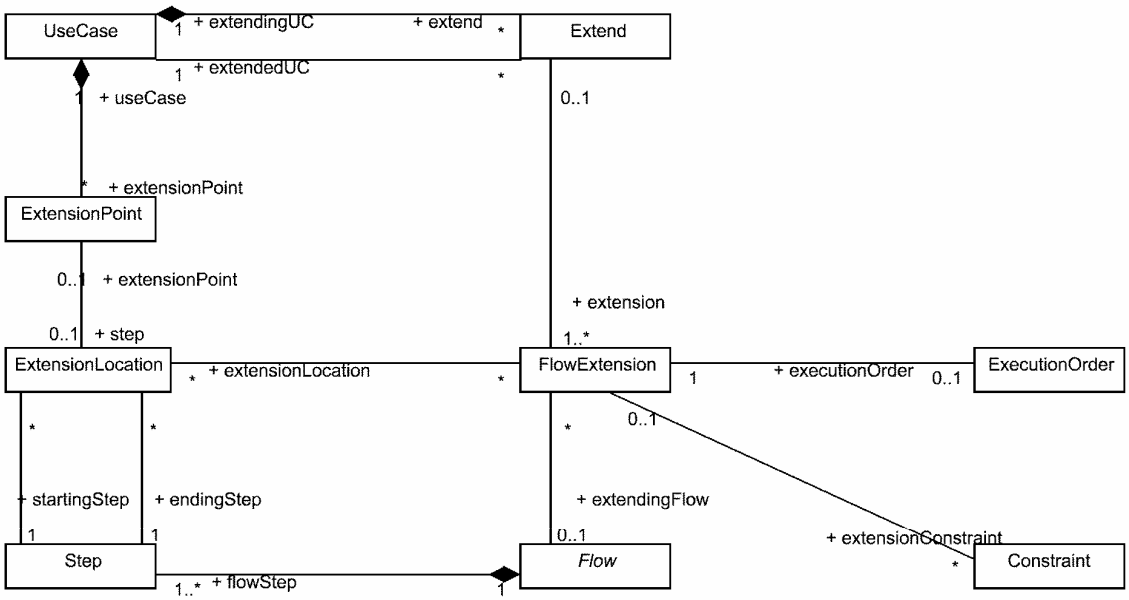


Figure 6. The extend relationship.

ternative Flows in Alternative Flows. All these options would be realized by constraining associations `Flow–InclusionFlow` and `Flow–ExtensionFlow` to allow only desired subtypes of `Flow`.

The rest of Boolean options has the realization in multiplicity restriction as listed below:

- no *Multiple Extension Locations in an Extension*: maximum `extensionLocation` multiplicity is 1
- no *Extension by a Specific Flow*: maximum `extendingFlow` is 0
- no *Extension Flow Constraint*: maximum `extensionConstraint` is 0
- no *Extension Flow Execution Order*: maximum `executionOrder` is 0
- no *Inclusion Flow Constraint*: maximum `inclusionConstraint` is 0
- no *Inclusion of a Specific Flow*: maximum `includedFlow` is 0

There are two configuration options that consist of a list of elements (not listed in the table). *Description Items* represents a (possibly) empty list of textual description items a use case can have. *Execution Order Types* is a list of values that represent possible execution order types of extension flows, usually before, after, or around (as has been mentioned in Sect. 3.1).

5. Related Work

The UML metamodel [11] is a prominent attempt of establishing a common diagrammatical use case modeling notation, but it also defines a set of notions, which we find useful as a basis for our use case modeling metamodel.

Rui and Butler [13] proposed a use case modeling metamodel focused on a single use case modeling notation. Others have focused on unifying specific notational issues in use case modeling such as alternative flow types [9], formalizing the include and extend relationships [3], or even formalizing use cases as such [14].

While the use case modeling metamodel proposed in this paper also attempts to cover diversity of options in use case modeling, its goal is not to unify them—at least not more than necessary—but to map the common and variable among them and to provide a way to opt for a particular notation or a combination of several notations as needed. The latter requires a consistency check, which the proposed metamodel is capable of, too.

6. Conclusions and Further Work

In this paper we proposed a use case modeling metamodel. It is based on UML metamodel and embraces mainly Jacobson's and Cockburn's use case notation. The proposed metamodel is configurable by including or omitting some of its elements, posing some constraints on their

use, or by constraining multiplicity of relationships among them.

We expect our metamodel would develop further to embrace other possibilities of use case modeling. For example, we have not considered explicitly business use cases [7], which certainly deserve attention. One way of exploring broadly the use case modeling domain is the actual realization of the metamodel in a configurable use case modeling tool that would also help test the metamodel and choice of configuration options in practice, which we are currently working on.

With respect to configuration, we would like to explore the possibility of employing feature modeling [6, 16]. For getting a full use of a feature model, the metamodel would have to be transformed to include all the possibilities of its configuration space and have them mapped to respective features according to the approach of the superimposed variants [5].

Yet another line of further work is exploring the position of the elements of our use case modeling metamodel within the UML metamodel that would bring us to its proper integration with the UML metamodel.

References

[1] J. Arlow and I. Neustadt. *UML 2 and the Unified Process*. Addison-Wesley, 2005.

[2] K. Bittner and I. Spence. *Use Case Modeling*. Addison-Wesley, 2002.

[3] A. Bragança and R. J. Machado. Extending uml 2.0 metamodel for complementary usages of the «extend» relationship within use case variability specification. In *Proc. of 10th International Software Product Line Conference, SPLC 2006*, pages 123–130, Baltimore, USA, 2006. IEEE Computer Society Press.

[4] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2000.

[5] K. Czarnecki and M. Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In R. Glück and M. R. Lowry, editors, *Proc. of Generative Programming and Component Engineering, 4th International Conference, GPCE 2005*, LNCS 3676, pages 422–437, Tallinn, Estonia, Oct. 2005. Springer.

[6] K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[7] J. Heumann. Introduction to business modeling using the unified modeling language (uml). developerWorks, IBM, Nov. 2003. `http://www.ibm.com/developerworks/rational/library/360.html`.

[8] I. Jacobson and N. Pan-Wei. *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley, 2004.

[9] P. Metz, J. O'Brien, and W. Weber. Specifying use case interaction: Types of alternative courses. *Journal of Object-Oriented Programming*, 2(2):111–131, Mar. 2003.

[10] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall, second edition, 1997.

[11] Object Management Group. OMG unified modeling language (omg uml), superstructure, v2.1.2. Technical report, Nov. 2007. `http://www.omg.org/docs/formal/07-11-02.pdf`.

[12] T. Pender. *UML Bible*. Wiley, 2003.

[13] K. Rui and G. Butler. Refactoring use case models: The metamodel. In M. J. Oudshoorn, editor, *Proc. of 26th Australasian Computer Science Conference, ACSC 2003*, pages 301–308, Adelaide, Australia, Feb. 2003.

[14] P. Stevens. On use cases and their relationships in the unified modelling language. In H. Hußmann, editor, *4th International Conference on Fundamental Approaches to Software Engineering, FASE 2001, held as a part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001*, LNCS 2029, pages 140–155, Genova, Italy, Apr. 2001. Springer.

[15] G. Övergaard and K. Palmkvist. *Use Cases: Patterns and Blueprints*. Addison-Wesley, 2004.

[16] V. Vranić. Reconciling feature modeling: A feature modeling metamodel. In M. Weske and P. Liggsmeyer, editors, *Proc. of 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World (Net.ObjectDays 2004)*, LNCS 3263, pages 122–137, Erfurt, Germany, Sept. 2004. Springer.

Príloha B: Príspevok na študentskú konferenciu IIT.SRC 2009

Unifying Different Approaches to Use Case Modeling by a Configurable Metamodel

Ľuboš ZELINKA*

Slovak University of Technology
Faculty of Informatics and Information Technologies
Ilkovičova 3, 842 16 Bratislava, Slovakia
zelinka04@student.fiit.stuba.sk

1 Extended abstract

Requirements engineering in software development includes elicitation, analysis and documentation of requirements. It is one of the most important tasks in software engineering, because it has a significant impact on the success or failure of a software project. Use cases are a useful technique for documenting functional requirements. The whole intended behaviour of a system is described by a use case model, which contains use cases, stakeholders and relations. A use case model is typically represented as textual description or UML diagram just like the use cases themselves.

Our work focuses on the semiformal textual description of use cases, because the semiformal structure of the text makes it relatively easy for the stakeholders to read or write use case descriptions. A semiformal use case description is simply a natural language text structured using a text template, which divides the text into logical parts. Even though there is no broadly accepted standard use case template, our analysis of various use case templates shows that they are quite similar. We managed to identify several common parts of semiformal use case descriptions: name, brief description, actors, preconditions, postconditions, flows of events, extension points and use case relationships.

Based on the analysis of semiformal use case descriptions we created a configurable metamodel for use case modeling. The idea was to generalize different semiformal use case descriptions and provide a way to represent various writing guidelines. The metamodel is based on the concept for use case modeling presented in UML, but since this concept does not provide any details about the use case description we changed and extended it accordingly. It supports the creation of use case fragments, extension points over multiple steps, multiple flow inclusions in an include

---

* Master degree study programme in field: Software Engineering
Supervisor: Dr. Valentino Vranić, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

IIT.SRC 2009, Bratislava, April 29, 2009, pp. 1–2.

relationship, multiple flow extensions in an extend relationships (see Figure 1) and configurations.

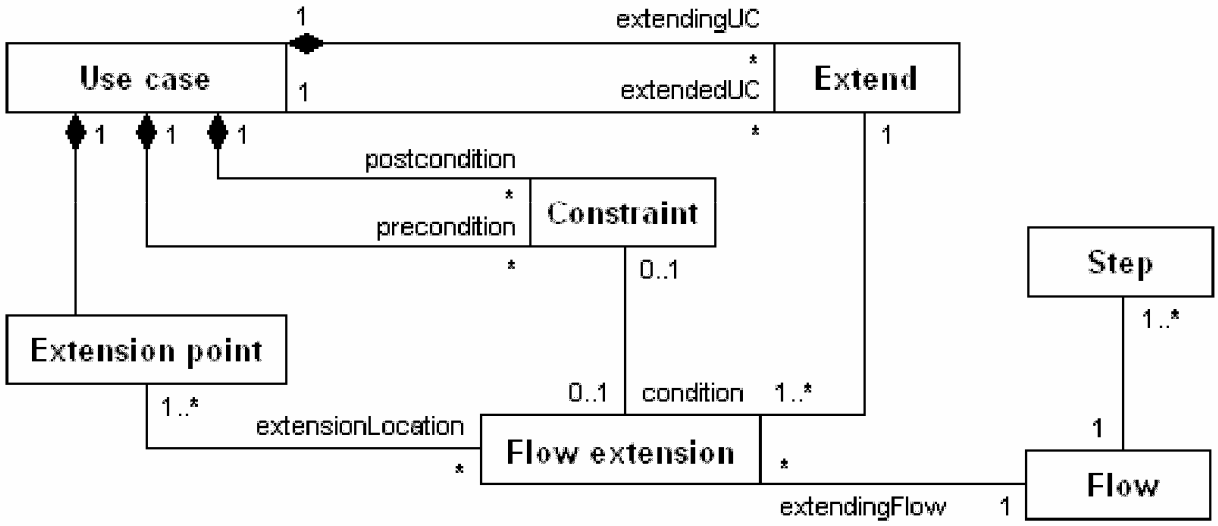


Figure 1. Representation of extend relationship.

Configurations were created to represent different use case writing guidelines, which are important in practical use case modeling. A configuration is a set of option and their values. Each configuration option represents a specific aspect of use case modeling, for example:

- use of multiple step extension points
- use of subflows in other subflows
- use of extend relationships
- use of use case fragments

The combinations of their values characterize different unique configurations. Every configuration is like a writing guideline, but it is based on the created metamodel rather than a practical experience. In our work we were able to identify over 20 configuration options in our metamodel and successfully created configurations for several use case writing guidelines.

Currently we are in the process of implementing a prototype tool for use case modeling based on the created configurable metamodel. The tool is designed as a special text editor for use case descriptions, where the use case project contains a configuration that represents the use case writing guideline for use case models in it.

Acknowledgement: This work was partially supported by the Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava.

Príloha C: Obsah elektronického nosiča

Adresár/Súbor	Popis
Dokumenty\dp.doc	Diplomová práca
Dokumenty\anotacia.doc	Anotácia k diplomovej práci
Framework\dotnetfx.exe	Inštalačný súbor NET Framework 2.0 Redistributable Package
Implementacia	Adresár so zdrojovými kódmi prototypu
obsah.txt	Obsah elektronického nosiča