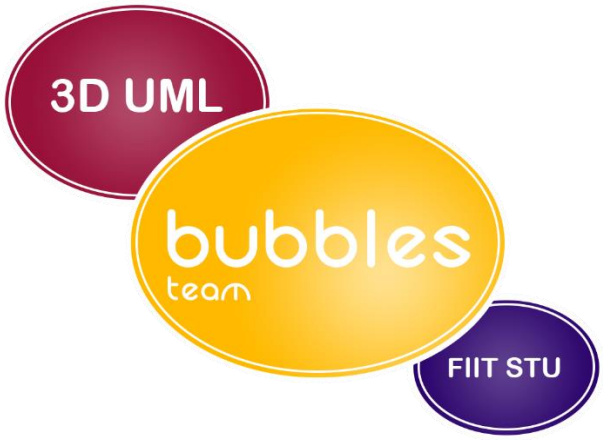
Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Ilkovičova 2, 842 16 Bratislava 4



3D UML

Dokumentácia k inžinierskemu dielu

Tím č. 4

Tím: *Bc. Hana Baranovičová, Bc. Francisc Juraş, Bc. Miroslav Kudláč, Bc. Lukáš Markovič, Bc. et Bc, Martin Melis, Bc. Michal Valovič, Bc. Andrej Železňák*
Vedúci tímu: *Ing. Ivan Polášek, PhD.*
Študijný program: *Informačné systémy/Softvérové inžinierstvo*
Predmet: *Tímový projekt I*
Akademický rok: *2014/2015, zimný semester*

Obsah

Zoznam obrázkov

1. Úvod

Jazyk UML je najrozšírenejším a najuznávanejším jazykom pre modelovanie, či už pri analýze, návrhu alebo samotnom vývoji informačných systémov. Jazyk sa postupne stal neoddeliteľnou súčasťou života IT architektov, návrhárov a každého človeka, ktorý pracuje na rozsiahlejšom projekte.

Zobrazenie UML diagramov v 3D priestore sa v súčasnosti bežne nepoužíva, ale v budúcnosti má určite obrovsky potenciál. Zložitosť softvérových systémov každým rokom narastá, s čím je samozrejme spojený aj nárast zložitosti UML diagramov. Pridanie tretieho rozmeru ku štandardným 2D UML diagramov prináša nové možnosti zobrazenia, napríklad na poli viacvrstvových systémov, kde je možné vrstvy systému modelovať priamo prepojenými vrstvami v jazyku 3D UML. Ďalším prínosom sú prehľadnejšie diagramy, či možnosť vizualizácie modelu systému v 3D priestore.

V 3D UML má aj vďaka týmto vlastnostiam široké uplatnenie vo vedeckej a akademickej oblasti, no zároveň má potenciálne komerčné uplatnenie aj v oblasti softvérového inžinierstva, kde by si nástroj podporujúci jazyk 3D UML našiel určite svojich priaznivcov.

Úlohou tohto projektu je vylepšiť doterajší prototyp nástroja umožňujúceho modelovanie prostredníctvom jazyka UML v 3D priestore. Nástroj je v súčasnosti vo fáze vývoja, kde aktuálne podporuje diagram tried a sekvenčný diagram.

Tento dokument slúži ako dokumentácia inžinierskeho diela, ktorého cieľom je implementovať do existujúceho prototypu taktiež diagram aktivít, ktorý by zároveň zahŕňal funkcionalitu fragmentov, ktoré poznáme zo sekvenčných diagramov. V dokumente sa nachádza analýza existujúceho riešenia, analýza UML aktivity diagramov a taktiež rozbor UML fragmentov. Ďalšia časť dokumentu obsahuje samotný návrh architektúry – metamodelu 3D UML diagramu aktivít.

2. Ciele projektu v zimnom semestri

Na projekte 3D UML pracovalo v minulosti už viacero ľudí, ktorí vytvorili dobrý základ a overili technológie na vytváranie ďalších častí tohto modelovacieho nástroja.

Jedným z hlavných cieľov projektu v zimnom semestri je analýza a návrh korektného metamodelu diagramu aktivít pre 3D UML, ktorý by zároveň spĺňal definíciu jazyka UML, no bol by rozšírený o fragmenty, ktoré sa nachádzajú napríklad v sekvenčnom diagrame.

Druhým cieľom zimného semestra je implementácia navrhnutého metamodelu diagramu aktív do existujúceho prototypu. S touto úlohou sa spája aj kompletný refaktoring doteraz implementovaného riešenia, ktoré funguje bez akéhokoľvek metamodelu a implementácia fragmentov do diagramu aktivít.

Nemenej dôležitou úlohou je návrh a implementácia grafického rozhrania prototypu, ktoré bude umožňovať jednoduchú a pohodlnú manipuláciu s diagramom a jeho jednotlivými komponentami, t.j. pridávanie, spájania a úprava komponentov.

Vývoj tohoto prototypu prebieha v programovacom jazyku C++, pričom na grafickú vizualizáciu je využitá knižnica OGRE. Tieto technológie budú využité aj v našom projekte a konzistencia použitých technológií zostane zachovaná.

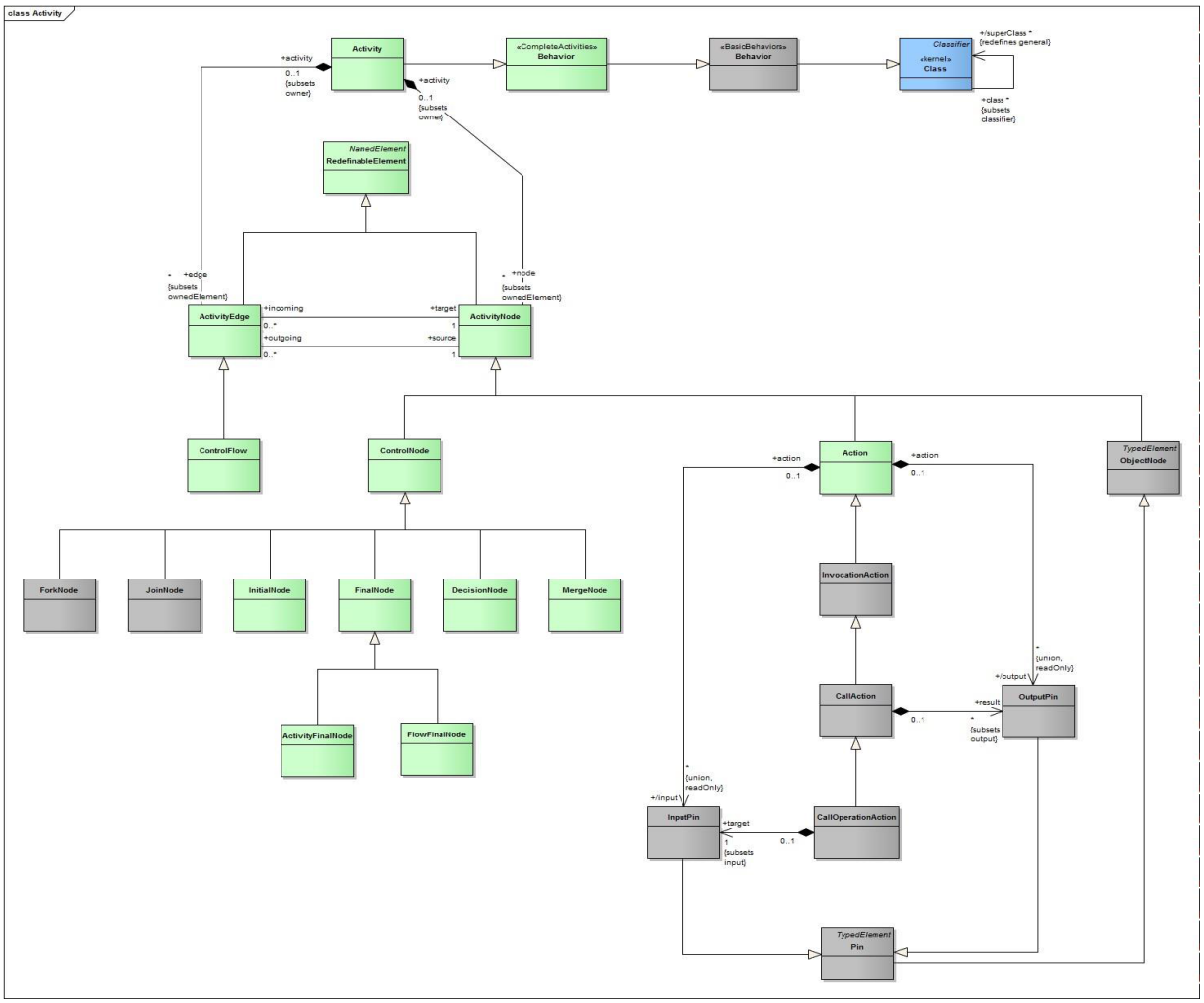
Nižšie sú prehľadne zhrnuté hlavné ciele projektu pre zimný semester:

- *Návrh metamodelu diagramu aktivít 3D UML obsahujúceho fragmenty*
- *Implementácia metamodelu do existujúceho prototypu*
- *Návrh a implementácia grafického používateľského rozhrania pre diagram aktivít*

3. Aktuálny pohľad na systém

Keďže projekt 3D UML obsahuje v súčasnom stave aj diagram aktív, bolo preto nutné urobiť jeho dôkladnú analýzu, ktorá je pre nás kľúčová.

Diagram aktivít, ktorý je aktuálne vytvorený v prototype reprezentuje model aktivít, ilustrovaný na obrázku č. Obr. 1 Diagram modelu aktív. Prvky, ktoré neboli implementované sú znázornené šedou farbou.

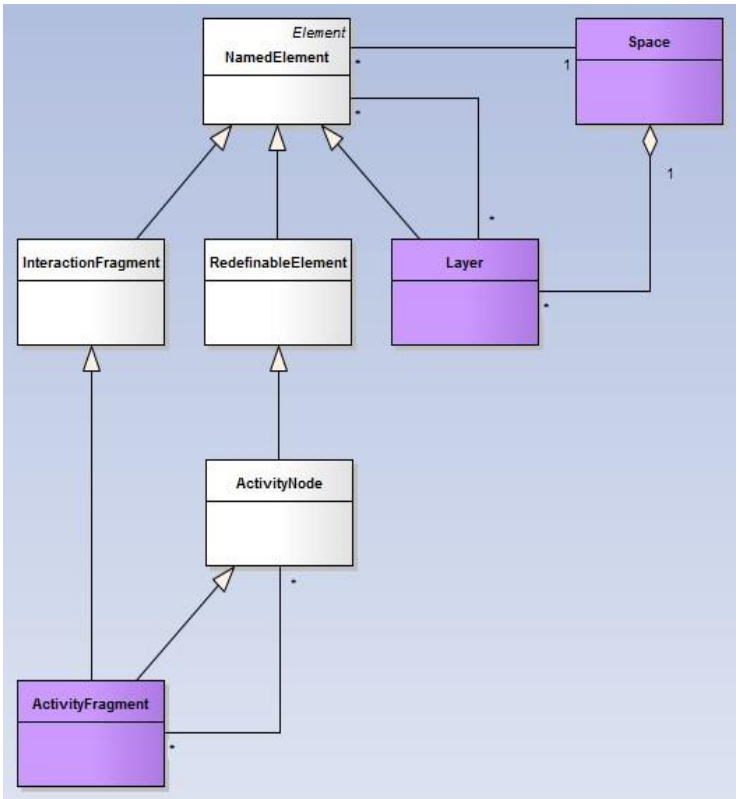


Obr. 1 Diagram modelu aktív

Popis jednotlivých tried diagramu:

Názov triedy	Popis triedy
Behavior <CompleteActivities>	Správanie je špecifikácia toho, ako sa klasifikátor kontextu mení v čase. Reprezentuje možné vykonateľné správanie, alebo ilustráciu zaujímavej podmnožiny možných správaní.
Activity	Aktivita je špecifikácia parametrizovaného správania, ktoré je sekvenciou podriadených prvkov. Týmito podriadenými prvkami sú akcie.
Action	Akcia reprezentuje jeden krok v rámci aktivity. Je to činnosť, ktorá je vykonávaná aktivitami.
ActivityEdge	Hrana aktivity reprezentuje abstraktnú triedu pre prepojenia medzi dvomi uzlami aktivít.
ActivityNode	Uzol aktivity je abstraktná trieda pre body, nachádzajúce sa v toku aktivity, ktoré sú prepojené hranami.
ControlFlow	Riadiaci tok je hrana, ktorá spúšťa uzol aktivity po tom, ako bol predchádzajúci uzol dokončený.
ControlNode	Riadiaci uzol je abstraktný uzol aktivity, ktorý usmerňuje toky v rámci aktivity.
InitialNode	Inicializačný uzol je riadiaci uzol, v ktorom začína tok, keď je vyvolaná aktivita.
FinalNode	Koncový uzol je abstraktný riadiaci uzol, v ktorom sa tok v aktivite zastaví.
DecisionNode	Rozhodovací uzol je riadiaci uzol, ktorý vyberá, ktorým z vychádzajúcich tokov sa bude aktivita ďalej uberať.
MergeNode	Zlučovací uzol je riadiaci uzol, ktorý spája dohromady niekoľko alternatívnych tokov. Tento uzol nie je používaný k synchronizácií paralelných tokov, ale na spájanie alternatívnych tokov.
ActivityFinalNode	Koncový uzol aktivity slúži a zastavenie všetkých tokov v aktivite.
FlowFinalNode	Koncový uzol toku slúži na ukončenie toku.

Vzhľadom na potrebu zobrazovania prvkov diagramu v trojrozmernom priestore a ich zaraďovania do vrstiev, musel byť UML metamodel obohatený o ďalšie triedy. Tieto triedy je možné vidieť na obrázku nižšie (viď obr.29). Zvýraznené sú fialovou farbou.



Obr. 2 Elementy, o ktoré bol metamodel UML doplnený

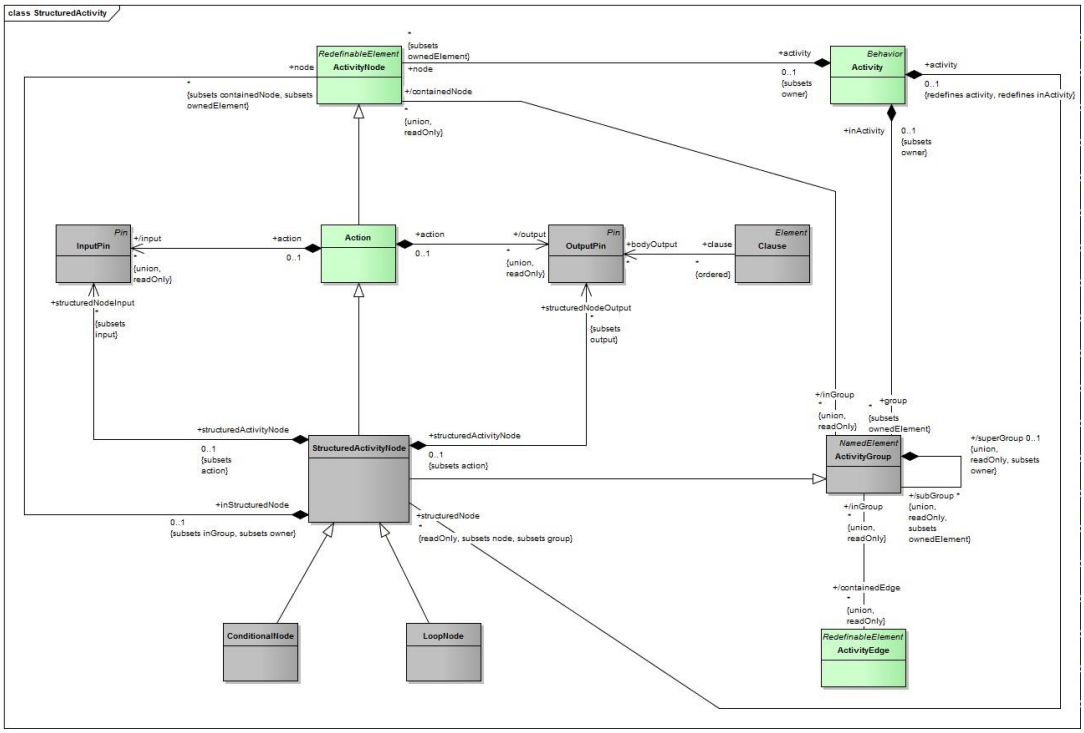
Prvou zložkou, ktorá bola do metamodelu pridaná je samotná vrstva (trieda *Layer*). Tieto sa v prototype využívajú na zobrazovanie zhluku prvkov, ktoré spolu súvisia. Ďalším pridaným prvkom je priestor (trieda *Space*). Tento reprezentuje trojrozmerný priestor, v ktorom sú všetky jednotlivé elementy umiestňované.

Okrem týchto zložiek bola do modelu pridaná trieda *ActivityFragment*. Tento fragment slúži na reprezentáciu fragmentov v diagrame aktivít. Bežné UML diagramy aktivít totiž neumožňujú vytváranie fragmentov.

Dedenie nového fragmentu od triedy *InteractionFragment* je celkom intuitívne, keďže sa jedná o typ fragmentu. Na prvý pohľad nemusí byť zrejmý dôvod, prečo táto trieda dedí tiež od ďalšej triedy – *ActivityNode*. Toto dedenie bolo pridané z praktického dôvodu – umožňuje totiž veľmi jednoduchú integráciu do diagramu aktivít. Vďaka tomuto dedeniu môže byť fragment tiež cieľom alebo zdrojom riadiaceho toku. Tento fragment môže obsahovať rôzne iné uzly aktivít.

3.1. Model štruktúrovaných aktivít

Ďalším diagramom je diagram znázorňujúci model štruktúrovaných aktivít. Tento diagram nie je v prototype implementovaný, môžeme však použiť časti, ktoré sú implementované v modely aktivít a pomocou nich ho implementovať. Tento diagram sa budeme ale snažiť nahradiť fragmentom.



Obr. 3 Diagram modelu pre štruktúrované aktivity

Podrobný opis týchto tried ako aj tried z activity diagramu môžete nájsť v Analýze Activity diagramu[1] a v špecifikácii UML[2].

[1] http://labss2.fiit.stuba.sk/TeamProject/2014/team04is-si/documents/ActivityDiagram.pdf

[2] http://labss2.fiit.stuba.sk/TeamProject/2014/team04is-si/documents/UML25.pdf

4. Analýza

V rámci zimného semestra bol analyzovaný existujúci diagram aktivít klasického 2D UML a taktiež fragmenty zo sekvenčného diagramu. Analýza metamodelu Activity diagramu

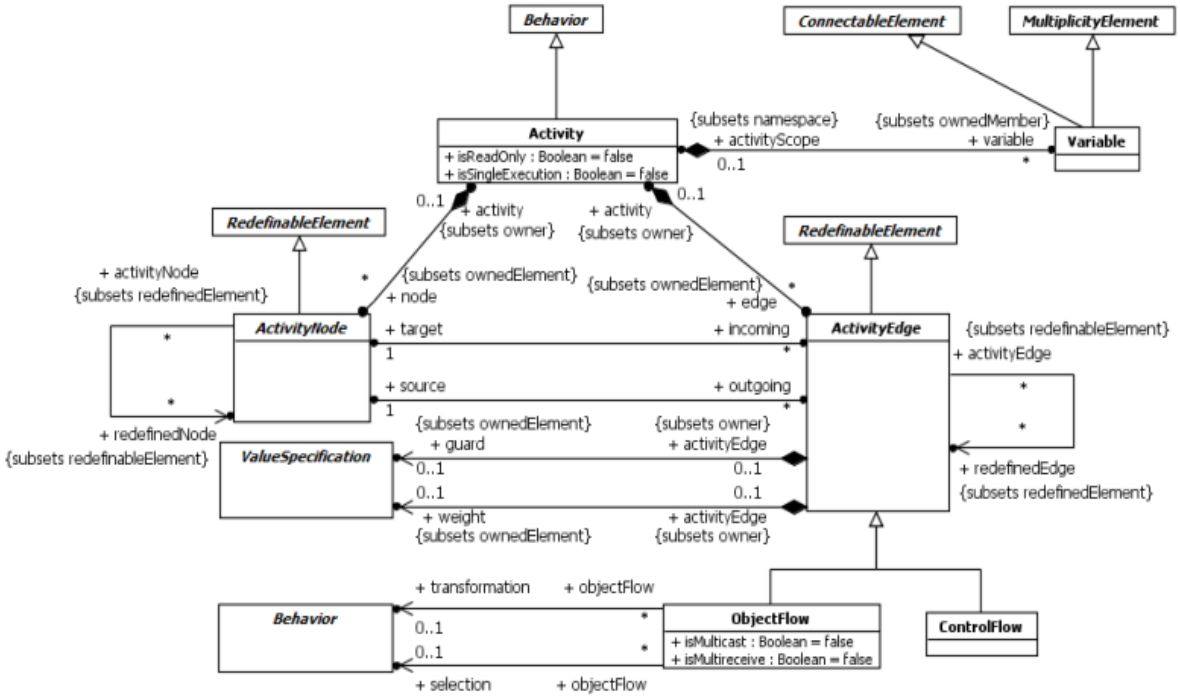
4.1.1. Diagram aktivít

Základným komponentom diagramu aktivít je samotná entita *Activity*, v rámci ktorej modelujeme jej priebeh a tok (pomocou do nej vnorených entít). Každá z aktivít sa podľa metamodelu skladá z 2 základných entít a to hrán (ActivityEdge) a uzlov (ActivityNode).
Uzly delíme do 3 základných skupín:

- Kontrolné uzly (***ControlNodes***), ktoré zabezpečujú vetvenie a riadenie toku
- Objektové uzly (***ObjectNodes***), ktoré definujú manipulované objekty
- Vykonateľné uzly (***ExecutableNodes***), ktoré vykonávajú akciu, resp. manipulujú s objektami

Hrany rozlišujeme:

- Objektovo riadiace (***ObjectFlow***), ktoré znázorňujú pohyb objektov
- Riadiace tok (***ControlFlow***), ktoré znázorňujú logiku toku v diagrame



Obr. 4 Metamodel diagramu aktív

Základným princípom fungovania diagramu aktivít je monitorovanie toku, ktorý je reprezentovaný fingovaným posúvaním Tokenov medzi jednotlivými uzlami. Tokeny môžu byť objektové, teda de-facto reprezentujúce manipulované objekty (vrátane vlastností objektov), alebo kontrolné, ktorých úlohou je ovplyvňovať činnosť uzlov, no nenesú žiadne informácie a pohybujú sa iba po hranách riadiacich tok.

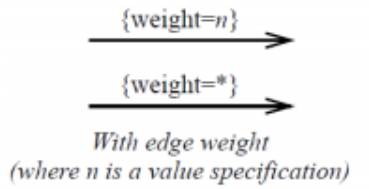
Každá z hrán (ObjectFlow/ControlFlow) môže byť nositeľom podmienky (Guard) v tvare „[need_to_be_met]“, až po ktorej naplnení môže posúvať tok k ďalšiemu z uzlov. Každá z hrán má taktiež vlastnosť *weight*, ktorá špecifikuje minimálny počet tokenov, ktoré musia cez danú hranu paralelne prechádzať. Ak cez danú hranu prechádza menej tokenov, daná hrana ich ďalej neprepustí.

Hrany riadiace tok objektov môžu obsahovať aj 2 základné funkcie a to *transform* a *select*. Transform mení vstupné tokeny na modifikované výstupné pre cieľový ActivityNode. Select aplikuje zvolený filter na vstupné tokeny a len tokeny ktoré prejdú testom, sú následne posunuté cieľovému ActivityNode-u.

Samotný element *Activity* môže podľa metamodelu obsahovať aj premenné, ktoré môžu byť počas vykonávania toku globálne modifikované. Zároveň môže mať aj predpoklady a dôsledky (*pre- and post-condtitions*) pre vykonanie. Aktivita ako behaviorálny element môže mať aj parametre, čo sú vstupné a výstupné štruktúry (reprezentované ako ActivityParameterNodes) očakávané pri inicializácií, resp. finalizácií danej aktivity. Tieto uzly potom posúvajú svoje tokeny ďalej k toku cez hrany. Jednou zo základných funkcií aktivity je funkcia *isSingleExecution()*, ktorá definuje, či počas vykonávania danej aktivity môžu do danej aktivity paralelne vstupovať ďalšie parametrické objekty (teda či môže byť aktivita vyvolaná druhý krát už počas vykonávania, alebo až po vykonaní predchádzajúceho volania).

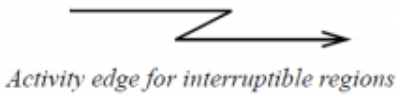
4.1.2. Grafická notácia

Hrany



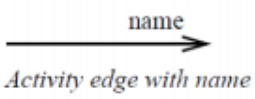
Obr. 5 Hrana s definovanou *weight*

Štandardná notácie pre hranu, ktorá ma definovanú *weight*, teda počet tokenov, ktoré môže prenášať.



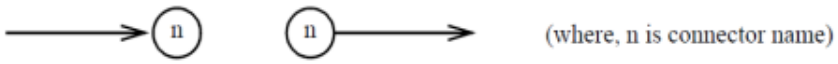
Obr. 6 Hrana vyjadrujúca tok

Hrana vyjadrujúca tok, ktorý sa aktivuje, ak dôjde k nekorektnému ukončeniu akcie.



Obr. 7 Hrana s popisom

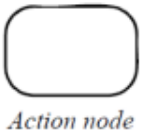
Popis a meno hrany sa píšu nad hranu, môže obsahovať aj podmienku na prepustenie toku (v hranatých zátvorkách)



Obr. 8 Zapís hrany cez konektor

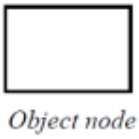
Verzia zápisu hrany cez konektor (len kvôli prehľadnosti)

Uzly



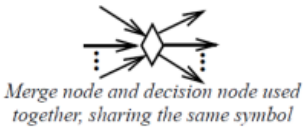
Obr. 9 Uzol akcie

Štandardná akcia. Príklad akcie môže byť *odoslanie zásielky* a pod.



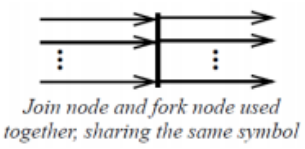
Obr. 10 Uzol objektu

Objekt môže byť reprezentovaný napríklad triedou z Class diagramu, tu bude z pohľadu viacerých vrstiev v 3D UML treba vytvoriť možné prepojenie medzi rôznymi diagramami. Meno objektu môže byť priamo napísané v objekte, zvykne byť podčiarknuté. Do hranatých zátvoriek je potom možné znázorniť stav objektu.



Obr. 11 Decision a Merge

Decision a Merge. Reprezentované budú ako 1 trieda, keďže môže existovať aj skombinovaný objekt podľa obrázka. To či bude objekt fungovať ako Decision, alebo ako Merge bude jasné podľa toho, koľko hrán doňho vstupuje a vystupuje. Merge funguje ako OR, teda prepúšťa tok, ak príde tok z aspoň jednej zo vstupných hrán.



Obr. 12 Fork a Join

Podobne funguje aj Fork/Join. Fork a Join vyjadrujú paralelné vykonávanie tokov. Join prepúšťa tok, až keď doň vstúpia všetky vstupné hrany. Funguje teda ako AND.

Obr. 13 Štart celého toku

Obr. 14 Ukončenie celej aktivity/ukončenie včetkých ost. tokov

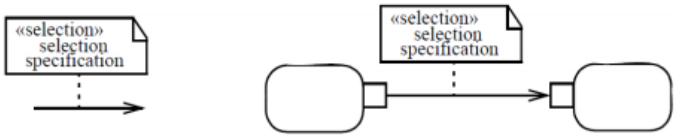
Ukončenie celej aktivity a paralelne aj ukončenie všetkých ostatných tokov.

Obr. 15 Ukončenie danej vetvy toku

Ukončenie danej vetvy toku. Ostatné vetvy pokračujú nezávisle.

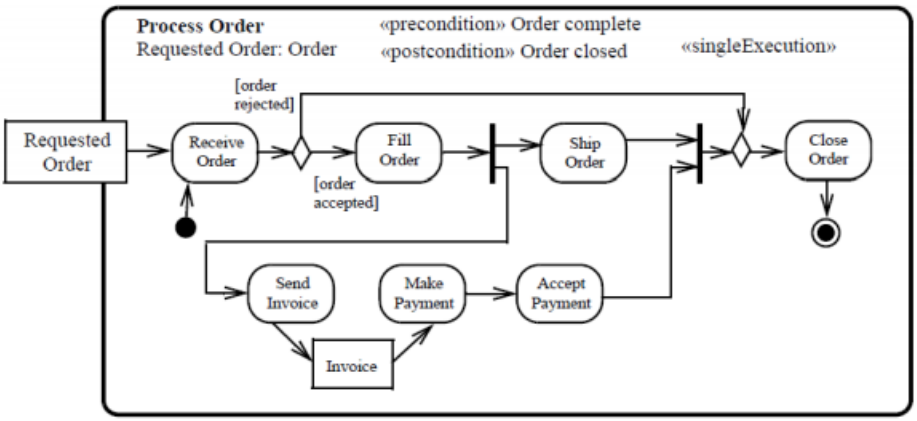
Obr. 16 Konektory (piny)

Konektory (piny) sú iným zápisom pre prepojenie 2 akcií posielaním objektu. Výstupný pin z akcie definuje výstupný objekt a vstupný pin vstupný objekt. Medzi dvoma akciami môže byť vymieňaných aj viac objektov, preto môže byť aj viac paralelných pinových prepojení.



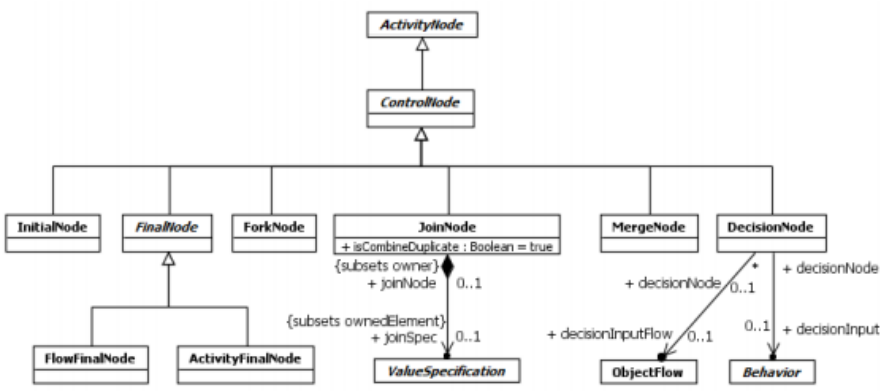
Obr. 17 Anotácia

Popis konkrétnej selekcie, alebo transformácie môže byť zaznamenaný v anotácií v príslušnom stereotype.



Obr. 18 Príklad pre zápis modelovej aktivity

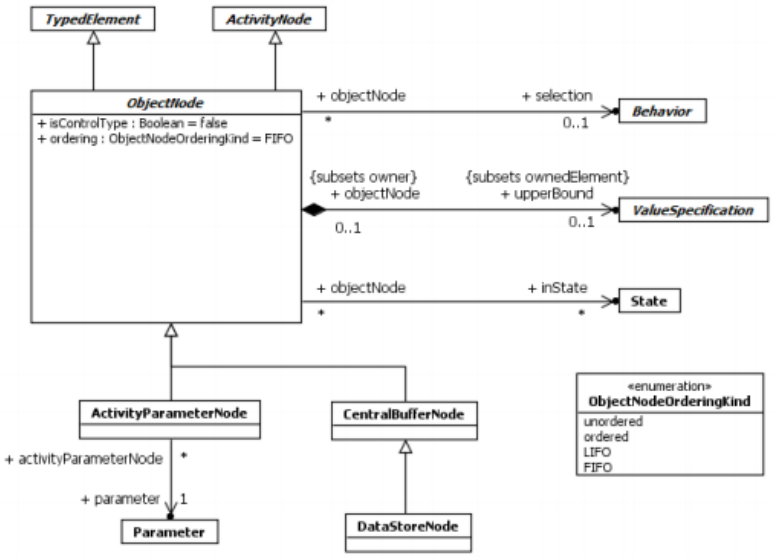
4.1.3. Metamodel kontrolných uzlov



Obr. 19 Metamodel kontrolných uzlov

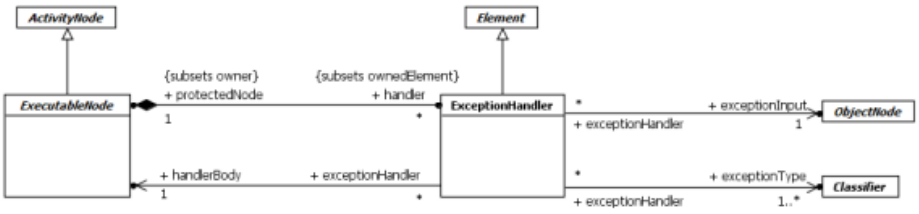
Funkcia isCombinesDuplicate definuje, či ak príde na Join elemtn viacero tokenov s rovnakým obsahom, tak má Join brána posunúť tieto duplikované tokeny ďalej všetky, alebo len 1.

4.1.4. Metamodel objektových uzlov



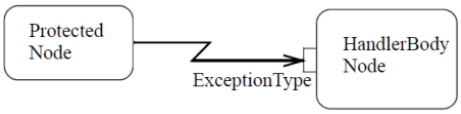
Obr. 20 Metamodel objektových uzlov

4.1.5. Metamodel výkonateľných uzlov



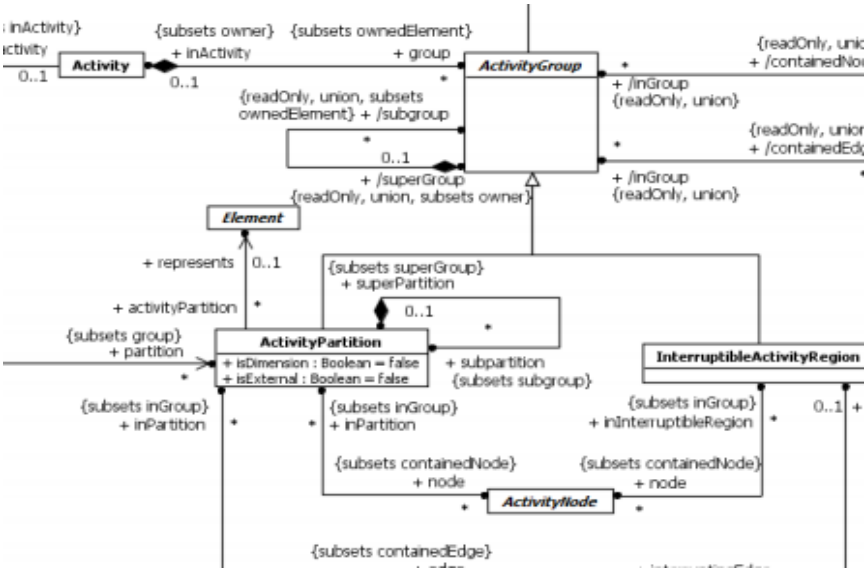
Obr. 21 Metamodel vykonateľných uzlov

Typickým vykonateľným uzlom je akcia. V tele vykonateľného uzla budú zadefinované funkcie ktoré bude vykonávať, pričom ExceptionHandler následne zabezpečuje vykonávanie v prípade ak dôjde k nepredvídanej chybe pri realizácií pôvodnej funkcie (ExecuteNode vyvolá ExceptionHandler).



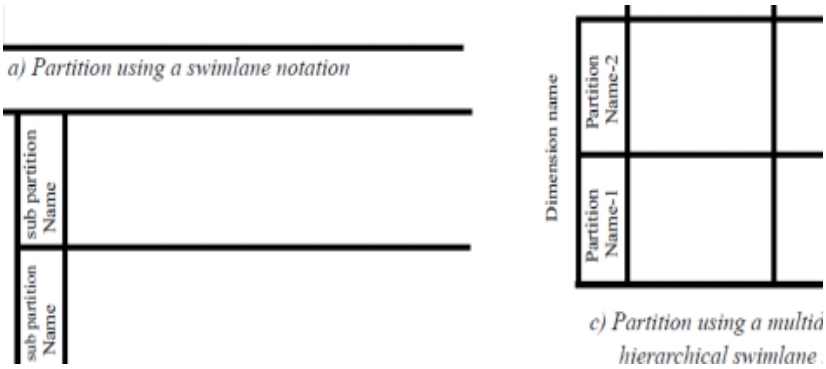
Obr. 22 ExceptionHandler

4.1.6. Metamodel združovacích entít



Obr. 23 Metamodel združovacích entít

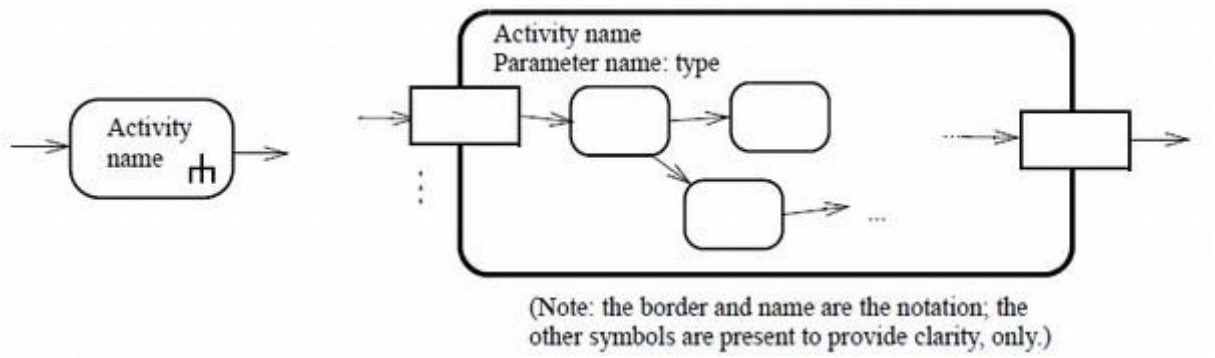
Úlohou združovacích entít (ActivityPartitions) je zoskupovať elementy na základe špecifickej vlastnosti. Vo všeobecnosti obsahujú funkcie isDimension, ktorá ak je nastavená na True hovorí, že daná skupina objektov (Partition) nemôže byť už v rámci danej aktivity vnorená do ďalšej skupiny objektov. Všetky objekty v rámci danej skupiny musia mať priradený spoločný klasifikačný objekt (Classifier). Funkcia isExternal hovorí o objektoch, resp. subpartíciách, ktoré majú v rámci väčšej skupiny priradený iný klasifikačný objekt, no stále sa podieľajú na výkone danej skupiny akcií (ide o výnimku).



Obr. 24 Vyjadrenie ActivityPartition za pomoci notácie plaveckých dráh

Najčastejšou formou vyjadrenia ActivityPartition je využitie notácie plaveckých dráh. V prípade zavedenia fragmentou zavedieme nové triedy, ktoré budú dediť od triedy ActivityPartition, budú mať vlastnú notáciu typickú pre fragmenty a vlastné vlastnosti.

Ďalšou z častí využívaných pri zoskupovaní sú prerušiteľné skupiny aktivít (*interruptable activity regions*)



Obr. 25 Prerušiteľné skupiny aktív

4.1.7. Štruktúra akcie

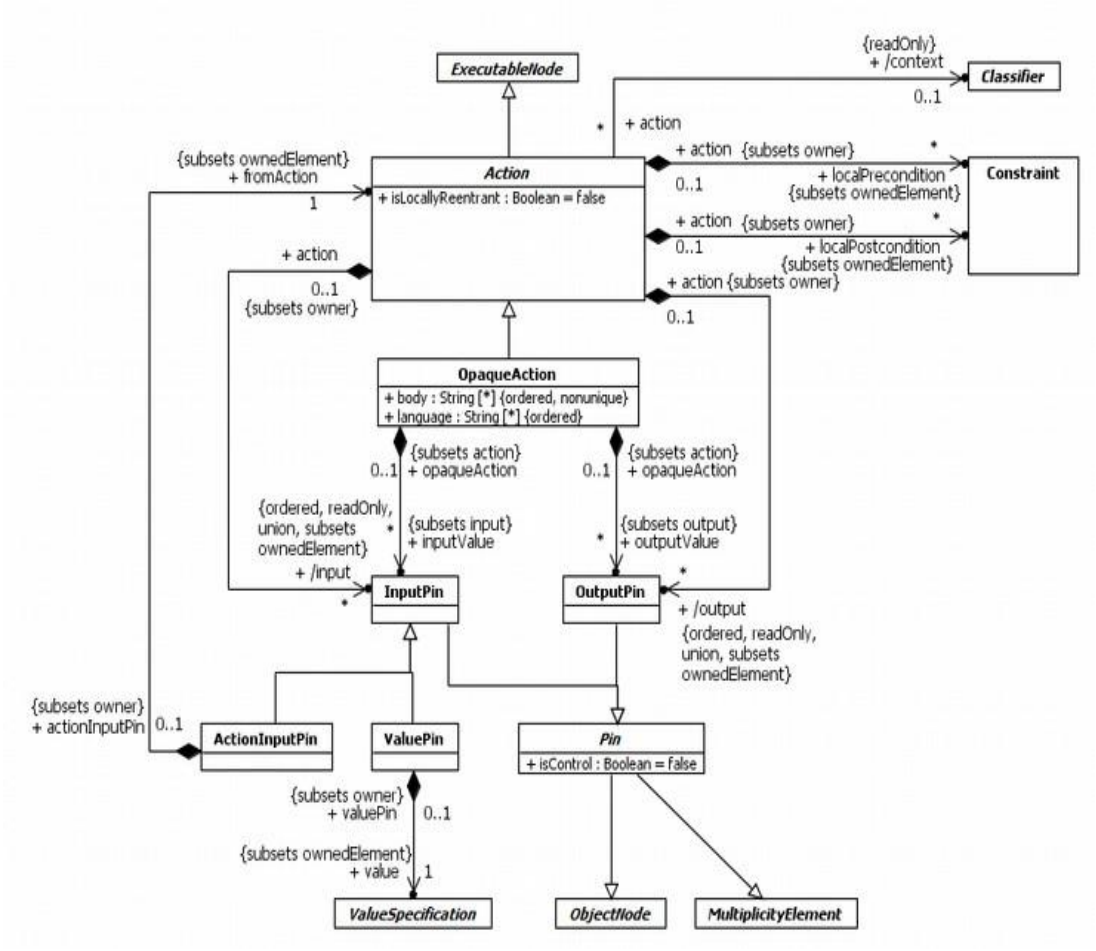
Akciu môžeme vníma všeobecne ako spúšťač. Najčastejšie akcia spúšťa špecifické správanie sa, alebo funkciu, ktorá je v UML reprezentovaná triedou *Behaviour*. V špecifických prípadoch môže akcia spúšťať aj sled akcií v podobe druhej aktivity, vtedy využívame v notácií piktogram trojzubca.

Akcie vo všeobecnosti delíme na 2 typy:

- Akcie ktoré vyvolávajú správanie
- Akcie ktoré preposielajú signály/objekty

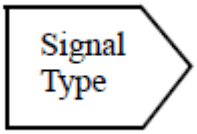
Akcie vyvolávajúce správanie môžu vyvolávať priamo objekt reprezentujúci správanie (správanie sa systému), môžu vyslať správu spracúvanému objektu aby vyvolal správanie (systémové), alebo môžu inicializovať priamo správanie sa objektu. Toto budeme realizovať cez overloadig. Volanie môže by synchrónne alebo asynchrónne. Ak je volanie synchrónne, daná akcia bude ukončená až vtedy, keď bude ukončené aj správanie sa, ktoré inicializovala.

Parameter isLocallyReentrant definuje, či môže by akcia znovu vyvolaná skôr, než sa ukončilo predošlé volanie a vykonávanie sa.



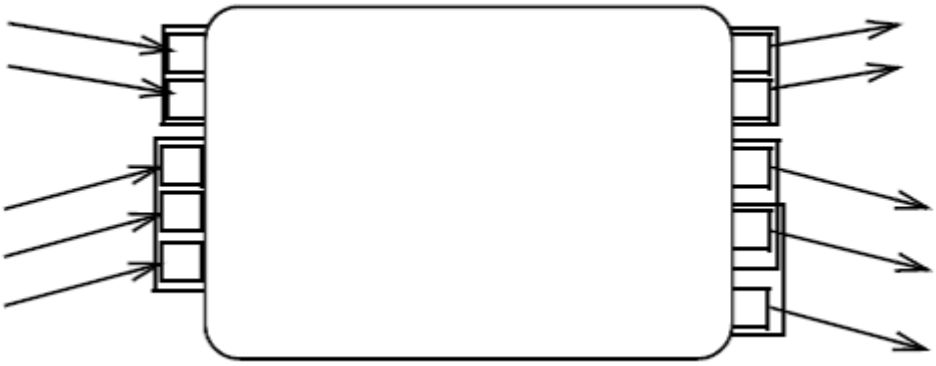
Obr. 26 Štruktúra akcie

Akcie preposielajúce signály môžu vyslať signál na jeden odchodzí pin, môžu vyslať signál na všetky odchodzie piny (broadcast), alebo môžu odoslať na pin špeciálny tip objektu (Ak odošlú objekt typu signál, ide o prvú zo spomenutých možností).



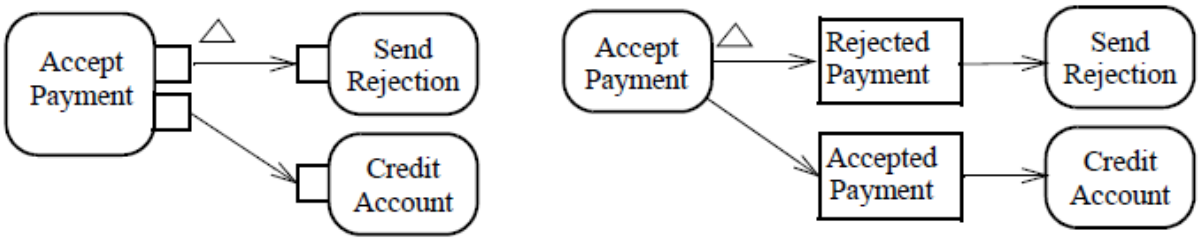
Obr. 27 Notácia znázorňujúca akciu, ktorá vysiela objekt typu signál.

Jednotlivé vstupné a výstupné piny môžu by aj zoskupované do takzvaných parametrických skupín. Parametrická skupina vyjadruje, že akcia bude spustená (nad daným objektom) až po prijatí všetkých objektov patriacich do daného set-u a paralelne, objekty budú vyslané naraz, až keď budú pripravené všetky podľa príslušnosti.



Obr. 28 Parametrická skupina

V prípade, ak prenášaný objekt je výnimkou (exception) reprezentujúcou spustenie alternatívneho toku, tak v notácií využívame and hranou malý trojuholník.



Obr. 29 Príklad využitia malého trojuholníka nad hranou na spustenie alt. toku

Štruktúrované akcie

Štruktúrované akcie presne zodpovedajú fragmentom, ktoré je naším cieľom do modelu zaviesť. Špecifikácia vraví, že nie ej definovaný štandardná notácia pre podmienky, slučky a sekvencie a preto zavedieme notáciu zo sekvenčného diagramu.

Slučka sa ako trieda skladá z 3 základných častí:

- setupPart, kde sa
- test
- bodyPart, ktorý zahŕňa skupinu vykonateľných uzlov.

Pri vstupe do slučky sú automaticky povolené všetky inicializačné uzly (InitialNodes). Vykonateľné uzly sú povolené len v prípade, že sa povolí vstup do bodyPart.

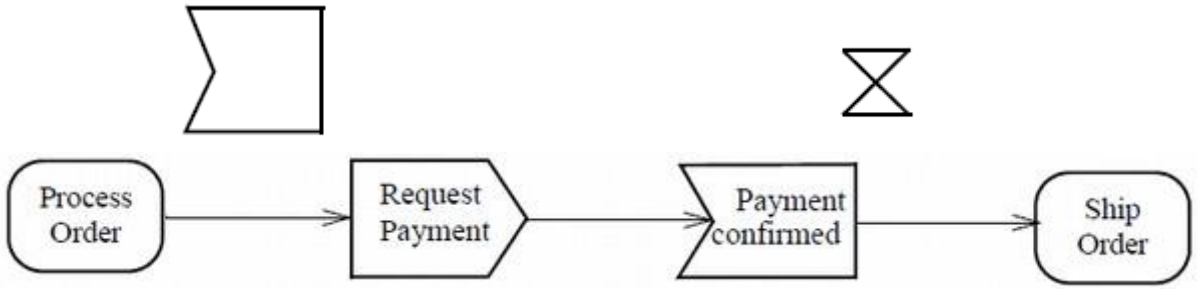
setupPart je prediteračná fáza. Po nej môže nastať priamo fáza bodyPart, alebo test a to podľa toho, či je nastavené isTestFirst. Pri jednotlivých iteráciách si slučka posúva dáta pomocou objektov na jednotlivých pinoch (loopVariable outpu pins, bodyOutput output pins, result output pins). Začiatok vykonávania slučky je zabezpečený presunov tokenov z loopVariableInput pinov na loopVariable outpu piny.

Podmienka (ConditionalNode) sa skladá z viac klauzúl, ktoré, reprezentujú jednotlivé vetvy toku. Každá z klauzúl sa skladá zo sekcie bodyPart a test a funguje podobne ako pri slučke.

Niekedy je vhodné, aby dáta spracované v slučke, alebo inej štruktúrovanej akcií boli izolované a nedošlo tak k ich modifikácia z externého toku. Na to slúži značka mustIsolate.

Akcie schvaľujúce udalosti

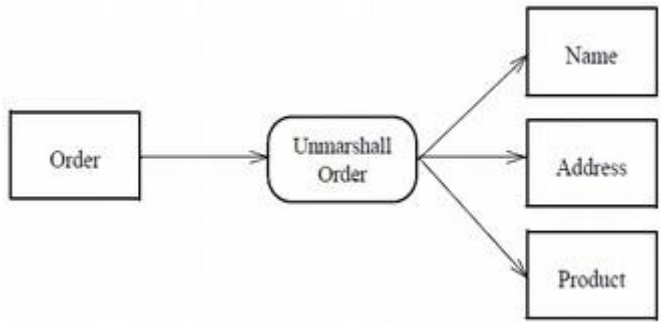
Takéto akcie vnímame ako spúšťače (Triggers) jedného, alebo viacerých udalostí. Nie sú to, ale akcie ako také, sú to len prvky, ktoré vyčkávajú na vykonanie istej udalosti, resp. prijatí signálu a až následne spustia ďalšiu akciu. Sled akcií ktoré tieto spúšťače sledujú sa nachádza v tzv. Event poole.



Obr. 30 Priklad akcie odoslania zásielky

Presýpacie hodiny znázorňujú časový spúšťač. Príkladom môže by odoslanie zásielky každý mesiac. Časový údaj sa k entite pripisuje ako text. Akcie schvaľujúce udalosti nemusia mať do seba primárne vchádzajúci kontrolný tok, v prípade, ak je ich spustenie inicializované napríklad udalosťou externou k danej aktivite.

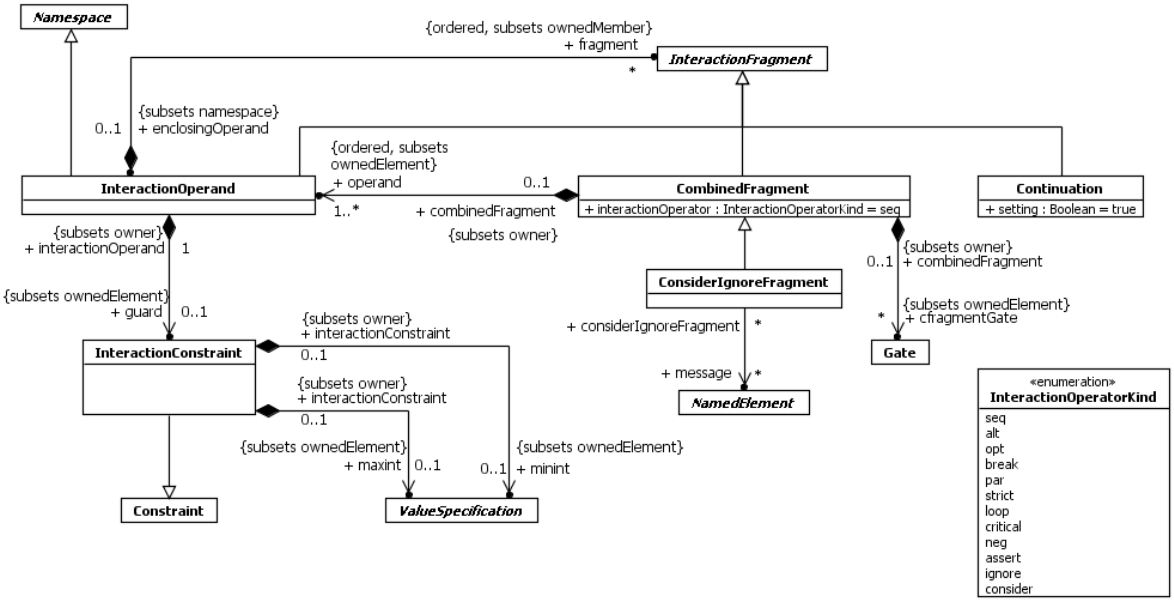
Špecifické postavenie majú tzv. rozkladné akcie (unmarshall actions), ktorých úlohou je z prichodeného objektu vyparsovať špecifické dáta a podľa nastavených kritérií ich následne postúpiť na odchodzie objekty.



Obr. 31 Rozkladná akcia

4.2. Analýza metamodelu fragmentov

4.2.1. Abstraktná syntax



Obr. 32 Metamodel fragmentu

Interakčný operand:

Predstavuje oblasť vyhradenú Kombinovaným fragmentom. Aby bol operand vykonaný, musí mať pravdivé ohraničenie. Ak ohraničenie nie je definované, automaticky sa berie, ako pravdivé.

Interakčné ohraničenie:

Používajú sa v kombinácií s kombinovanými fragmentami.

Kombinovaný fragment:

Sémantika je závislá na interakčnom operátore. Prvok „Gate“ reprezentuje syntaktické rozhranie medzi kombinovaným fragmentom a jeho okolím.

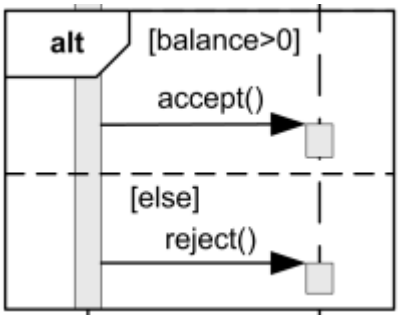
4.2.2. Kombinované fragmenty

Kombinované fragmenty sú fragmenty, ktoré definujú výraz, na základe interakčných fragmentov. Kombinačné fragmenty sú definované pomocou interakčného operátora a interakčných operandov.

Medzi interakčne operátory patria:

Alternatives:

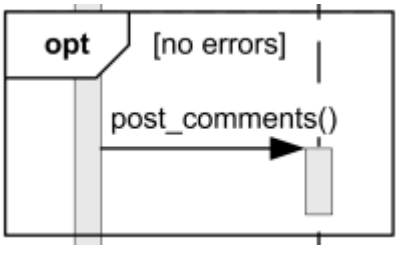
Operátor označuje možnosť výberu medzi viacerými tokmi. Výber operandu závisí na definovanom ohraničení, ktoré musí byť explicitne uvedené



Obr. 33 Príklad alternatívy

Option:

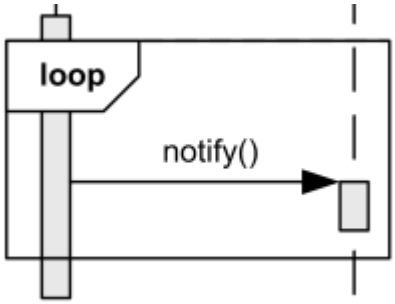
Operátor označuje možnosť, medzi vykonaním operadnu, alebo nevykonaním. Rozhodnutie o vykonaní závisí na vyhodnotení ohraničujúcej podmienky, ktorá musí byť explicitne stanovená.



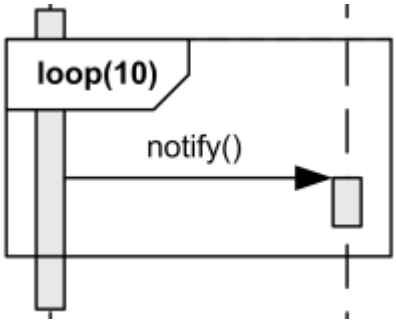
Obr. 34 Príklad možnosti

Loop:

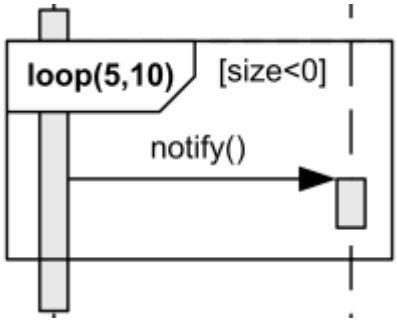
Operand v operátore „loop“ sa vykoná stanovený počet krát. Počet, koľko krát sa operand vykoná závisí na stanovenom ohraničení. V prípade, že ohraničenie nie je stanovené, ide o nekonečný cyklus. V prípade zadanie jednej hranice sa cyklus vykoná presne stanovený počet krát. Je možné taktiež zadať hornú aj dolnú hranicu. Okrem toho môže byť uvedené aj ďalšie ohraničenie, ktoré podmieňuje samotné spustenie cyklu.



Obr. 35 Nekonečný cyklus



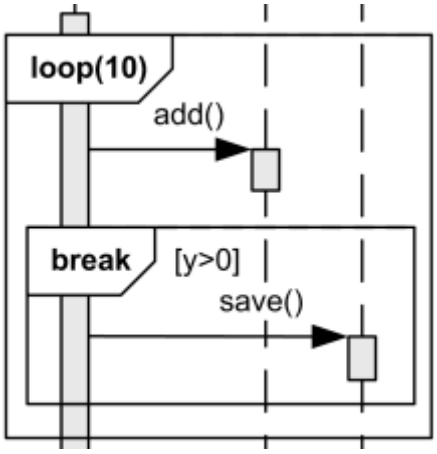
Obr. 36 Cyklus opakujúci sa 10-krát



Obr. 37 Cyklus s minimálnym a maximálnym ohraničením

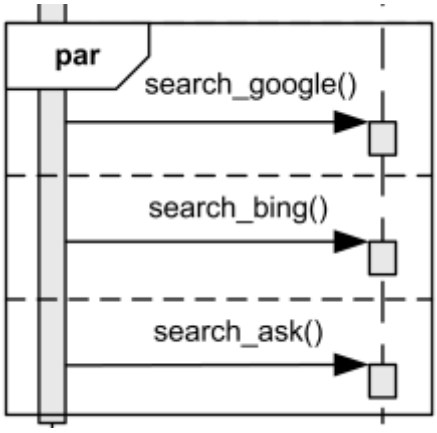
Break:

Operátor sa používa na ukončenie fragmentu, do ktorého je vnorený. Ukončenie nastáva, ak je stanovené ohraničenie splnené. V prípade, že podmienka nie je uvedené ide o nedeterministické správanie.



Obr. 38 Príklad ukončenia

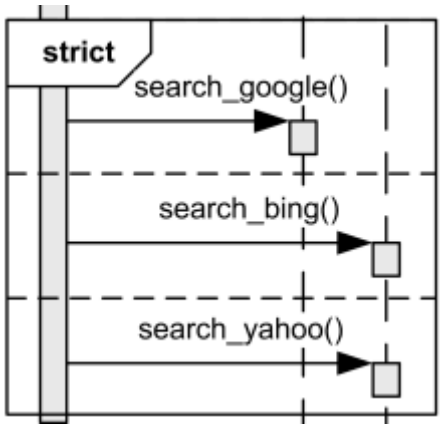
Parallel:
Označuje paralelizmus. Každá časť sa vykoná paralelne s ostatnými časťami operátora.



Obr. 39 Príklad paralelizmu

Strict Sequencing:

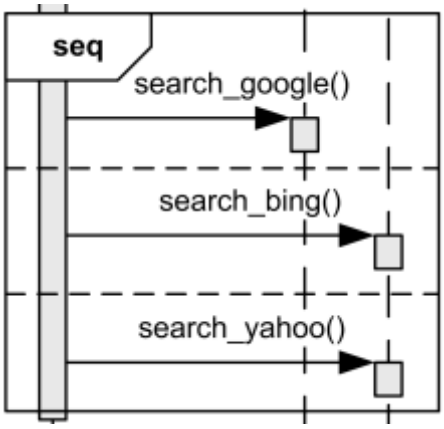
Operátor označuje, že operátory musia byť vykonané v presne stanovenom poradí.



Obr. 40 Prísna sekvencia

Weak Sequencing:

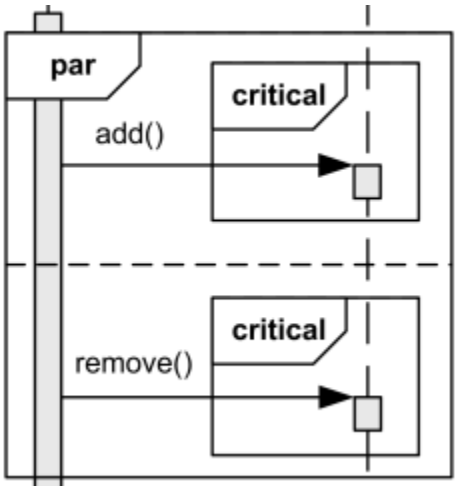
Operátor označuje sekvenciu, ktorá, na rozdiel od predchádzajúceho prípadu, nemusí byť prísne po poradí. Konkrétne to znamená to, že operandy na jednej línii sa musia vykonať v poradí, no na rôznych líniách na poradí nezáleží.



Obr. 41 Príklad sekvencie

Critical Region:

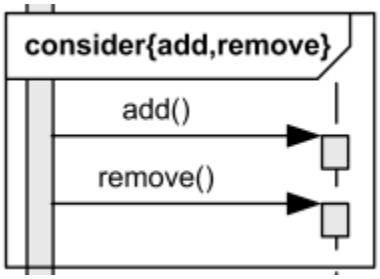
Operand špecifikuje kritický región, ku ktorému je nutné pristupovať atomicky.



Obr. 42 Príklad kritickej oblasti

Consider:

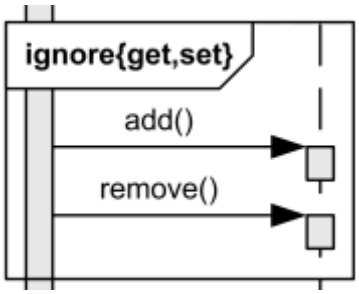
Operátor označuje oblasť, v ktorej musí byť zvážené, ktorá z poskytnutých správ bude zvolená. Zoznam poskytnutých správ na výber sa musí nachádzať v „{}“ zátvorkách. Iba jedna správa bude volaná, ostatné sa ignorujú.



Obr. 43 Príklad zváženia

Ignore:

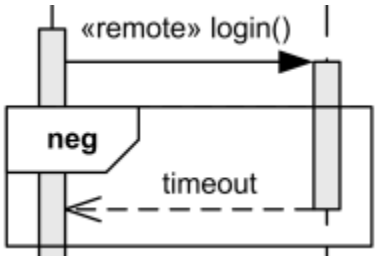
Označuje časti, ktoré nemajú byť zobrazené.



Obr. 44 Príklad ignorovania

Negative:

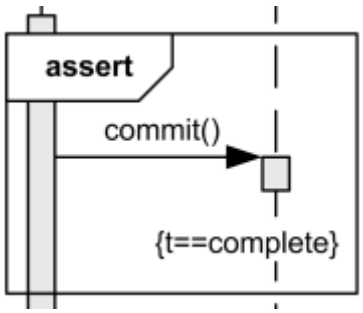
Operátor označuje oblasť, ktorá sa vykonáva v prípade neúspešného volania. Napríklad v prípade pádu systému. Všetky ostatné typy fragmentov sú považované za pozitívne.



Obr. 45 Príklad negative

Assertion:

Operátor označuje oblasť, ktorá musí byť úspešne ukončená pre korektné pokračovanie programu.



Obr. 46 Príklad assertu

4.2.3. Notácia

Interakčný operand:

Interakčné operandy sú od seba oddelené vodorovnou prerušovanou čiarou. Spolu tvoria orámovaný kombinovaný fragment. V sekvenčnom diagrame je poradie operandov dané vertikálnou polohou operandu.

Interakčné ohraničenie:

Interakčné ohraničenie sa uvádza v hranatých zátvorkách. Majú tvar: <interactionconstraint> ::= ‘[‘ (<Boolean-expression> | ‘else‘) ‘]’

Pokiaľ ohraničenie nie je uvedené, predpokladá sa pravdivé tvrdenie.

Kombinovaný fragment:

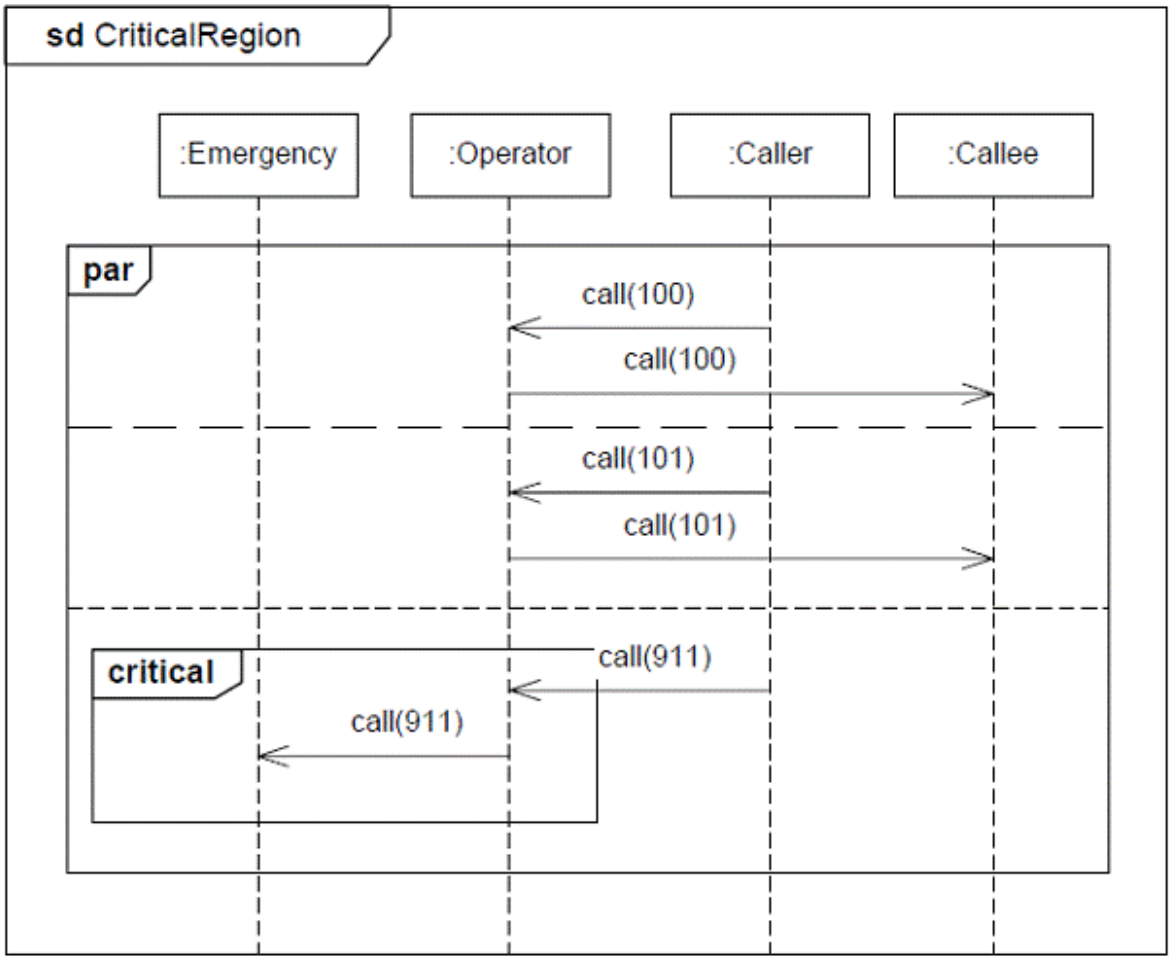
V sekvenčnom diagrame sa uvádza, ako obdĺžnik. Operátor je uvedený v päťuholníku v ľavom hornom rohu.

Consider / Ignore Fragment:

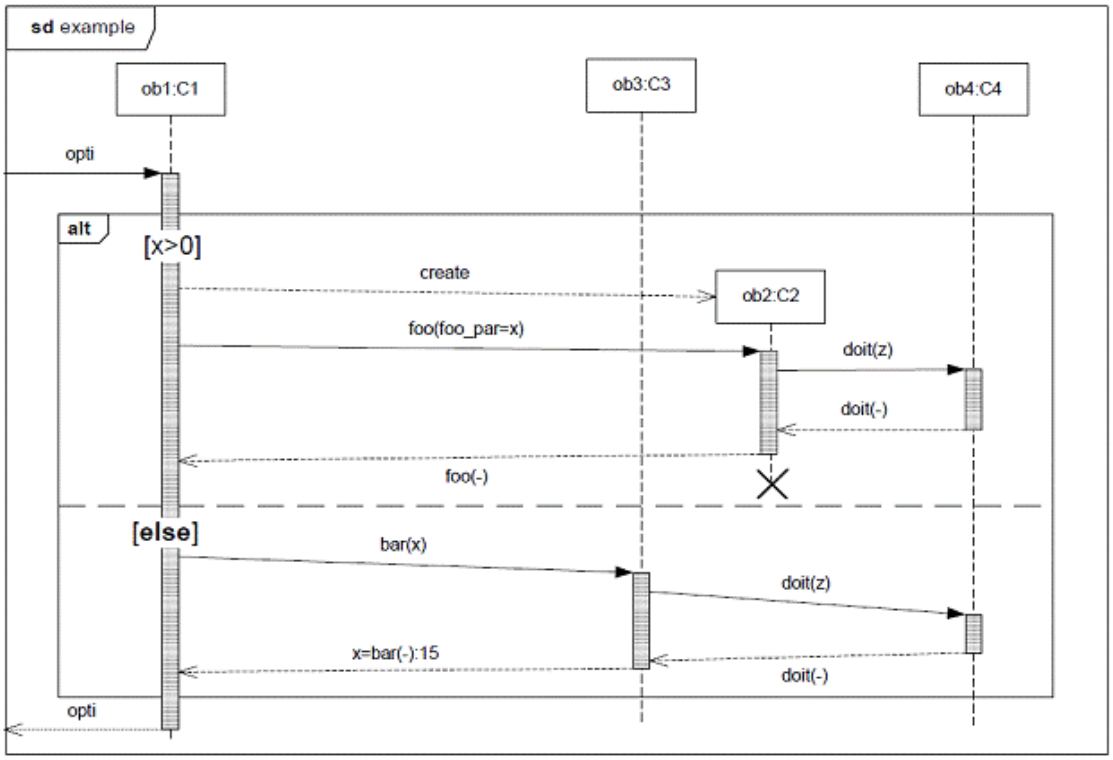
Na rozdiel od ostatných kombinovaných fragmentov sa za operátorom ešte uvádza zoznam parametrov uvedený v „{}“.

(‘ignore’ | ‘consider’) ‘{‘ <message-name> [‘,’ <message-name>]* ‘}’

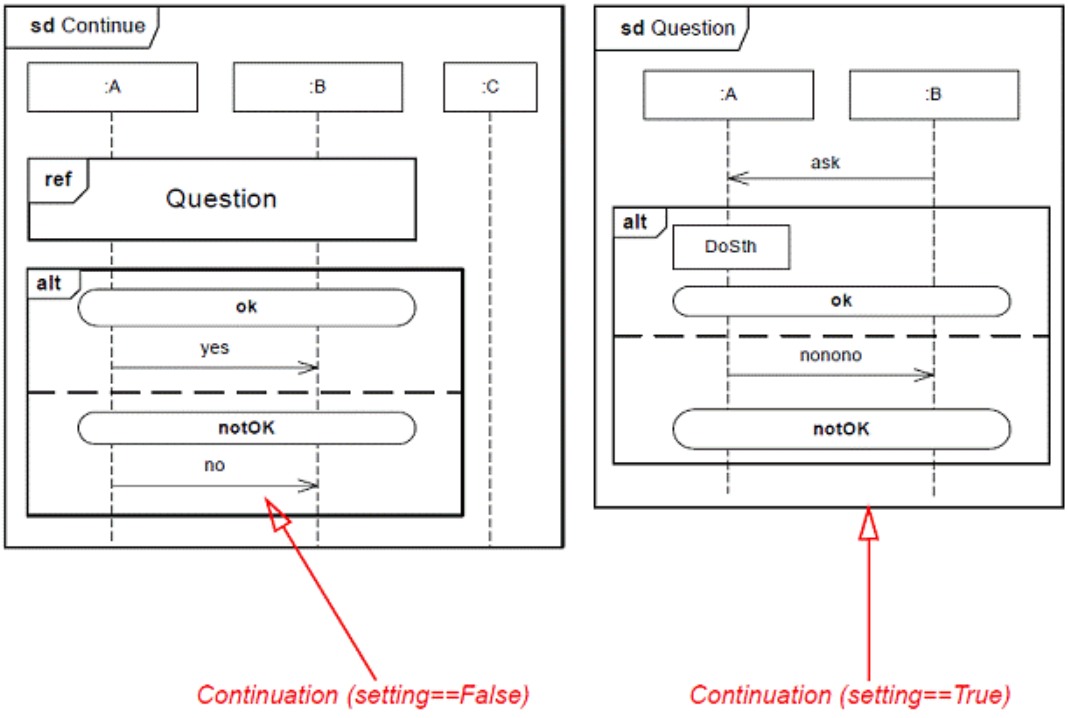
4.2.4. Príklady kombinovaných fragmentov zo špecifikácie UML



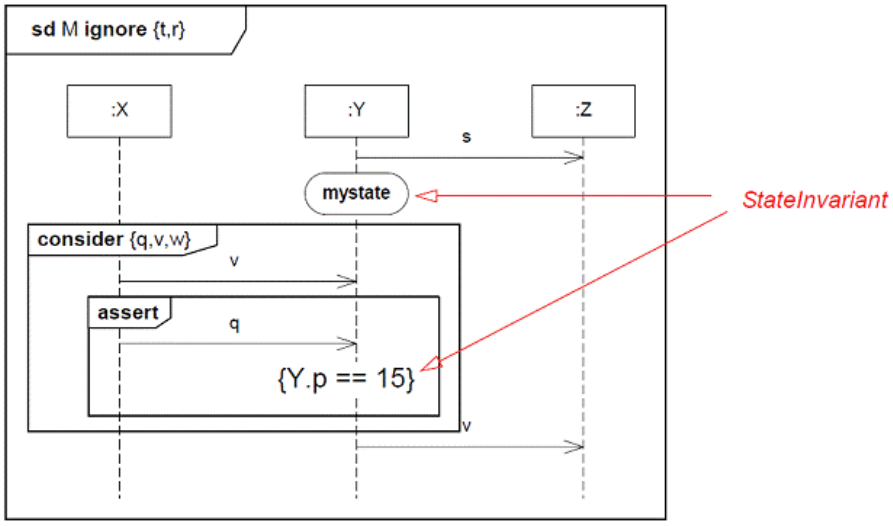
Obr. 47 Príklad kombinovaného fragmentu



Obr. 48 Príklad kombinovaného fragmentu



Obr. 49 Príklad kombinovaného fragmentu

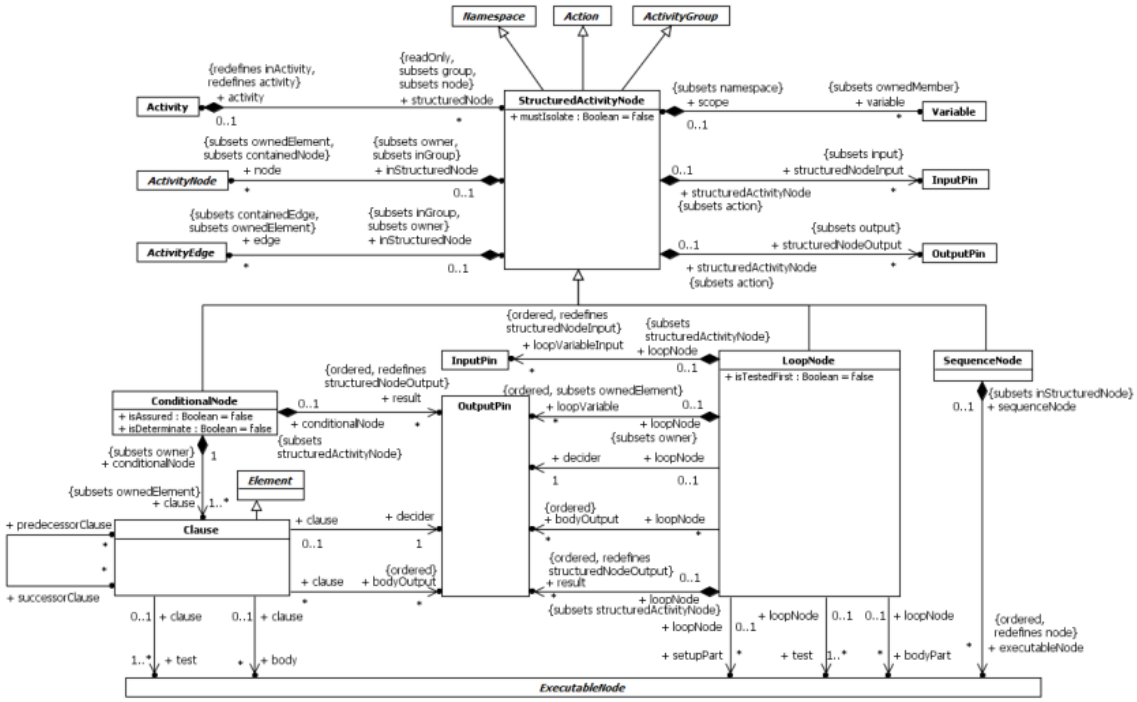


Obr. 50 Príklad kombinovaného fragmentu

5. Návrh riešenia

5.1. Prepojenie diagramu aktivít na fragmenty sekvenčného diagram

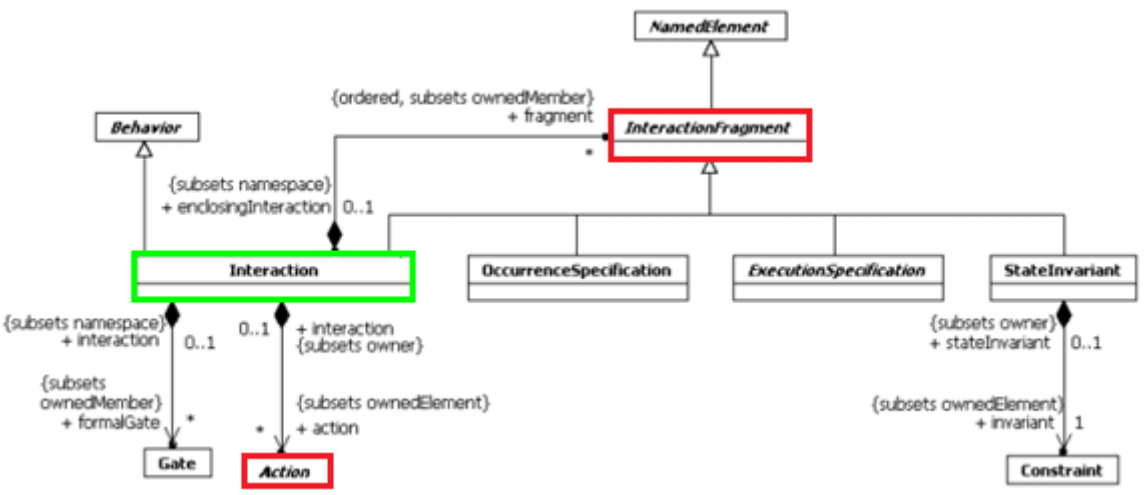
Diagram aktivít je podobne ako sekvenčný diagram, UML diagramom správania sa. Tento diagram samotný už obsahuje podľa špecifikácie štruktúrované entity, ktoré dovoľujú alternatívne vetvenie toku, prípadne tvorbu slučiek, problémom však je, že okrem limitovaného počtu akcií, ktoré nám tieto štruktúrované entity poskytujú, nie je podľa metamodelu takto zabezpečené vnáranie.



Obr. 51 Metamodel sekvenčného diagramu

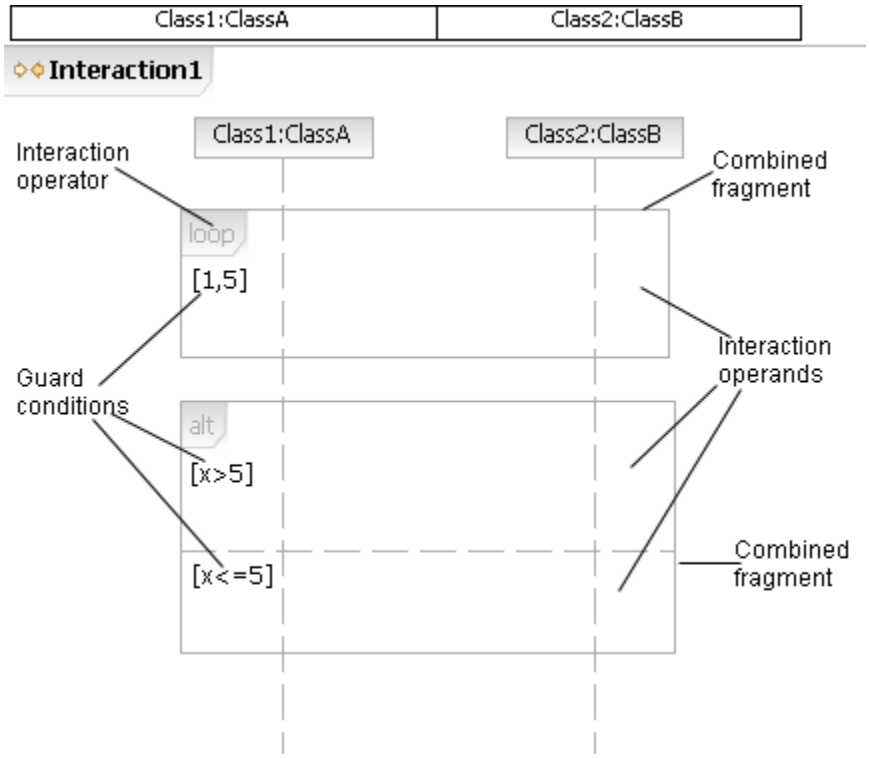
Našou úlohou, keďže chceme zachovať konzistenciu s metamodelom UML, je preto nájsť prepojenie medzi metamodelmi diagramu aktivít a sekvenčným diagramom, konkrétne fragmentami sekvenčného diagramu. Prepájacou entitou, ktorú vieme v tomto prípade identifikovať je entita interakcie. Interakciu vnímame ako špecializáciu triedy správanie

(Behavior), pričom každá z interakcií môže obsahovať n akcií, ale zároveň aj n interakčných fragmentov (toto zabezpečuje to spomínané prepojenie, ktoré hľadáme). Špecializáciou triedy InteractionFragment sú ďalej triedy InteractionOperand a InteractionFragment. Podľa špecifikácie je každý InteractionFragment súčasťou 0-1 InteractionOperandu a každý InteractionOperand je súčasťou z 0-1 CombinedFragmentu.



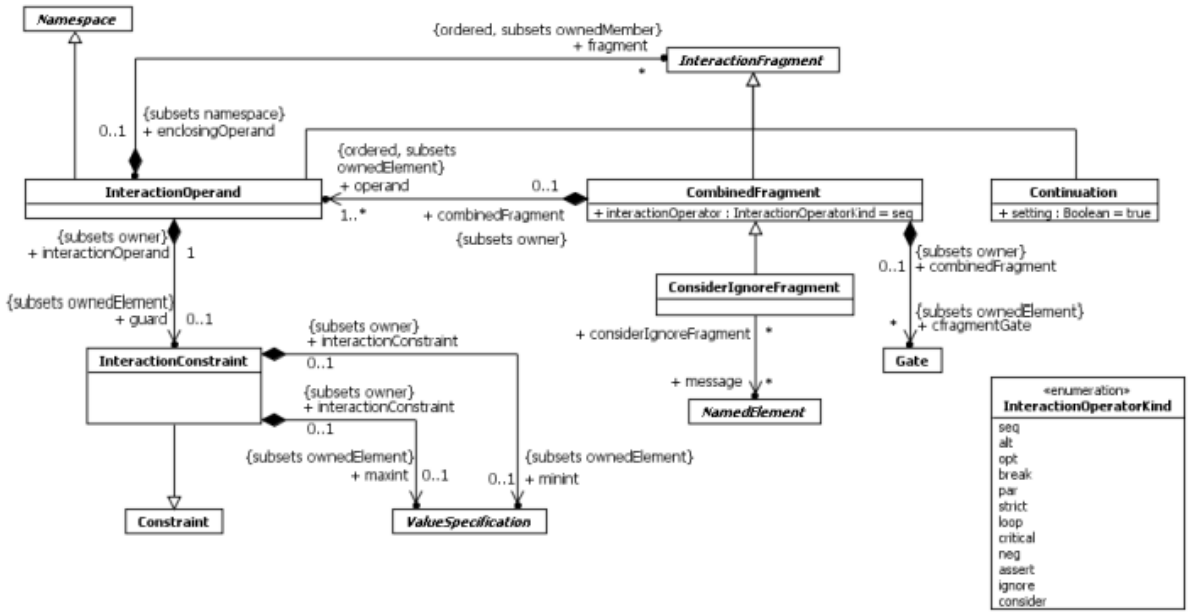
Obr. 52 Identifikovaná entita Interaction na prepojenie metamodelov diagramu aktív a sekvenčného diagramu

Zjednodušene povedané, Interaction fragment vnímame ako región akcií, teda akoby element zoskupujúci akcie, no neobsahujúci žiadnu logiku (nevie urobiť loop, alt,...), táto trieda bude len niesť základné informácie o tom, aké elementy zoskupuje a pod. CombinedFragment je potom element, ktorý od neho dedí a teda ho špecifikuje. CombinedFragment, tiež zoskupuje akcie, no na rozdiel od InteractionFragmentu už zahŕňa aj logiku, pretože obsahuje funkciu interactionOperator (pozri obrázok nižšie pre list možných operácií) a zároveň sa skladá z jedného alebo viac InteractionOperandov, teda entít zahŕňajúcich celý tok akcií v danej vetve.



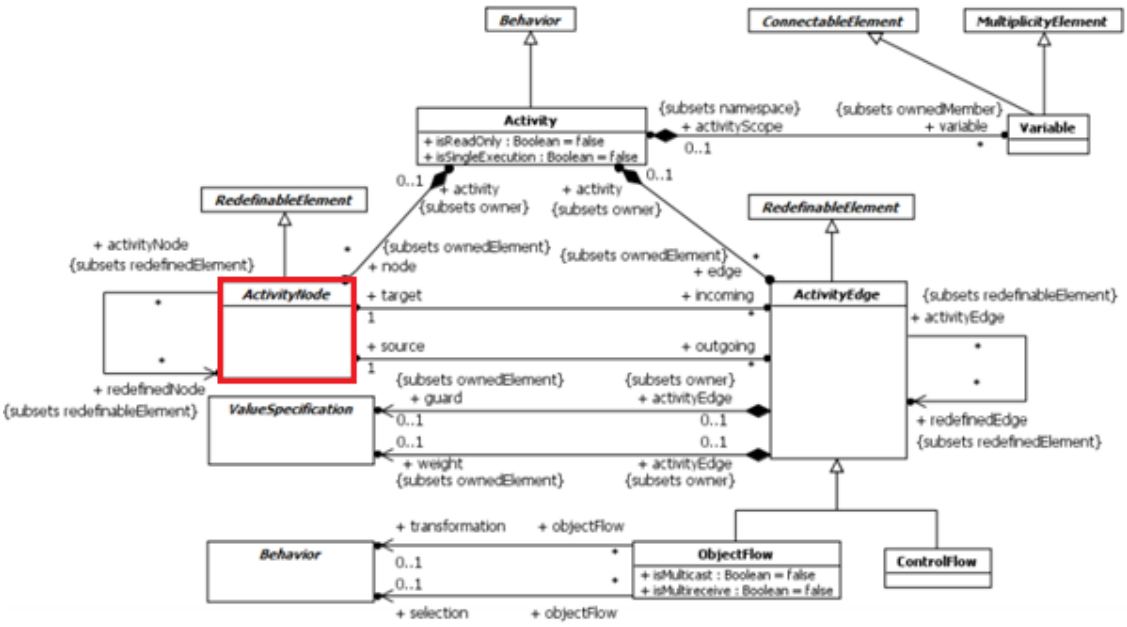
Obr. 53 Combined fragment

InteractionConstraint je trieda reprezentujúca podmienku vykonania, ktorá je priamo súčasťou zodpovedajúceho InteractionOperanda.



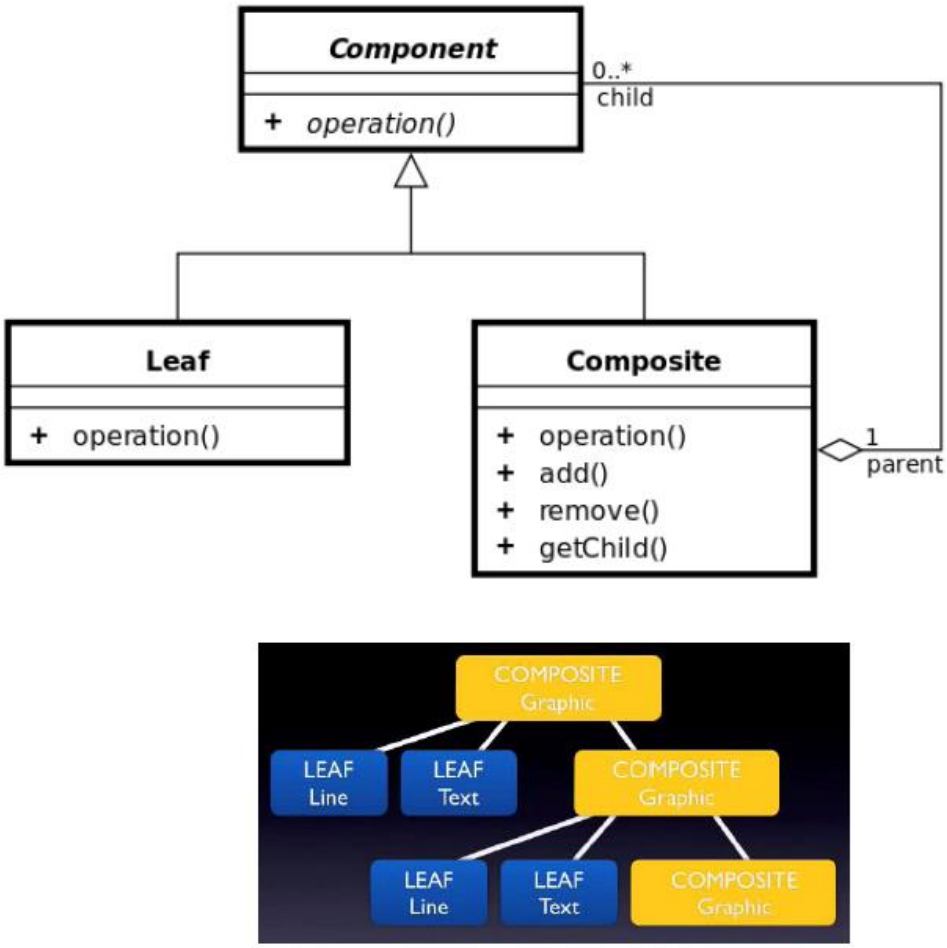
Obr. 54 InteractionOperand

Ak sa pozrieme na metamodel Diagramu aktivít vidíme, že akcia je typu ExecutableNode, čo je podtyp ActivityNode-u. Jeho usporiadanie v rámci diagramu aktivít je nasledovné:



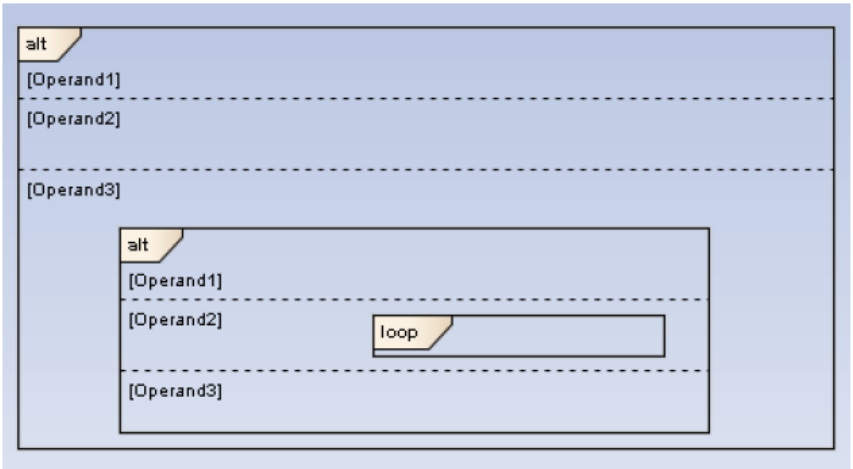
Obr. 55 Metamodel diagramu aktív

Pri vnáraní sa jednotlivých fragmentov do seba využijeme návrhový vzor Composite. Composite funguje tak, že máme 2 typy uzlov (3, ale reálne 2 ktoré dedia od spoločného nadtypu) a to listový uzol (leaf) a composite uzol, čo je to isté, ako listový uzol, ibaže môže mať ďalších potomkov (v grafovej reprezentácií). Keďže môže mať ďalších potomkov, tak obsahuje aj funkcie na vrátenie zoznamu potomkov, vymazanie konkrétneho potomka a pridanie potomka.



Obr. 56 Využitie návrhového vzoru Composite

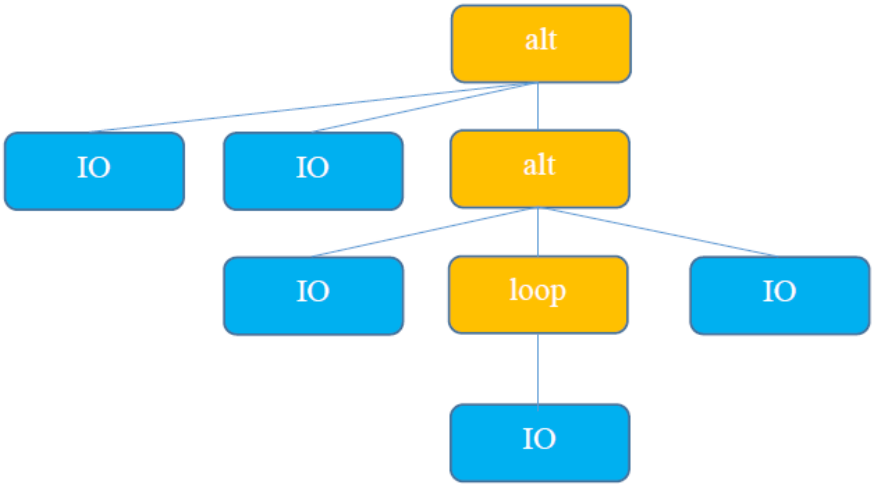
V našom prípade bude nadtriedou, od ktorej budeme dediť (Componentom) modifikovaná trieda InteractionFragment, teda najvšeobecnejšie zoskupenie akcií bez špecifických podmienok na vykonanie. Každý InteractionFragment je spojený s triedou Lifeline, ktorá špecifikuje poradie v akom sú vykonávané jednotlivé entity, ktoré do InteractionFragmenta spadajú. Funkcia execute pre vykonávanie v IF iba spustí vykonávanie a tok do náležitých InteractionOperandov, ktorých súčasťou bude IF (resp. spustí tok danej Lifeline). Compositom bude trieda CombinedFragment. V tomto prípade bude funkcia execute prekonávaná podľa toho, aký interactionOperator bude pridaný za parameter. Loop sa bude napríklad vykonávať prirodzene inak ako Alt. Leaf-om, resp. koncovým uzlom bude trieda InteractionOperand. Tu si treba uvedomiť podstatnú vec a to, že ak si užívateľ zvolí, že nakreslí napríklad fragment alt, systém nemôže vedieť, či je to už fragment finálny a v grafe ho môže reprezentovať ako leaf, alebo, či sa do samotného fragmentu (InteractionOperandu) nepokúsi užívateľ vnoriť ďalší fragment. Pridávanie nových leafov, cez Composite náležiaci danej úrovne zabezpečuje sekvenciu po sebe idúcich fragmentov, nezabezpečuje však vnáranie fragmentov. Pre vnáranie sa fragmentov do seba, by mal každý z leaf-ov mať možnosť transformovať sa dodatočne na Composite (metamodelovo je to zabezpečené, pretože InteractionOperand môže obsahovať ďalší CombinedFragment).



Obr. 57 Fragment Alt

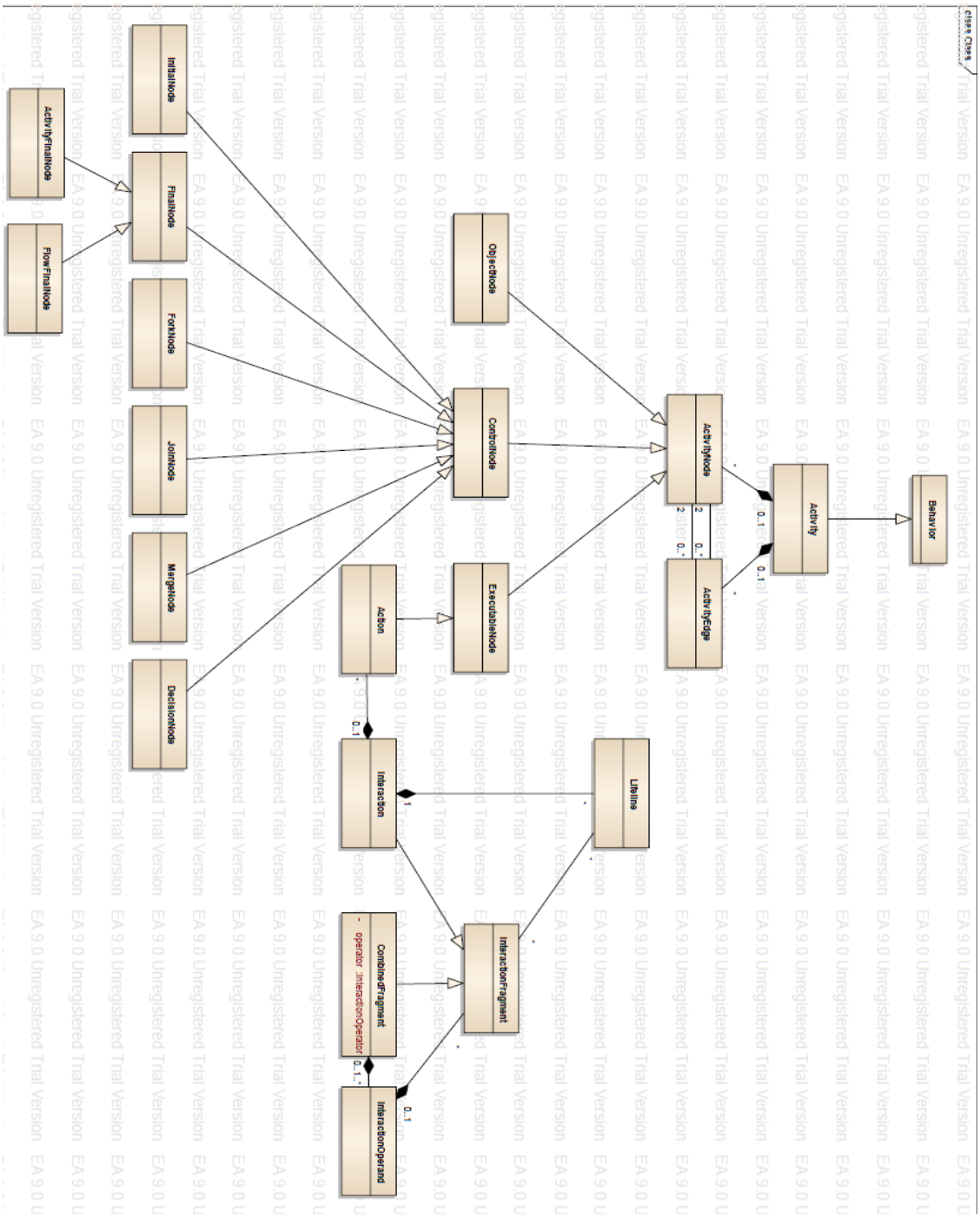
Diagram na hornom obrázku je reprezentovaný grafovou štruktúrou dole. Pri samotnom vykonávaní toku, bude program postupovať tak, že zanalyzuje objekt Composite na najvyššej

úrovni (alt) a na základe vstupných parametrov a zvoleného typu execute (ide o alt nie napríklad loop) začne vykonávať akcie v príslušných operandoch. Ak podľa parametra aktivujeme Operand3 tak sa rekurzívne zopakuje proces vnárania. Až po tom, čo bude do vykonávaný posledný zo zvolených fragmentov, bude odoslaný návratový impulz kompozitu o level vyššie etc...Teda inak povedané horný alt sa do vykonáva, až keď dostane správu o do vykonaní vnoreného (teda všetkých vnorených) fragmentov.



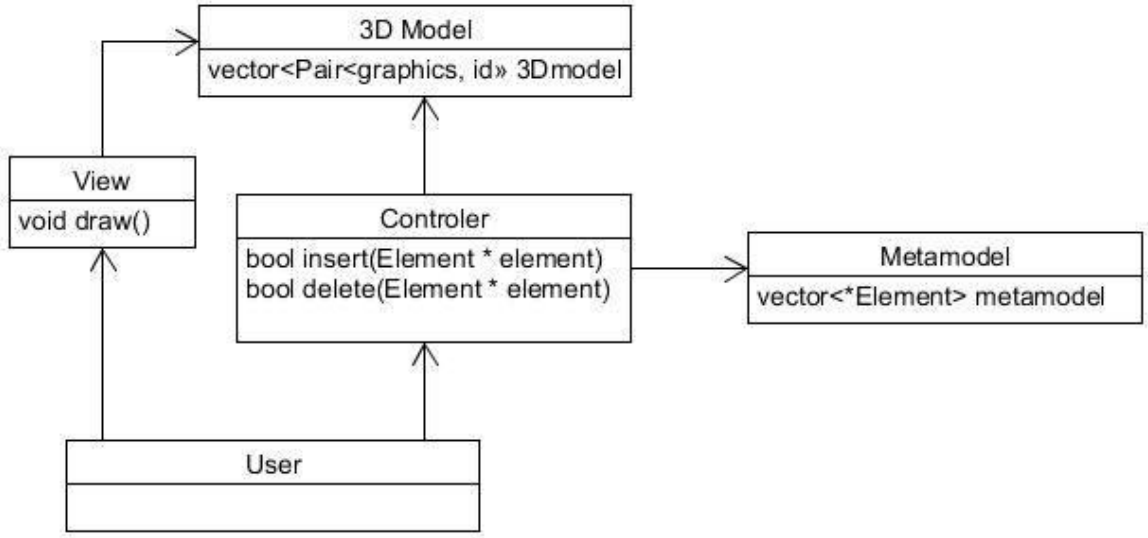
Obr. 58 Grafová interpretácia vnorených fragmentov

Nižšie uvedený model kompletne popisuje základný model prepojenia fragmentov zo sekvenčného diagramu na akcie diagramu aktivít.



Obr. 59 Prepojenie fragmentov zo sekvenčného diagramu na akcie diagramu aktív

5.2. Architektúra systému



Obr. 60 Zvolená architektúra systému

Pre náš systém sme zvolili architektúru MVC – Model View Controller rozšírenú na 2 oddelené modely, ktoré riadi a obsluhuje jeden spoločný kontroler. Našu architektúru môžeme teda nazvať Metamodel-3D Model-View-Controller.

Popis jednotlivých komponentov:

- Metamodel – Obsahuje jednotlivé elementy namodelovaného diagramu s konkrétnymi vzťahmi.
- 3D Model – Obsahuje informácie o koordinátoch elementu v 3D priestore napárovaného na Metamodel.
- Conroller – Reaguje na udalosti, ktoré prichádzajú od používateľa a zaisťuje obsluhu dát v oboch modeloch.
- View – Zobrazuje vytvorené dáta, v našom prípade diagram aktív v 3D priestore.

5.3. Návrh algoritmov

Táto kapitola obsahuje návrh algoritmov, ktoré sú použite v prototyp 3D UML. Väčšina algoritmov sa týka grafického rozhrania a zahŕňajú algoritmy na vykreslenie čiary, alebo nájdenie bodov spájania.

5.3.1. Algoritmus pre nájdenie bodov spájania Merge a Decision bloku

Algoritmus hľadá body spájania, ktoré sú na hrane Decision a Merge bloku. Keďže oba tieto bloky majú rovnakú notáciu, algoritmu je rovnaký. Kosoštvorec, ktorý predstavuje používanú notáciu týchto blokov má štyri vrcholy. Preto existujú štyri body spájania, ktoré sa nachádzajú na vrcholoch kosoštvorca. Keďže objekt je zložený zo štyroch ohraničujúcich čiar, je možné súradnice bodov spájania zistiť prostredníctvom začiatočného a koncového bodu dvoch súbežných z nich

Pseudokód

```
LIST points
poins.add(POINT(element.line1.startPoint.x,  element.line1.startPoint.y))
poins.add(POINT(element.line1.endPoint.x,  element.line1.endPoint.y))
poins.add(POINT(element.line3.startPoint.x,  element.line3.startPoint.y))
poins.add(POINT(element.line3.endPoint.x,  element.line3.endPoint.y))
RETURN points
```

5.3.2. Algoritmus pre nájdenie bodov spájania Join a Decision bloku

Algoritmus hľadá body spájania, ktoré sú na hrane Join a Fork bloku. Keďže oba tieto bloky majú rovnakú notáciu, algoritmu je rovnaký. Obdĺžnik, ktorý predstavuje štandardnú notáciu týchto blokov bude obsahovať na oboch svojich dlhších stranách po jednom bode, pričom jedna strana bude predstavovať vstup a druhý výstup. Súradnice možno získať tak, že dlhšie strany budú rozdelené na polovicu, pričom súradnice tejto polovice budú predstavovať súradnice tohto bodu. Pseudokód uvažuje prípad, kedy je blok orientovaný na výšku. A *line1* a *line3* sú dlhšie hrany bloku.

Pseudokód

```
LIST points
INTEGER y := (element.line1.startPoint.x + element.line1.endPoint.x) / 2.0
poins.add(POINT(element.line1.startPoint.x,  y))
poins.add(POINT(element.line3. startPoint.x,  y))
RETURN points
```

6. Používateľská príručka

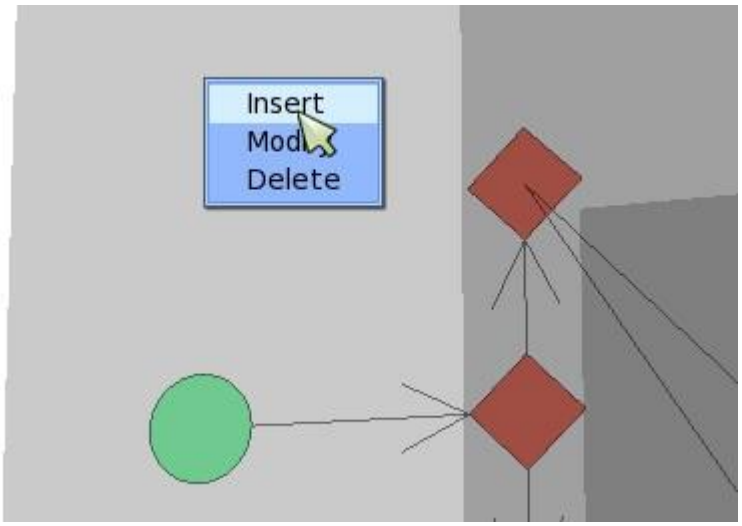
6.1. Použitie diagram aktivít v prototype

Vyberieme tlačidlo, ktorým je možné pridať novú vrstvu do diagramu. Je to z toho dôvodu, že v diagrame aktivít sa využíva iný princíp ovládania, ktorého efektivitu sme chceli otestovať. Tento princíp sa viac zameriava na efektívne využitie myši. Vďaka tomu je používateľovi umožnené pravým tlačidlo vyvolať vyskakovacie menu. Okrem toho je možné v tomto režime vyvolať odsunutie fragmentov kliknutím na ich zvýraznenú časť.

6.1.1. Vkladanie pomocou pravého kliknutia

V snahe zefektívniť vytváranie elementov sme vytvorili ďalší spôsob vkladania nových prvkov. Týmto spôsobom je zobrazenie vyskakovacieho okna po pravým kliknutím myšou. V tomto okne sa zobrazia možnosti pre manipuláciu s elementami. Momentálne funguje len vloženie novej aktivity. Ostatné možnosti slúžia len na ilustráciu.

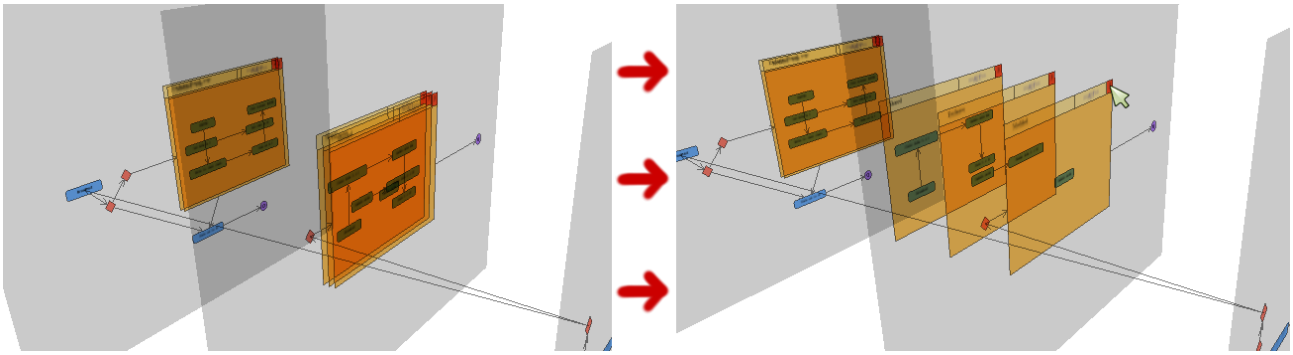
Do menu je možné pridávať ďalšie rozbaľovacie možnosti spôsobom, aký poznáme z bežných aplikácií. Ako takéto rozbaľovacie okno vyzerá je možné vidieť na obrázku nižšie (viď obr.4).



Obr. 61 Vyskakovacie okno pre vkladanie novej aktivity [1]

6.1.2. Animácia fragmentov

Vďaka animáciám je možné rozbaliť do priestoru niekoľko fragmentov, ktoré sú pokope. Toto zvyšuje ich prehľadnosť. V momente keď sú vrstvy animované, odsúvajú sa vrstvy nachádzajúce sa za nimi, spolu s elementami, ktoré sa na nich nachádzajú. Na obrázku nižšie (viď. obr. 5) je možné vidieť, ako vyzerajú elementy pred odsunutím fragmentov a následne, ako vyzerajú po kliknutí na roh fragmentu a odsunutí fragmentov dozadu.



Obr. 62 Odsunutie fragmentu po kliknutí na jeho roh [1]

Pre prehľadnosť sa v rohu fragmentu zobrazuje číslo. Toto číslo určuje počet fragmentov, ktoré sú pokope.

7. Použité zdroje

[1] Ing. Matej Škoda, *Trojdimenzionálne zobrazenie UML diagramov [diplomová práca]*, Slovenská Technická Univerzita v Bratislave, 2014.

[2] OMG Unified Modeling Language TM (OMG UML) Version 2.5.
`http://www.uml.org/`
[Navštívené 13.10.2014]

[3] Combined Fragment
`http://www.uml-diagrams.org/sequence-diagrams-combined-fragment.html`
[Navštívené 20.10.2014]