Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Ilkovičova 2, 842 16 Bratislava 4

Trojdimenzionálne UML

Dokumentácia k inžinierskemu dielu

Tím č. 2

Členovia tímu: Bc. Brndiarová Gabriela, Bc. Kyseľ Peter, Bc. Martoš Ivan,
Bc. Štajer Andrej, Bc. Štetiar Matej, Bc. Šuta Erik,
Bc. Valko Andrej, Bc. Kolačkovský Tomáš (do 6. týždňa ZS)

Študijný program: IS/SI

Ročník: 1. Ing.

Predmet: Tímový projekt

Vedúci tímu: Ing. Polášek Ivan, PhD.

Ak. rok: 2013/14

Obsah

1 Úvod

Tento dokument predstavuje dokumentáciu k inžinierskemu dielu tímu číslo 2: *GAMATEPI* – trojdimenzionálne UML. Obsahom tohto dokumentu je celkový pohľad na produkt vytváraný v rámci predmetu Tímový Projekt v akademickom roku 2013/2014. V kapirole č. 1 – Úvod je opísaný celkový pohľad na projekt, jeho ciele a základné implementačné informácie. V nasledujúcich kapitolách je opísaný postup prác na projekte vo forme šprintov, kde prebehla implementácia jednotlivých používateľských scenárov. V kapitole č. 8 – Celkový pohľad na produkt je opísaný celkový pohľad na projekt v kontexte softvérového inžinierstva. Kapitola sa venuje architektúre produktu a dátovému modelu. V prílohách možno nájsť používateľskú príručku a preberacie protokoly.

1.1 Celkový pohľad na projekt

UML, ako dôležitý nástroj softvérového inžiniera, slúži na návrh, špecifikáciu, modelovanie a dokumentovanie softvéru. Jednou z jeho úloh je ponúknuť grafickú alternatívu opisu softvéru. Aktuálne sa väčšina grafických nástrojov pre vizualizáciu UML zameriava len na 2D priestor. Našou snahou je vytvoriť prototyp, ktorý bude vizualizovať UML diagramy v 3D priestore.

3D zobrazenie napomáha k rýchlejšej orientácia v spleti komplikovaných vzťahov. Pridaním ďalšej dimenzie dokážeme prehľadnosť diagramu ešte zvýšiť, či zvýrazniť niektoré zaujímavé časti. Otvárajú sa tu taktiež možnosti rozšírenej vizualizácie jednotlivých častí diagramov, prípadne interakcií medzi diagramami.

Na začiatku projektu bol dodaný prototyp, ktorý obsahoval implementovanú funkcionalitu pre vytvorenie jednoduchého diagramu tried. Tento prototyp plánujeme naďalej rozvíjať. Prototyp a aj naše riešenie je zamerané na osobné počítače, avšak nie je vylúčená možnosť rozšírenia na ďalšie druhy výpočtových zariadení.

1.2 Ciele projektu

Cieľom projektu je do dodaného prototypu implementovať časti funkčnosti sekvenčného diagramu. V prvom semestri sa budeme zaoberať problematikami spustenia prototypu, vytvorenia čiary života, interakcie medzi čiarami života a základnej reprezentácie fragmentu. Samozrejmosťou je možnosť pomenovania jednotlivých prvkov diagramu, pričom nie nutne splniteľná súčasť plánu je taktiež vytvorenie vnútorného UML modelu na základe ktorého by

bolo možné vytvárať jednotlivé prvky diagramu definované v modeli. Okrem týchto funkcií spojených s diagramom je cieľom implementovať a dodať aj všeobecnú funkčnosť vytvorenia viacerých vrstiev v rôznych hĺbkach pre trojdimenzionálnosť produktu a taktiež aj serializácia vytvorených diagramov vo forme XML súborov. Cieľom je taktiež funkčnosť načítania týchto súborov a znovu vytvorenie uloženej scény.

1.3 Implementácia

Projekt nadväzuje na existujúci prototyp, ktorý je implementovaný v jazyku C++. Vo veľkom rozsahu je využívaná externá knižnica OGRE (Open Source 3D Graphic Engine). Grafické používateľské rozhranie podporuje externá knižnica MyGUI. Na kompiláciu je používaný MinGW kompilátor.

2 Ciele projektu – zimný semester

Cieľom celého projektu je vytvorenie funkčného prototypu, ktorý bude obsahovať funkčnosť vytvárania, prezerania a upravovania 3D UML sekvenčných diagramov. Táto funkčnosť bude implementovaná do existujúceho prototypu, ktorý sa ale zameriava na prácu s 3D diagramom tried. Pri ukončení práce na projekte je predpoklad zlúčenia týchto prototypov do jedného celistvého a funkčného prototypu, ktorý bude obsahovať funkčnosť oboch momentálne samostatne vyvíjaných častí prototypu.

Tretí rozmer riešenia je dosiahnutý vytvorením viacerých vrstiev, ktoré budú uložené v priestore v odlišnej hĺbke. Každá vrstva môže obsahovať celý diagram, prípadne len jeho časť. Týmto spôsobom je možné jednotlivé diagramy a ich časti jednoduchšie vizualizovať. Bude taktiež umožnená interakcia medzi vrstvami, takže samotný diagram sa nebude musieť nutne nachádzať len na jednej vrstve, ale pre zlepšenie prehľadnosti je možné diagram rozložiť do hĺbky.

Funkčnosť vytváraná v prototype musí obsahovať okrem štandardného vytvorenia, modifikácie a prehliadania diagramov aj ostatné používateľovi viditeľné, ale aj skryté funkčnosti. Vnútri diagramu je predpoklad ku možnej implementácií dátového UML metamodelu, ktorý bude ukladať prvky v diagrame a na základe neho bude môcť vytvárať nové prvky. Jedná sa napríklad o triedy a ich inštancie. Pri pomenúvaní prvkov bude ale vždy potrebné, aby používateľ vykonal vstup na základe ktorého je element pomenovaný.

Taktiež je dôležité, aby sa jednotlivé prvky diagramu navzájom na jednej vrstve neprekrývali a taktiež aby boli usporiadané podľa konvencií UML, pričom stále je dôležité, aby bol diagram prehľadný a nebol problém sa v ňom orientovať.

V prototype bude taktiež vytvorená možnosť uloženia diagramu a aj jeho načítanie z uloženého súboru.

3 Šprint č. 1

3.1 Vytvorenie prezentačnej stránky

Za účelom prezentácie projektu sme vytvorili internetovú stránku. Stránku je možné si pozrieť na adrese http://labss2.fiit.stuba.sk/TeamProject/2013/team02is-si/ a jej úvodná stránka sa nachádza na obrázku pod textom (Obr. 3.1). Počas celého priebehu projektu bude aktualizovaná o nové informácie a bude odzrkadľovať priebeh samotného projektu. Základnými časťami stránky sú:

- Tím – Obsahuje fotografie členov tímu. Po kliknutí na jednotlivé fotografie stránka ponúkne stručné predstavenie dotyčnej osoby s jednoduchou možnosťou návratu k prehliadaniu stránky kliknutím na čierne pole stránky.
- O projekte – Informácie ohľadom rozdelenia úloh v tíme ako aj základný opis nášho projektu.
- Galéria – Obrazová galéria realizácie projektu.
- Dokumenty – Nachádzajú sa tu dokumenty k projektu (t.j. zápisnice zo stretnutí, dokumentáciu riadenia a dokumentáciu k inžinierskemu dielu

Obr. 3.1: Úvod prezentačnej stránky

Pri implementácii prezentačnej stránky boli použité technológie HTML5, CSS3 a Javascript.

3.2 Analýza problémovej oblasti

Sekvenčný diagram je diagramom správania jazyka UML. Jeho využitie je hlavne v modelovaní scenárov prípadov použitia navrhovaného systému. V porovnaní s diagramom interakcií je výhodnejší pri malom počte komunikujúcich objektov a vyššom počte metód, cez ktoré komunikujú. Najvýraznejšou črtou tohto diagramu je zobrazenie času. Je na to využitá vertikálna os. Udalosti za sebou nasledujú zhora - nadol.

Medzi artefakty sekvenčného diagramu patrí čiara života, symbolizujúca aktéra scenára (interagujúci objekt); správy, zachytávajúce metódy a volania funkcii, a fragmenty. Fragmenty modelujú špeciálne situácie, vetvenia, cyklenia, paralelného spracovania a iných.

3.3 Spustenie existujúceho prototypu

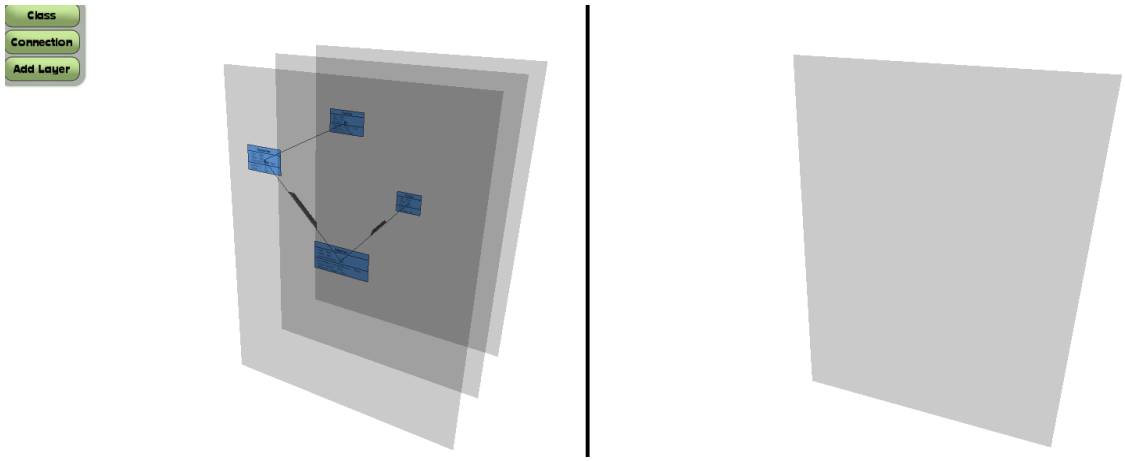
Náš projekt sa opiera o už existujúce riešenie. To znamená, že každý člen tímu musel dokázať spustiť tento prototyp u seba lokálne. Ako bolo spomínané v kapitole 1.3 systém je založený na rozsiahlych externých knižniciach a využíva špecifický druh kompilátora, preto sa táto úloha ukázala ako netriviálna.

3.4 Príprava aplikácie

Pred začiatkom nášho vývoja bolo nutné zabezpečiť zachovanie možnosti pracovať s diagramom tried. Za týmto účelom bola v hlavnej triede zdrojového kódu vytvorená statická premenná, ktorá nadobúda hodnoty pravda/nepravda. Na základe tejto hodnoty sa na začiatku behu program určí, či bude používateľ pracovať s diagramom tried (hodnota “pravda”) alebo sekvenčným diagramom (hodnota “nepravda”).

V tomto štádiu nám toto riešenie prinieslo spôsob ako sa pri implementácii sústrediť na našu úlohu. Do budúcna bude možné takýto prepínač ponúknuť aj v rámci používateľského rozhrania, bez nutnosti veľkého zásahu do architektúry systému.

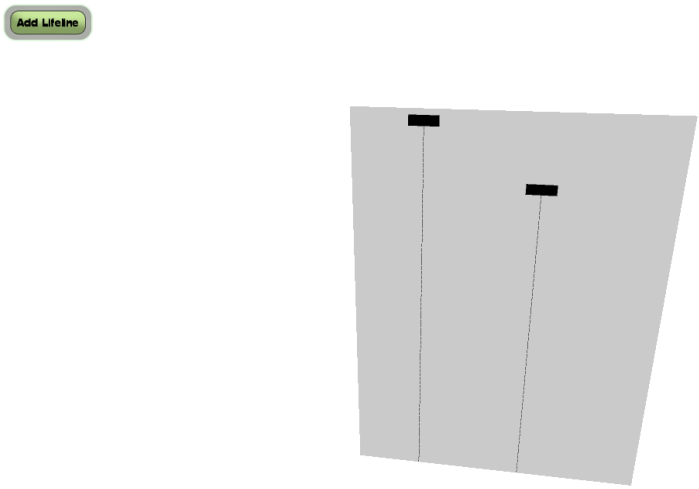
Pôvodný prototyp po spustení aplikácie používateľovi ponúkol na obrazovke 3 vrstvy so vzorovými triedami a vzťahmi medzi nimi. Príprava aplikácie na našu prácu preto zahŕňala aj odstránenie prebytočných prvkov scény. Výsledok tejto práce zobrazuje Obr. 3.2.



Obr. 3.2: Vľavo je stav po spustení diagramu tried. V pravo po spustení sekvenčného diagramu

3.5 Pridať čiaru života

V aplikácii bola implementovaná nová funkcionalita. Používateľ môže pridať novú čiaru života na vrstvu. Jej grafická reprezentácia pozostáva z modrého obdĺžnika, z ktorého vychádza prerušovaná čiara (viď Obr. 3.3).

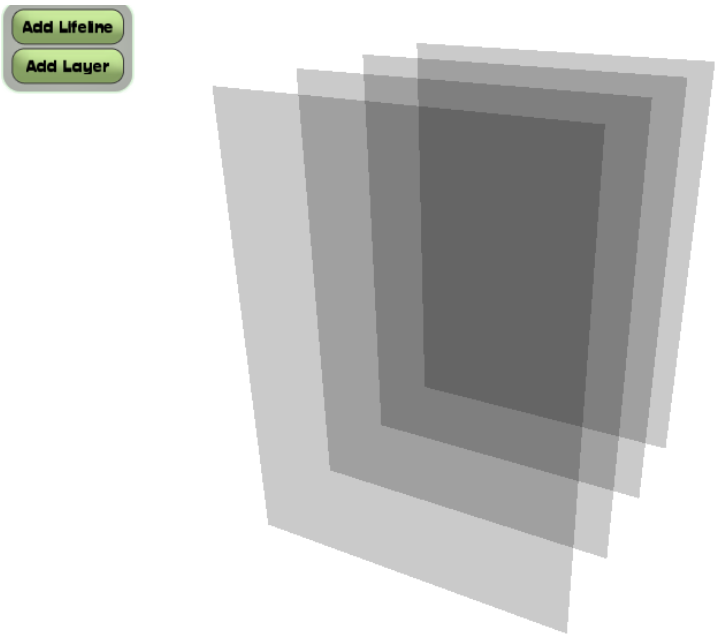


Obr. 3.3: Obrazovka aplikácie po pridaní dvoch čiar života

4 Šprint č. 2

4.1 Pridať novú vrstvu

Trojrozmernosť sekvenčného diagramu v aplikácii spočíva v jeho rozdelení na niekoľko vrstiev v priestore. Preto používateľ potrebuje možnosť pridávať nové vrstvy na obrazovku. V tomto prípade však nešlo o implementáciu novej funkcionality. Pridávanie novej vrstvy bolo už implementované v časti s diagramom tried. Práca preto spočívala v jej sprístupnení aj pre sekvenčný diagram. Výsledok po pridaní troch nových vrstiev zobrazuje Obr. 4.1.



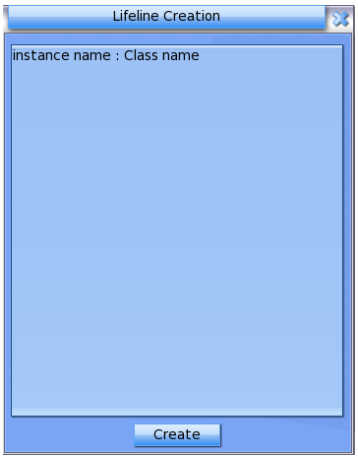
Obr. 4.1: K jednej vrstve, vytvorenej pri spúšťaní aplikácie, sme pridali 3 nové vrstvy

4.2 Zadať názov inštancie a názov triedy pre čiaru života

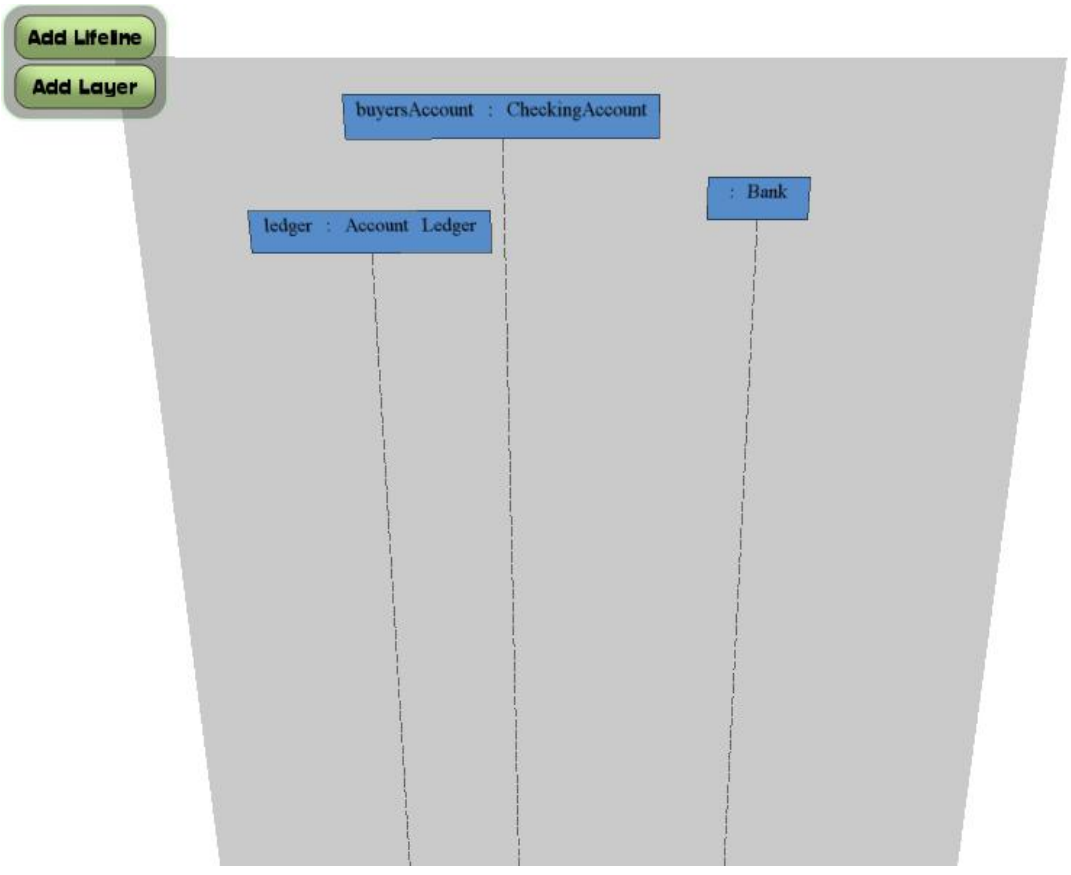
Čiara života v sekvenčnom diagrame predstavuje konkrétnu inštanciu triedy (v špeciálnych prípadoch sa uvádza iba názov triedy). Tieto dva názvy sú od seba oddelené znakom „:“ (dvojbodka) a zobrazujú sa v obdĺžniku čiary života.

Na základe tejto UML notifikácie bolo implementované zadávanie a zobrazovanie názvu. Používateľ určuje názov inštancie a triedy počas pridávanie novej čiary života. Obr. 4.2 zobrazuje

formulár na zadávanie textu aj so vzorkou, a na obrázku pod ním (Obr. 4.3) je výsledné zobrazenie.



Obr. 4.2: na zadanie názvu inštancie a názvu triedy



Obr. 4.3: Stav po pridaní troch čiar života

4.3 Zjednodušenie výberu čiary života

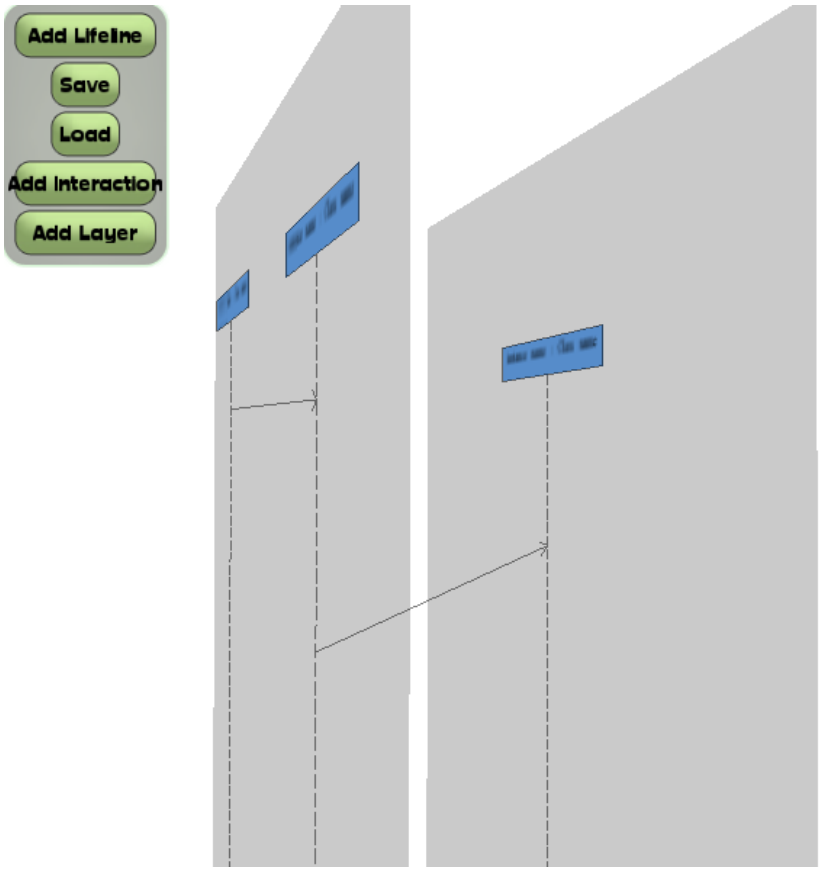
Pre zjednodušenie interakcie používateľa s prototypom bol vytvorený všeobecný obal pre grafické prvky. Obal je reprezentovaný neviditeľným obdĺžnikom, ktorý pokrýva celú plochu grafického prvku. Hlavnou úlohou obalu je zjednodušenie výberu grafického prvku používateľovi a to tak, že je postačujúce kliknúť na obal prvku. Toto môže zjednodušiť napríklad pridávanie interakcie, kedy používateľ nemusí kliknúť priamo na zvislú prerušovanú čiaru, ale stačí aby klikol do jej okolia a to v rámci šírky hlavičky čiary života.

Obal prvku, ktorý sme vytvorili, taktiež napomáha pri usporadúvaní prvkov v priestore tak, aby sa neprekrývali. Toto správanie však nie je implicitne zabezpečované obalom prvku. Jedná sa len o podporný prostriedok, ktorý si vyžaduje špecifickú implementáciu, pre každý jeden grafický prvok.

4.4 Pridať interakciu medzi objektmi

Spolupráca medzi objektmi sekvenčného diagramu je určená interakciami. Ako prvá interakcia bola implementovaná asynchrónna správa s tým, že vždy má svoj začiatok a koniec.

Ako možno vidieť na Obr. 4.4 interakcia medzi objektmi nie je obmedzená na objekty rovnakej vrstvy. Aktuálne nie sú implementované ohraničenia toho, ako pridávať interakciu, čo sa týka smeru a orientácie.



Obr. 4.4: Interakcie

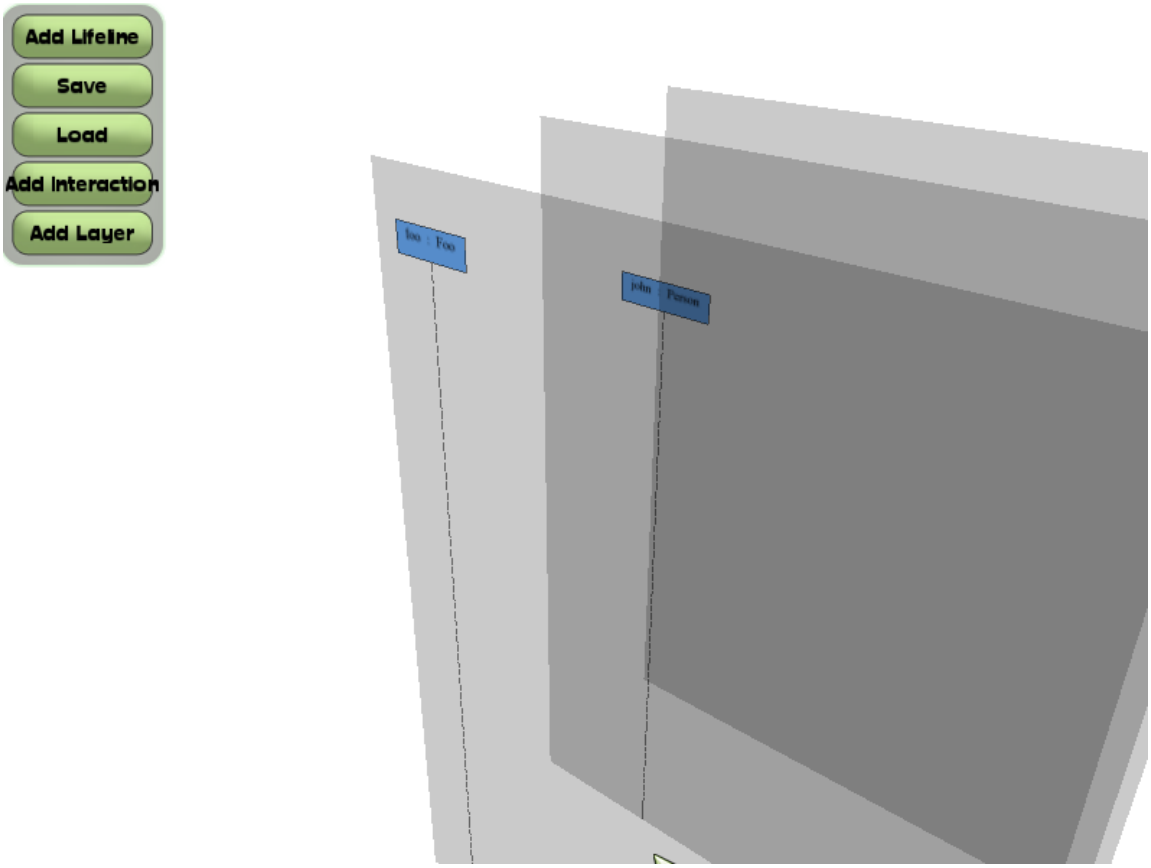
4.5 Zarovnanie čiar života

Ďalším cieľom šprintu bolo zarovnanie čiar života pri ich vytvorení tak, aby sa neprekrývali a diagram bol teda prehľadný. Pri implementácií sa zistili určité problémy návrhu aplikácie, ktoré by boli ale príliš časovo náročné na opravu. Taktiež bolo navrhnuté doimplementovanie určitej funkcionality. Zarovnanie bolo vyriešené. Pri pridaní novej čiary života, sa čiara života pridá na príslušné miesto pokiaľ je tam dostatok voľného priestoru. Pokiaľ dostatok priestoru nie je, nič sa nepridá. Je to z toho dôvodu, že to by sa už jednalo o modifikáciu existujúcich čiar života, čo nie je cieľom tejto úlohy.

4.6 Uložiť a načítať diagram

Jedným z cieľov šprintu číslo 2 bola aj implementácia funkcionality na ukladanie a spätné načítavanie používateľom vytvoreného sekvenčného diagramu. Prvotný pokus o implementáciu sa opieral o princípy serializácie. Z technických príčin však vyplynulo, že ísť cestou serializácie

by si vyžadovalo zmeny v prototype v neprípustnej miere. Ako ďalšiu alternatívu sme zvolili jednoduché ukladanie dôležitých dát elementov scény, ktoré sú potrebné na spätnú rekonštrukciu diagramu v textovom súbore. Implementovaná bola základná kostra ukladania. Ďalšie práce a ukončenie ukladania a načítavania sekvenčného diagramu budú vykonané v nasledujúcom šprinte. Nižšie (Obr. 4.5) možno vidieť súčasný stav tejto funkcionality.



Obr. 4.5: Zobrazenie tlačidiel na ukladanie a načítanie sekvenčného diagramu

4.7 Oprava chýb

Po prvom šprinte obsahoval prototyp 2 zobrazovacie chyby. Obe sa nám podarilo v rámci šprintu č.2 opraviť.

Tieň v obdĺžniku čiary života

Pri prezeraní diagramu z blízka používateľ videl modrú textúru zobrazovaného obdĺžnika (reprezentujúci začiatok čiary života). Po oddialení v obdĺžniku vznikal tieň, ktorý spôsoboval esteticky nepriaznivý efekt.

Túto chybu sa nám podarilo opraviť v rámci implementácie zadávania textu do obdĺžnika (kapitola 4.2). Do obdĺžnika bol vložené textové pole, v ktorom sa tieň nevytvára.

Prerušovaná čiara presahuje vrstvu

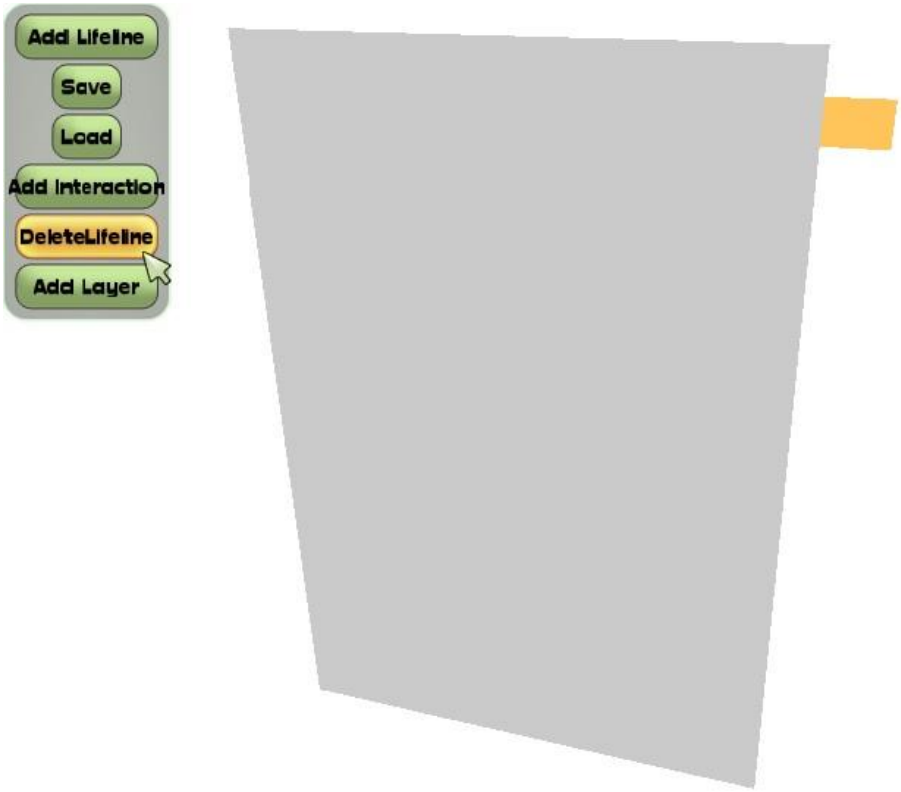
Pri vytváraní čiary života dochádzalo ku chybe spôsobujúcej presahovanie čiary života cez spodnú časť vrstvy pre vykresľovanie. Chyba bola spôsobená fixnou dĺžkou čiary života.

Táto chyba bola opravená v rámci samostatnej úlohy, kedy bolo potrebné implementovať variabilnú dĺžku čiary života. Táto dĺžka sa vypočítava na základe pozície čiary života na vrstve a celkovej dĺžky vrstvy.

5 Šprint č. 3

5.1 Vymazávanie objektov

Diagramy nie sú len o pridávaní nových prvkov, ale aj o ich odoberaní. V aplikácií pribudla nová funkcionalita. V menu vľavo pribudlo nové tlačidlo DeleteLifeline (Obr. 5.1). Používateľ si vyberie vrstvu, na ktorej sa čiara života nachádza pomocou oranžového obdĺžnika v pravom hornom rohu vrstvy. Po výbere vrstvy sa ostatné vrstvy skryjú. Nakoniec používateľ vyberie čiaru života, ktorú chce vymazať z diagramu. Funkcionalita vymazania čiary života zatiaľ nie je implementovaná. Pokračuje sa na nej v nasledujúcom šprinte.



Obr. 5.1: Zobrazenie tlačidla DeleteLifeline

5.2 Maximálna šírka čiary života

Pre možnosti zarovnania čiar života pri ich vytvorení je potrebné, aby bolo známe, aká môže byť maximálna šírka novovytvorenej čiary života. Táto maximálna šírka je daná konštantou vnútri

zdrojového kódu, pričom v budúcnosti je možné ju nastaviť ako súčasť konfiguračného súboru. Od tejto maximálnej šírky sa taktiež odvíja aj zalamovanie textu v čiare života.

Funkčnosť maximálnej šírky a prislúchajúcich obmedzení pri vykreslení bola úspešne implementovaná a otestovaná.

5.3 Pridanie atribútov pre prácu so šírkou a výškou elementov

Pre prácu s prvkami v grafickom prostredí bolo potrebné vytvoriť funkcionalitu ktorá umožní získanie aktuálnej šírky týchto elementov. Keďže elementy sú vykresľované tak, že sú zadefinované hraničné body polygónov a medzi týmito bodmi sú následne vytvorené čiary ktoré zobrazujú samotný element, je možné získať aktuálnu šírku, ale aj hĺbku a výšku z týchto hraničných bodov.

Výpočet je vytvorený veľmi jednoducho. Knižnica OGRE ponúka možnosť získania maximálneho a minimálneho vektora, ktorý obsahuje maximálne (minimálne) hodnoty ôs x, y a z, ktoré definujú hraničné body elementu. Na základe týchto vektorov bol vykonaný rozdiel maximálnej a minimálnej hodnoty pre danú os, pričom výsledok definuje aktuálnu šírku, výšku alebo hĺbku elementu.

5.4 Označenie čiar života ťahom myši (pre vloženie fragmentu)

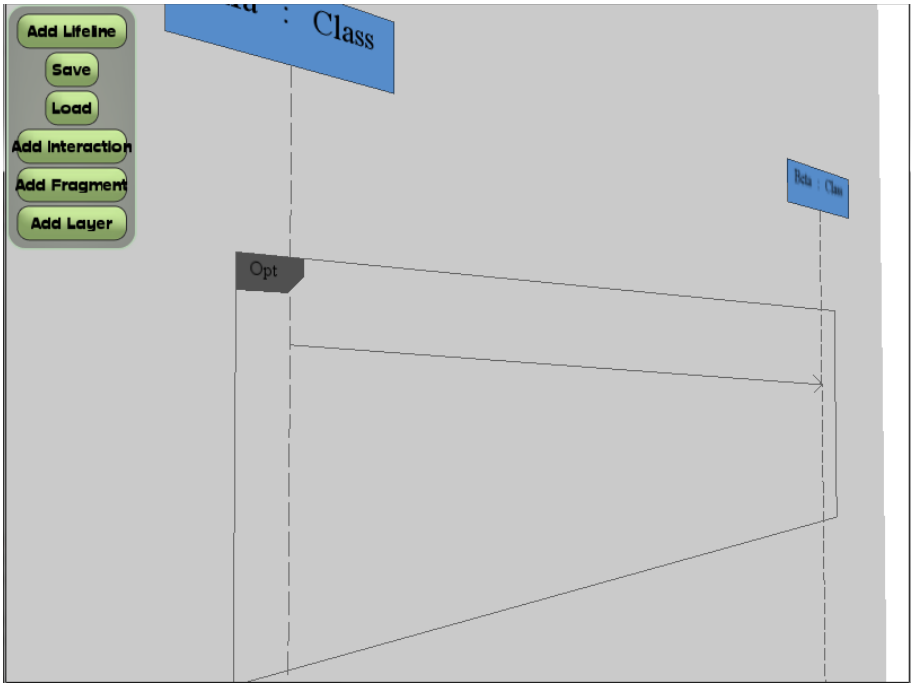
Pri implementácií fragmentu bolo potrebné umožniť používateľovi vybrať si umiestnenie fragmentu. Pre podobnosť so systémami, zaoberajúcimi sa podobnou funkcionalitou, sme sa rozhodli implementovať túto funkčnosť ako ťah. Ťah sme si definovali, ako:

1. Stlačenie tlačidla myši na mieste, kde chceme aby fragment začal.
2. Držanie stlačeného tlačidla.
3. Presunutie sa na miesto, kde by mal fragment končiť.
4. Pustenie tlačidla myši.

Na začiatku sme si mysleli, že bude vhodné začať aj skončiť ťah na čiare života. Táto myšlienka sa však pri testovaní ukázala ako nepraktická. Preto sme sa rozhodli, že túto logiku upravíme v ďalšom šprinte. Nová logika sa bude opierať o body, na ktoré používateľ klikol a čiary života obsiahnuté medzi nimi.

5.5 Návrh fragmentu

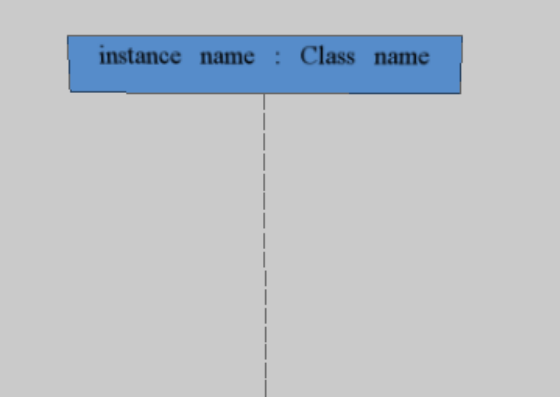
Bol navrhnutý jednoduchý fragment (Obr. 5.2), podľa neho budú vytvorené ostatné typy fragmentov. Aby bolo možné v 3D diagrame vytvoriť fragment, bolo nutné implementovať spôsob výberu čiar života (*lifelines*) Tento spôsob je opísaný vyššie. Pri vytváraní má používateľ zatiaľ možnosť určiť text vyjadrujúci typ fragmentu. Na vykreslenie bol vytvorený kresliaci algoritmus, ktorého výsledok znázorňuje (Obr. 5.2). Algoritmus vykresľuje ľavú a pravú hranu fragmentu tak aby bola zarovnaná na určenú vzdialenosť od príslušnej čiary života (*lifeline*).



Obr. 5.2: Prvotný návrh a vykreslenie fragmentu

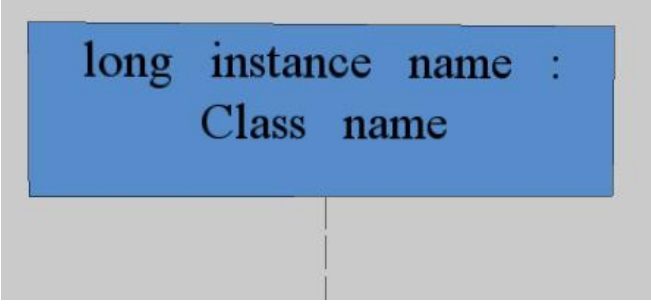
5.6 Zalamovanie textu v čiarach života

Pri implementáciu maximálnej šírky čiary života, je potrebné, aby text v jej názve zostal čitateľný, a teda aby sa prispôsobil tejto maximálnej šírke. Pri vytváraní textu môžu nastať dve možnosti. Pokiaľ je text kratší ako maximálna šírka lifeline, tak je možné ho vykresliť bez zmeny:



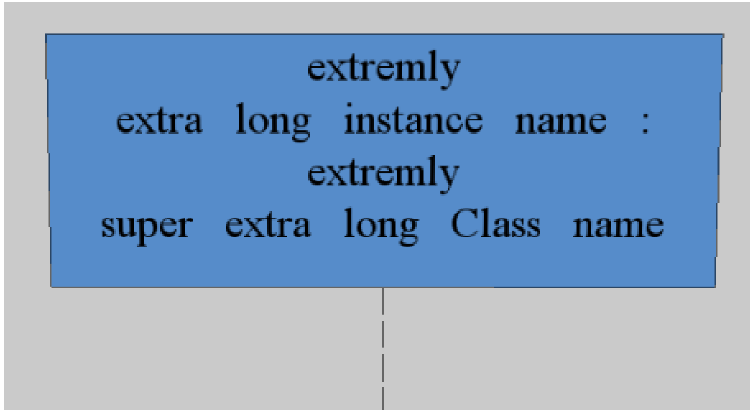
Obr. 5.3: Nie je potrebné modifikovať text

Ďalšou možnosť ale je, že šírka textu na jeden riadok bude väčšia ako maximálna šírka čiary života. Prvým krokom, ktorým je možné riešiť tento problém, je rozdeliť meno inštancie a meno triedy na samostatný riadok:



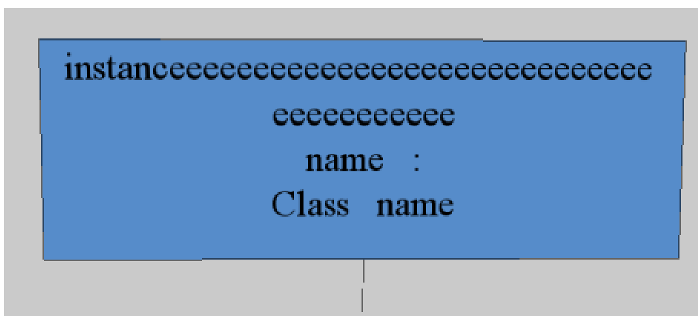
Obr. 5.4: Vyriešenie problému rozdelením mena inštancie a triedy na samostatné riadky

Stále môže ale nastať prípad, kde samostatný názov inštancie alebo riadku je dlhší ako maximálna možná šírka čiary života. V danom prípade sa rozdelí dlhšia časť názvu na riadky tým, že medzera v názve sa zamení za nový riadok:



Obr. 5.5: Zarovnanie pomocou rozdelenia názvov na riadky

Avšak stále môže nastať prípad, pri ktorom je samostatné slovo v názve triedy alebo inštancie dlhšie ako maximálna šírka. V tomto prípade je slovo rozdelené na viacero riadkov po dosiahnutí maximálnej šírky:



Obr. 5.6: Zarovnanie príliš dlhého slova

Takto sú všetky možné problémy vykreslenia textu ktoré môžu nastať pri príliš dlhom texte v čiare života vyriešené.

5.7 Zarovnávanie čiar života

V tomto šprinte bola doimplementovaná aj chýbajúca funkcionalita zarovnania čiar života a prislúchajúcich problémov ako zalamovanie textu, maximálna šírka, získanie aktuálnej šírky a okno s chybovou správou.

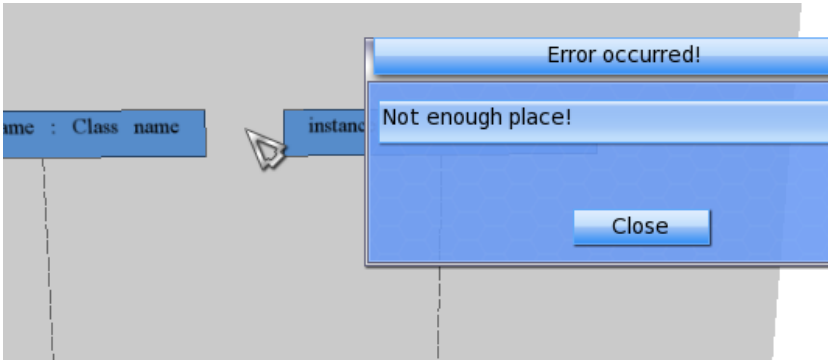
Maximálnu šírku bolo nutné doimplementovať z toho dôvodu, že aktuálny návrh aplikácie získaný od dodaného prototypu vypočítava výšku novo vytvorenej čiary života (ako aj ostatných prvkov) až pri ich vykreslení. Pre problém zarovnávania by „good practice“ bolo najprv

vypočítať veľkosť pridanej čiary života, následne čiaru života posunúť tak, aby bola zarovnaná, a vykresliť, avšak to možné nie je. Keďže v aktuálnom stave projektu nie je implementované upravovanie existujúcich elementov, je nutné zarovnanie čiar života vypočítavať na základe jej predpokladanej maximálnej šírky.

Používateľ pri zadávaní pozície novej čiary života musí špecifikovať jej polohu. Pokiaľ miesto ktoré označí je súčasťou existujúcej čiary života, tak pokiaľ klikol na jej ľavú polku, posunie sa stred novej čiary života na ľavo od nej, pokiaľ na pravú polku, tak na pravo od nej.

Pokiaľ používateľ klikol na aktívnu vrstvu, alebo na existujúcu čiaru života, je potrebné zistiť či v prípade vykreslenia v mieste ktoré označil nenastanú konflikty s inými existujúcimi čiarami života danej vrstvy. Tu sa kontroluje či je naľavo (napravo) od stredu novej čiary života dostatok miesta na vykreslenie. Táto kontrola prebieha na tom princípe, že v jednom pomyselnom stĺpci zadefinovanom existujúcou čiarou života, sa môže nachádzať len práve táto čiara života. Jednoduchšie povedané – čiary života sa nesmú prekrývať a nemôžu existovať jedna nad druhou. Táto kontrola sa vypočítava na základe maximálnej šírky novej čiary života a aktuálnej šírky existujúcich čiar.

Taktiež prebieha kontrola, či stred novej čiary života, ktorý bol posunutý v rámci výpočtu, sa stále nachádza na aktívnej vrstve. Pokiaľ nastane akákoľvek chyba, ktorá neumožňuje pridanie novej čiary života, prípadne pre ňu nie je dostatok miesta, tak je zobrazené chybové hlásenie a proces končí.



Obr. 5.7 Zobrazenie chybového hlásenia v prípade, že používateľ sa pokúša umiestniť novú čiaru života na pozíciu, kde nie je dostatok miesta

Funkcionalita bola implementovaná a otestovaná.

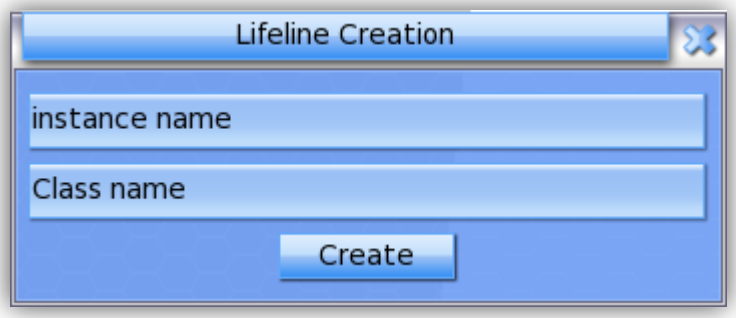
5.8 Ukladanie a načítavanie diagramov

Prvým krokom pri implementácii exportu vykresleného diagramu do XML súboru bola analýza dostupných knižníc zaoberajúcich sa práve prácou s XML v kontexte programovacieho jazyka C++. Po detailnej analýze sme vybrali knižnicu TinyXML. Integrácia do prototypu prebehla bez problémov. Ďalším krokom bola implementácia systému ukladania. Zvolili sme generický prístup, čo znamená, že v budúcnosti bude veľmi jednoduché rozšíriť alebo upraviť exportovacie schémy pre jednotlivé objekty v sekvenčných diagramoch. Úlohou pre ďalší šprint je dokončenie spätnej rekonštrukcie scény s jednotlivými elementmi na základe uloženého XML súboru.

5.9 Oprava chýb

Chybné získavanie údajov z formulára na vytvorenie čiary života

Pri vytvorení čiary života sa používateľovi zobrazí formulár, kde zadáva názov inštancie a triedy. Ak používateľ zadal dvojriadkový názov, tak aplikácia spadla. V rámci riešenia tejto chyby sme inovovali formulár, ktorý už neobsahuje iba jedno pole, ale dve (Obr. 5.8). Obidve polia sú jednoriadkové. Myslíme si, že týmto riešením sme nie len opravili existujúcu chybu, ale aj grafické používateľské rozhranie získalo na intuitívnosti.



Obr. 5.8: Formulár na zadanie atribútov čiary života

Oprava chyby výberu elementu na skrytej vrstve

Pri výbere elementu pomocou jeho obalu sa objavovala chyba, ktorá spôsobovala výber nezobrazeného elementu. Vybraný element sa nachádzal na skrytej vrstve. Pri implementácii bolo pridané overenie či sa nájdený element nachádza na aktívnej vrstve.

Nemožnosť vytvoriť čiaru života z kódu

Implementácia vykreslenia čiary života sa opierala o postupnosť funkcií vykonávanú pri vytváraní používateľom. V špeciálnych prípadoch je však potrebné vytvoriť čiaru života priamo z kódu. Preto bola logika vytvárania upravená tak, aby sa pri vykresľovaní algoritmus opieral o dáta zadané pri vytvorení čiary života – Presnejšie povedané o rodiča zadaného ako argument funkcie pri vytváraní.

6 Šprint č. 4

6.1 Vytvorenie programovej infraštruktúry pre mazanie

Úloha „Vytvorenie stavov a tlačidla“ bola upravená oproti predchádzajúcej verzií. Bolo nutné upraviť implementáciu, aby bola kompatibilná s novou verziou aplikácie. Nová logika umožňuje nielen vymazanie čiary života z diagramu, ale akéhokoľvek elementu. Bola vykonaná refaktorizácia a úpravy podľa pripomienok. Predchádzajúca verzia na zvolenej vrstve vyberala iba čiaru života a ostatné elementy ignorovala. Aktuálny stav, je taký, že je možné vybrať akýkoľvek element z vrstvy.

Spustenie mazania

Bola doplnená funkcionalita pre prepojenie výberu elementu zo scény s jeho vymazaním. Táto funkcionalita je prístupná cez tlačidlo „Delete Element“, pričom je bližšie opísaná v kapitole „Vytvorenie stavov a tlačidla“.

Vymazanie elementu

Bol implementovaný deštruktor, ktorý vymaže element zo zoznamu elementov, obrazovky a takisto vymaže jeho cover.

Mazanie asynchrónnych správ

Boli vytvorené deštruktory pre AsynchronousMessage, Element a Cover. Taktiež bola navrhnutá postupnosť deštruktorov elementov zobrazených na obrazovke. Každá trieda dediaca od Element musí sama zrušiť všetky referencie na seba, prípadne na ktoré sa odkazuje. Následne je zavolaný deštruktor nadtriedy (Element) v ktorom je vymazaný obal (ak nejaký má) a je vymazaná z ElementCollection a zo scény.

V rámci deštruktora asynchrónnej správy boli odstránené referencie na čiary života, ktorých sa týka. Taktiež v rámci tejto úlohy boli vytvorené metódy slúžiace na vymazanie elementu z obrazovky a z ElementCollection.

Napriek počiatočnému zadaniu tejto úlohy – vytvoreniu deštruktora, boli vytvorené metódy, ktoré umožňujú vymazania akéhokoľvek zobrazeného elementu, pričom vymazanie iných elementov bolo uľahčené a navrhnuté.

6.2 Načítavanie diagramu z XML

V tomto šprinte bola implementovaná aj funkcionalita určená na načítavanie diagramu, respektíve 3D scény so sekvenčným diagramom zo súboru vo formáte XML. Ukladanie, resp. XML reprezentácia diagramu bola implementovaná v predchádzajúcom šprinte. V tomto šprinte boli kvôli efektívnejšiemu načítavaniu vykonané zmeny aj v mechanizme ukladania. Bola pridaná aj funkcionalita pre ukladanie fragmentu. Mechanizmus načítavania je vytvorený, momentálny stav umožňuje načítať objekty typu vrstva, čiara života a asynchrónna správa. Načítanie fragmentu bude dodatočne spracované v nasledujúcom šprinte. Efektívnejšie načítavanie si momentálne vyžaduje zmeny dátového modelu, hlavne v kontexte riešenia referencií medzi jednotlivými objektami.

6.3 Označenie čiar života ťahom myši – upravená logika

Úloha označenie čiar života ťahom myši bola začatá v minulom šprinte. Avšak logika, ktorá bola pri výbere použitá, sa ukázala ako nesprávna. Preto sme logiku upravili tak, že neberieme do ohľadu to, či používateľ klikol na čiaru života. Na každej vrstve používame zoradený zoznam čiar života. Pri stlačení tlačidla myši a jeho pustení si zaznamenáme body, kde tieto udalosti nastali. Tieto 2 body nám definujú výsek, z ktorého budeme čiary života vyberať. To, či čiaru života zaradíme do výberu, záleží od jej stredového bodu (V prípade, že bod patrí do intervalu, čiaru života vyberieme).

6.4 Oprava chýb

Nepredvídateľné miznutie fragmentu

Oprava chyby, pri ktorej sa pri výbere niektorej z vrstiev neznevidi̇teľnili fragmenty na ostatných vrstvách.

Určenie pozície fragmentu na zadných vrstvách

Stav pre výber pozície fragmentu nenastal pri umiestňovaní na niektorú zo zadných vrstiev (Po stlačení na *layer bookmark* sa zobrazil hneď formulár). Problém bol v tom, že po stlačení na *layer bookmark* (v stave *SelectFragmentLayerState*) sa nastavil hneď nový stav (*SelectFragmentPositionState*), preto pri uvoľnení tlačidla myši sa spracovávala funkcia

processRelease už v novo nastavenom stave. Bola pridaná nová podmienka tak, aby sa funkcia *processRelease* vykonala až potom, ako sa v tom istom stave vykonala funkcia *processClick*.

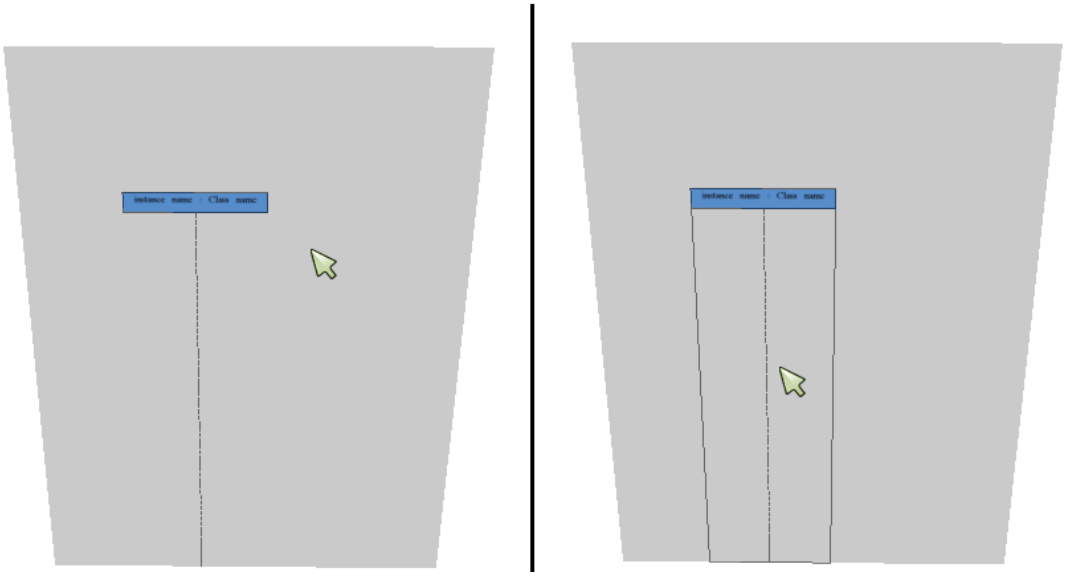
Doplnenie referencií na interakcie do čiar života

Ukladanie interakcií v objekte „lifeline“. Aby bolo možné vykonať zmazanie „lifeline“ ako aj správ závislých na tomto objekte, bolo potrebné doplniť referencie na správy, ktoré z daného objektu „lifeline“ vychádzajú aj doň vchádzajú.

7 Šprint č. 5

7.1 Zvýraznenie obalu pri prechode cez čiaru života

V rámci používateľskej interakcie s aplikáciou bola pridaná funkcionalita na zvýrazňovanie prvkov po prechode myšou. Pri prechode myšou ponad čiaru života sa zobrazí jej obal. V prípade, že je zobrazených viac vrstiev zvýraznenie sa spustí pre všetky čiary života, ktoré pretne lúč z kurzora myši. Obr. 7.1 ukazuje zvýraznenie. Vďaka súčasnej generickej implementácii bude rozšírenie zvýraznenia pre ďalšie prvky scény triviálnou úlohou. Momentálne sa zvýrazňovanie aplikuje len pre čiaru života.



Obr. 7.1: Zvýraznenie čiary života pri prechode kurzorom. Vľavo kurzor neukazuje na čiaru života a v pravo áno.

7.2 Vymazanie aktuálneho modelu

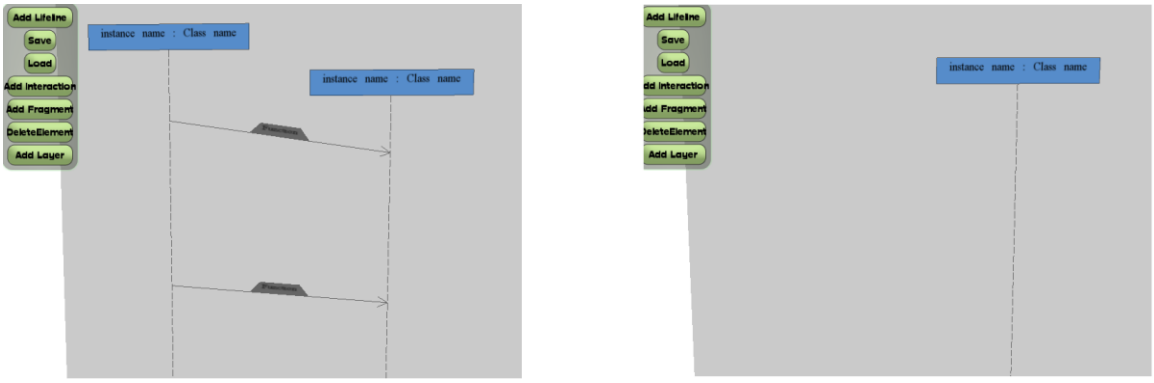
Pre potreby importu uloženého modelu z XML súboru je potrebné vyčistiť scénu od všetkých vykreslených elementov. Preto bol doplnený deštruktor pre element typu Layer. Po zavolaní metódy ostane scéna úplne prázdna. Funkcionalitu je v budúcnosti možné rozšíriť o tlačidlo vymazania scény, aby používateľ nemusel všetko odmazávať postupne.

7.3 Optimalizácia exportu a importu scény do/z XML

V tomto šprinte prebehla aj optimalizácia mechanizmov pre export diagramu do XML a jeho načítavanie. V prípade ukladania boli do modelu pridané referencie na rodičovský element, čo nám umožnilo zredukovať množstvo ukladaných informácii (nie je naďalej nutné ukladať pre každý element zoznam jeho potomkov). V mechanizme importu scény z XML opísaná optimalizácia umožnila zredukovať používanie pomocných dátových jednotiek, čím sa redukovala spotreba operačnej pamäte a aj nároky na výpočtové prostriedky. Načítavanie prebieha hierarchicky, čo znamená, že najprv sa načítajú všetky vrstvy, následne všetky čiary života a napokon aj asynchrónne správy. Po úspešnom vytvorení dátového modelu prebehne aj proces vykreslenia diagramu do scény.

7.4 Vymazanie čiary života z diagramu

Dokončenie úlohy z predchádzajúceho šprintu. Pridaná nová funkcionalita. Používateľ môže vymazať čiaru života z diagramu ako aj všetky jej správy. Vytvorený deštruktor pre čiaru života, ktorý spustí mazanie. Obrázok vľavo znázorňuje čiary života a interakcie medzi nimi pred vymazaním a obrázok vpravo po vymazaní.



Obr. 7.2: Vymazanie čiary života z diagramu vľavo pred a vpravo po vykonaní operácie

7.5 Oprava chýb

Fragment sa vykresľuje ako vodorovná čiara

Pri vykresľovaní fragmentu sa stávalo, že sa vykreslila len vodorovná čiara. Táto chyba sa vyskytla v prípade, že ťah vykreslenia fragmentu bol zľava doprava. Chyba nastala pri výbere

čiar života zahrnutých do fragmentu, kedy sa hodnota druhého vybraného bodu prepísala hodnotou prvého. Táto chyba bola odstránená pridaním lokálnej premennej.

Čiara života sa zvýrazní aj keď sa nemá

V prípade, že sa zobrazila iba jedna vrstva, napríklad pri pridaní čiary života, zvýrazňovali sa aj čiary života, ktoré boli skryté. Preto bola do zvýraznenia pridaná podmienka, ktorá overuje, či je zvýrazňovaný element zobrazený.

Vykresľovanie fragmentu na prekrývajúcu sa vrstvu

Fragment sa niekedy vykreslil na vrstvu, ktorá prekrývala požadovanú vrstvu. Problém bol v tom, že pri *raycast* funkcií sa vybrala zneviditeľnená vrstva, ktorá prekrývala požadovanú vrstvu. Z tohto dôvodu sa aj lokálne pozície vypočítali vzhľadom na nesprávnu vrstvu, a preto sa aj fragment vykresľoval nie len na nesprávnej vrstve ale aj na nesprávnych pozíciách. Súvisiaca podmienka bola upravená tak, aby bola akceptovaná len požadovaná vrstva, nie akákoľvek.

Pri ukladaní scény program zamrzne

Ak sa ukladala scéna, ktorá obsahovala fragment s viacerými čiarami života, program sa zasekol. Problém bol v metóde *saveElement* pre uloženie fragmentu, kde sa pri ukladaní čiar života do XML súboru, program zacyklil. Zodpovedajúci cyklus bol opravený.

8 Celkový pohľad na produkt

8.1 Prototyp ako celok

Dodaný prototyp bol implementovaný v jazyku C++ za použitia grafického rámca OGRE a knižníc OIS. Na vývoj je použité prostredie Eclipse s rozšírením CDT. V projekte sme pokračovali vo vývoji s použitím doteraz používaných technológií v tom istom vývojovom prostredí. Tento prototyp bol rozšírený o dodatočnú funkcionalitu, pričom práca na ňom naďalej aktívne prebieha.

Prototyp slúži na 3D vizualizáciu UML diagramov, pričom tento projekt sa zameriava na vizualizáciu sekvenčných diagramov. Trojdimenzionálnosť produktu je dosiahnutá zobrazovaním diagramov (prípadne ich častí) v rôznych vrstvách, ktoré sa nachádzajú v rozličnej hĺbke. Pohľad na diagram je dynamický, používateľ môže prezerať jednotlivé diagramy z rôznych uhlov. Taktiež je možné sa „pohybovať“ po diagrame a tým pádom sa zamerať na jednotlivé časti týchto diagramov, pričom pri pohybe po diagrame je možné pohybom priblížiť jednotlivé časti diagramu. Samotné ovládanie prototypu a aj pohybu po diagrame je veľmi jednoduché a intuitívne, pričom bližšie je opísané v používateľskej príručke, ktorá je súčasťou dokumentu.

Tento prototyp slúži na zobrazovanie a vytváranie UML sekvenčných diagramov. S použitím tretej dimenzie je možnosť posunutia diagramu do hĺbky. Priestorovosť diagramu môže byť dosiahnutá viacerými spôsobmi. Napríklad rozvrhnutím jedného diagramu do viacerých vrstiev po častiach, pričom stále je možné vytvárať interakcie medzi jednotlivými časťami diagramu napriek tomu, že sa nachádzajú na odlišných vrstvách.

Ďalším možným spôsobom je vizualizácia rôznych diagramov na odlišných vrstvách, pričom vzniká možnosť prehľadnejšieho prezerania diagramov a možných interakcií medzi nimi. Pri štandardných riešeniach by aktuálny scenár bol približne ten, že na rôznych oknách by mal používateľ vizualizované rôzne diagramy, pričom pre porovnanie týchto diagramov by bolo potrebné prepínanie medzi oknami. Avšak s pridaním tretej dimenzie do diagramu je možné toto porovnanie vykonávať v jednej obrazovke z rôznych uhlov a pohľadov.

Samotný prototyp obsahuje taktiež funkcionalitu, ktorá nie je priamo spojená s UML diagramom. Jedná sa napríklad o možnosť serializácie vytvoreného diagramu vo forme XML dokumentu pre prípad ďalšej práce na diagrame. Ďalšími riešenými problémami sú taktiež zarovnanie názvov

jednotlivých čiar života, prípadne ich zarovnanie z dôvodu neprekrývania sa vo vrstvách a taktiež aj tvorba diagramov podľa špecifikácie UML.

8.2 Architektúra produktu

Prototyp je samostatnou aplikáciou, preto má iba vnútornú architektúru. Návrh je založený na šiestich balíkoch. Každý balík vyjadruje logický celok, ktorý je čo najviac oddelený od ostatných súčastí. Každý balík sa špecializuje na inú časť programovej funkcionality.

Balík `OgreFramework` reprezentuje pracovný rámec OGRE. Používa sa na grafické vykresľovanie všetkých objektov a správu grafického prostredia.

Balík `Graphics` obsahuje všetky vykresľovacie algoritmy naviazané na použitie OgreFrameworku. Algoritmy na vykreslenie sú špecifické pre jednotlivé elementy, preto je potrebné, aby každý element mal vlastný postup vykreslenia.

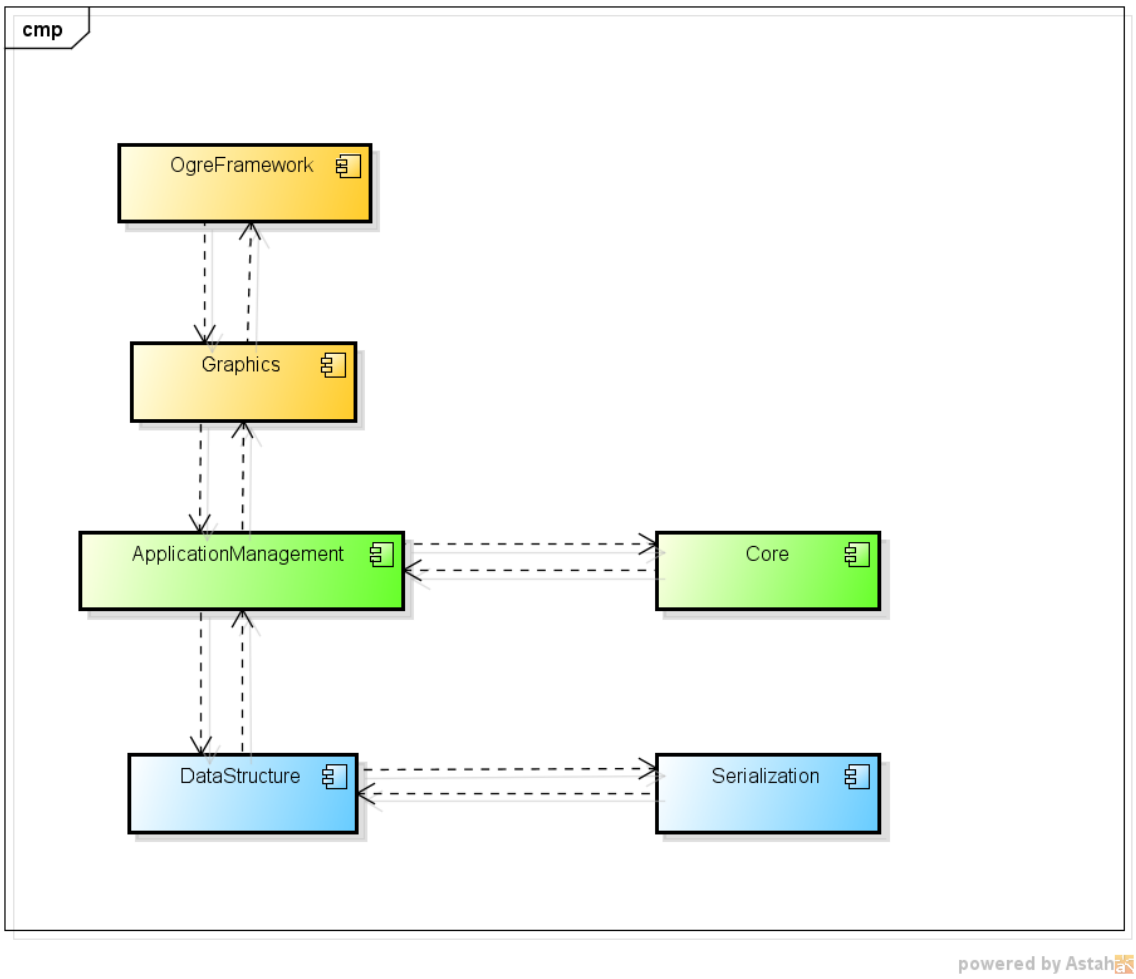
Balík `ApplicationManagement` je skupina centralizovaných tried slúžiacich na obsluhu a správu všetkých požiadaviek, ktoré vzniknú pri behu programu. Prototyp funguje ako stavový automat, ktorý pri prepínaní stavmi vykonáva iné činnosti.

Balík `Core` obsahuje konštruktory jednotlivých artefaktov reprezentovaných ako triedy. Konštruktory sú zapuzdrené pomocou vzoru Továreň.

Balík `DataStructure` je zložený z tried údajových štruktúr, ktoré reprezentujú objekty vykresľované do diagramu. Jednotlivé triedy obsahujú atribúty potrebné k serializácii.

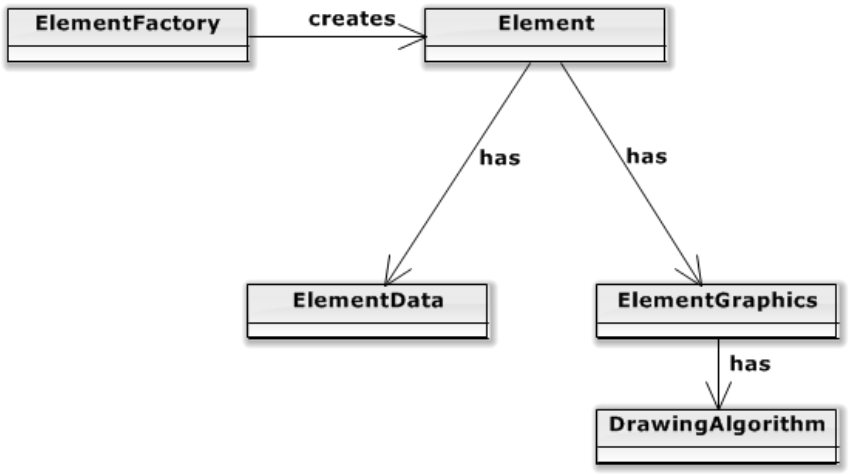
Balík `Serialization` obsahuje metódy ako uložiť a načítať údajové štruktúry. Toto je potrebné pri rekonštrukcií scény a zobrazovaného diagramu.

Náčrt architektúry produktu v kontexte balíkov možno vidieť na diagrame balíkov na Obr. 8.1.

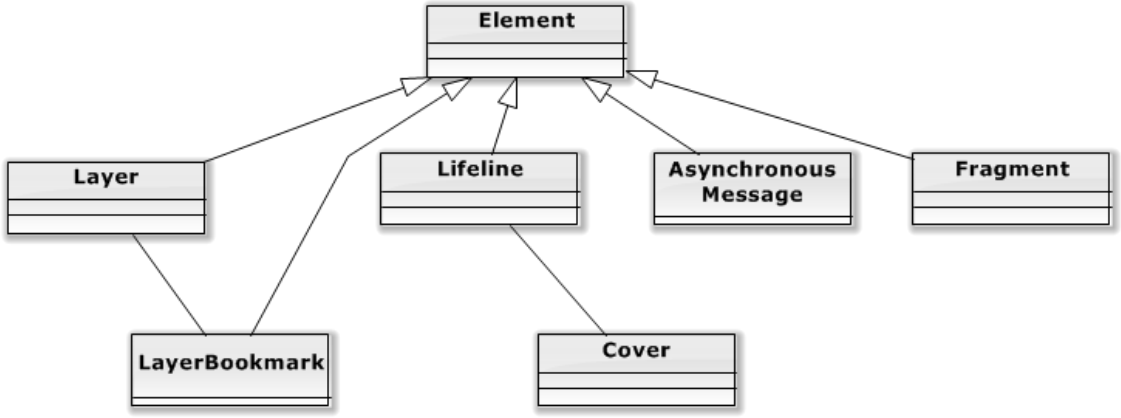


Obr. 8.1: Architektúra produktu zachytená diagramom balíkov

Diagram na Obr. 8.2 znázorňuje triedu *Element* a jej súčasti. Ide o dôležitú triedu, od ktorej sú odvodené všetky kľúčové triedy súvisiace s prvkami sekvenčného UML diagramu (Obr. 8.3). Každý objekt triedy *Element* má priradený objekt triedy *ElementData* a objekt triedy *ElementGraphics*. Objekt triedy *ElementData* obsahuje polia, ktoré reprezentujú vlastnosti prvkov UML diagramu vyplývajúce zo špecifikácie UML. Objekt triedy *ElementGraphics* obsahuje atribúty grafického zobrazenia daného prvku v 3D diagrame, a tiež objekt triedy *DrawingAlgorithm*, obsahujúci algoritmus na vykreslenie daného prvku v 3D diagrame.



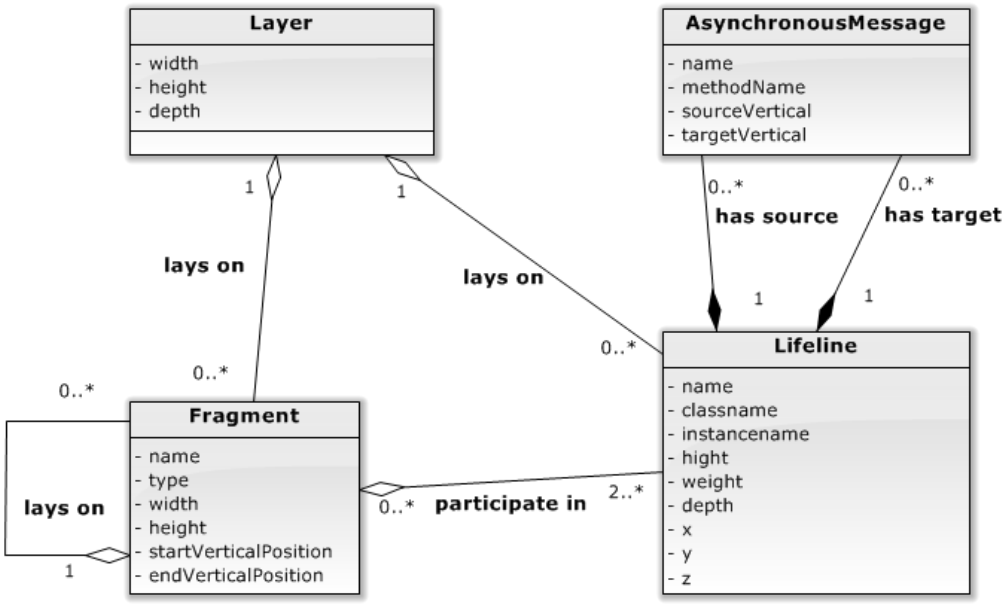
Obr. 8.2: Trieda Element



Obr. 8.3: Triedy odvodené od triedy Element

8.3 Dátový model

Logický model údajov je vyjadrený pomocou diagramu tried (Obr. 8.4). Popisuje hlavné prvky aplikácie. Tieto prvky predstavujú vizuálne prvky 3D diagramu. Znázornené atribúty sú použité pri ukladaní do súboru.



Obr. 8.4: Dátový model produktu zachytený diagramom tried

Layer (Vrstva)

Tento prvok predstavuje jednu vrstvu v 3D diagrame. Rôzne vrstvy sa líšia hlavne v pozícií (*z*) v rámci diagramu. *Layer* agreguje v sebe viaceré inštancie *Lifeline*, ako v klasickom sekvenčnom diagrame.

Lifeline (Čiara života)

Predstavuje jeden zo stavebných prvkov UML sekvenčného diagramu, obsahuje atribúty ako *classname* a *instancename* charakterizujúce triedu a objekt, pre ktorý bola vytvorená. Má presne určenú pozíciu v rámci *Layer* a tiež rozmery ako výška a šírka.

Asynchronous Message (Asynchrónna správa)

Predstavuje jeden zo stavebných prvkov UML sekvenčného diagramu. Pretože správa existuje medzi volajúcim a volaným objektom, v diagrame má *Asynchronous Message* určenú zdrojovú a cieľovú *Lifeline*. Atribút *methodName* predstavuje názov volanej metódy v cieľovom objekte. *SourceVertical* a *targetVertical* sú pozície začiatku a konca správy vzhľadom na príslušné *Lifeline*.

Fragment

Predstavuje ďalší prvok UML diagramu. Fragment vymedzuje určitú oblasť v rámci každej *lifeline*, ktorá doň bola zahrnutá. Fragmenty môžu vymedzovať oblasti aj v rámci iných fragmentov (Obr. 8.4).

A Používateľská príručka

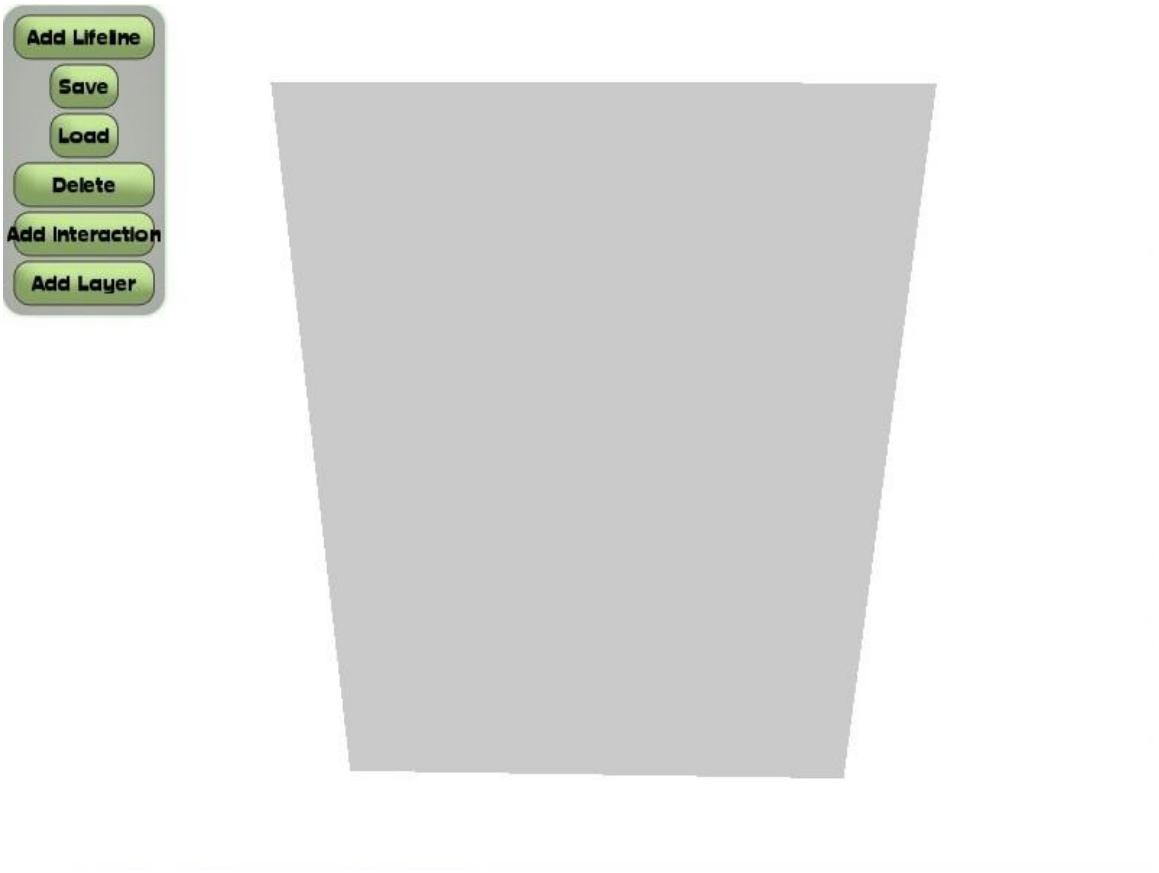
Aplikácia sa spúšťa cez súbor timak.exe, nachádzajúci sa v priečinku ...\3D_UML\Debug

Po spustení si používateľ vyberie Direct3D9 vykresľovací subsystém (Obr. príloha 8.1).

Obr. príloha 8.1: Výber vykresľovacieho subsystému

Následne sa používateľ dostane do používateľského prostredia v sekvenčnom diagrame.

Pohyb po priestore umožňujú klávesy W (šípka hore), A (šípka vľavo), S (šípka dole) a D (šípka vpravo). Používateľovi je prezentovaná úvodná obrazovka (Obr. príloha 8.2).

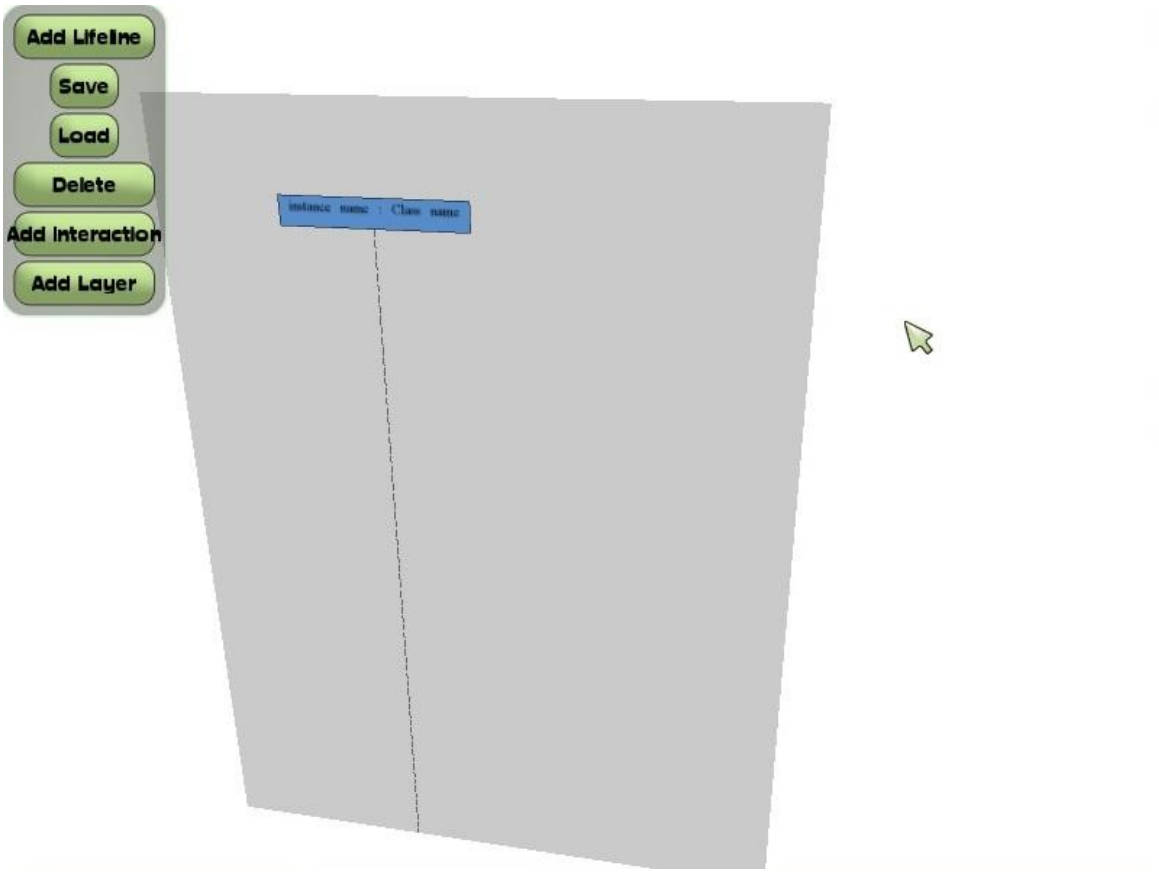


Obr. príloha 8.2: Úvodná obrazovka

V strede sa nachádza šedá plocha vrstvy. Je to oblasť, kde sa budú nachádzať prvky sekvenčného diagramu.

V ľavom hornom rohu sa nachádza menu obsahujúce položky:

AddLifeline: tlačidlo pridá novú čiaru života na zvolenú vrstvu. Používateľ vyberie vrstvu, na ktorej sa bude nový prvok nachádzať. Následne na vrstve vyznačí pozíciu a vyplní formulár, kde špecifikuje detaily objektu (Obr. príloha 8.3).



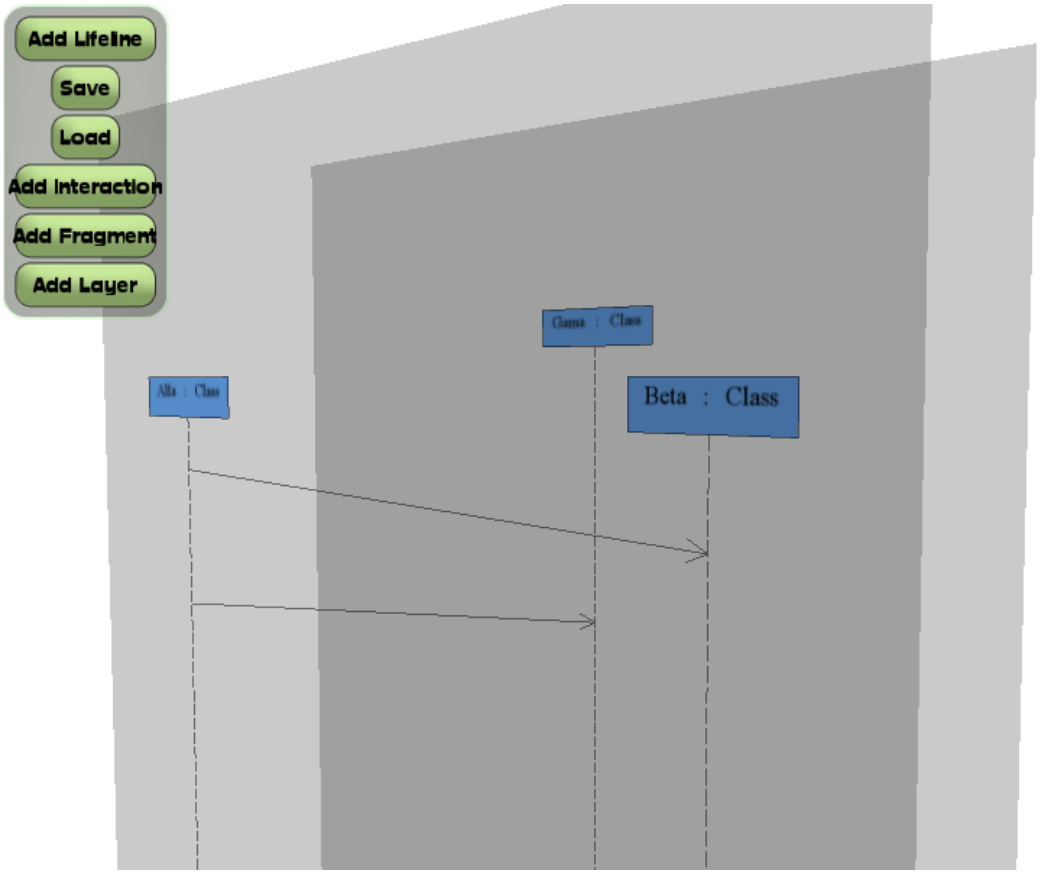
Obr. príloha 8.3: Vloženie novej čiary života (life-line)

Save: tlačidlo save uloží sekvenčný diagram do súboru.

Load: tlačidlo na načítanie sekvenčného diagramu zo súboru.

Delete: tlačidlo slúžiace na vymazanie objektov. Užívateľ si zvolí vrstvu, z ktorej chce objekt vymazať a následne konkrétny objekt, ktorý bude vymazaný.

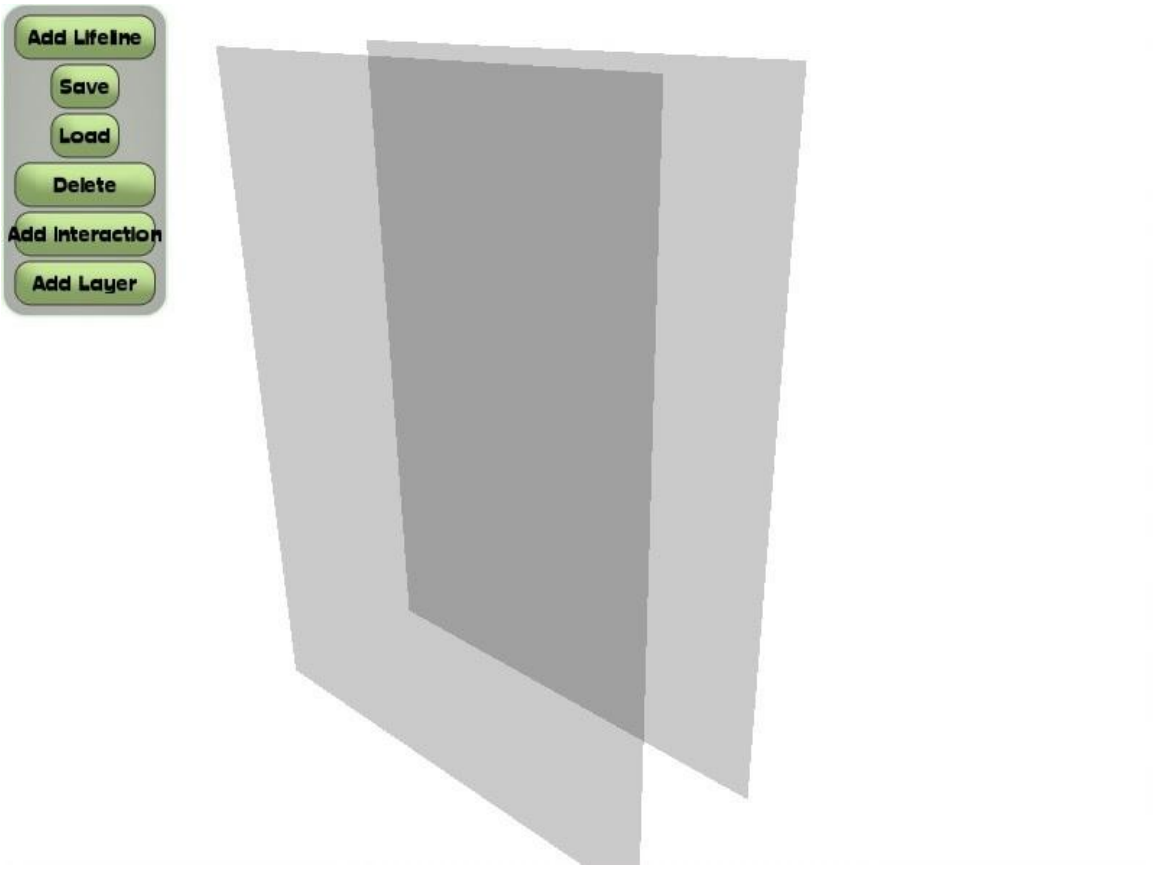
AddInteraction: tlačidlo pridávajúce interakcie medzi čiarami života. Užívateľ klikne na vrstvu, následne na zdrojovú čiaru života, potom na vrstvu a cieľovú čiaru života. Nakoniec vyplní názov operácie (voliteľná položka Obr. príloha 8.4).



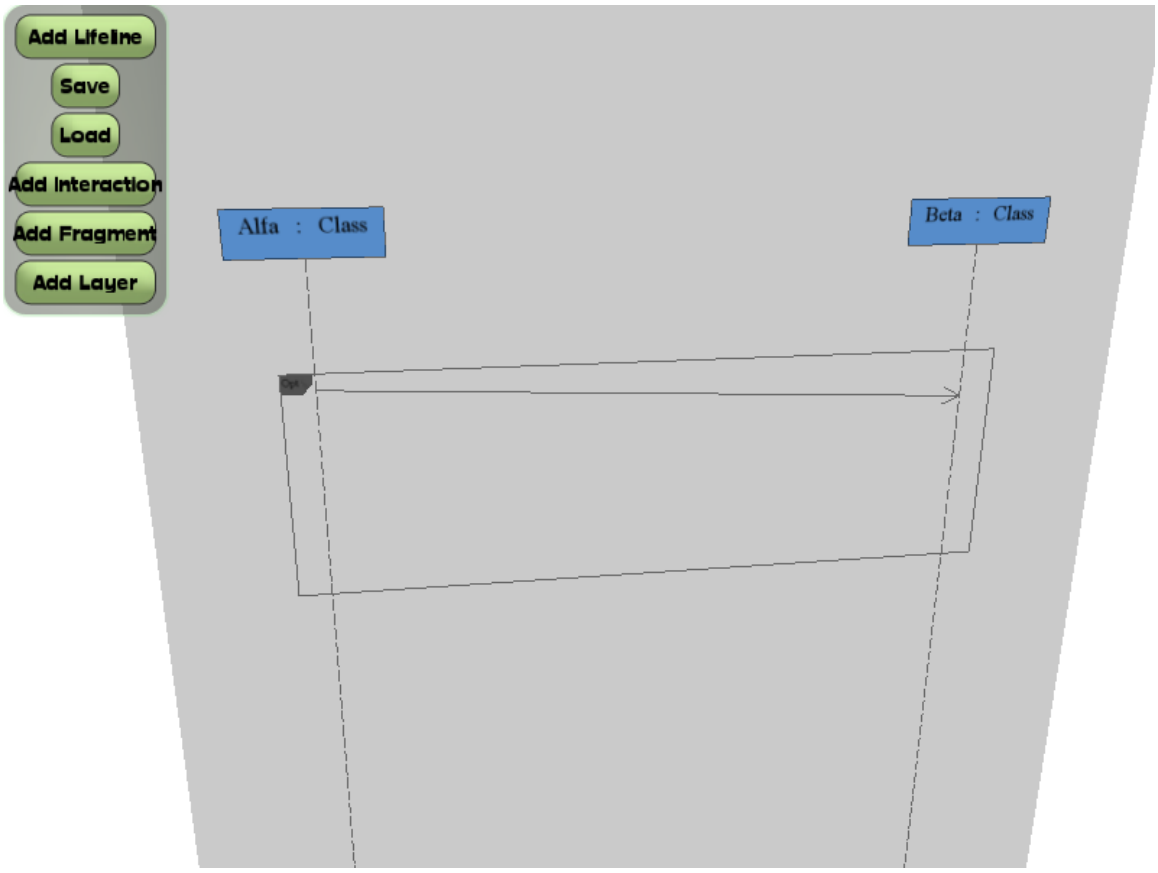
Obr. príloha 8.4: Pridanie interakcie

AddLayer: tlačidlo pridania novej vrstvy do modelu. Nová vrstva sa vytvorí za predchádzajúcou vrstvou (Obr. príloha 8.5).

AddFragment: tlačidlo na pridanie nového fragmentu. Po vybratí vrstvy, na ktorú sa fragment vykreslí, stačí kurzor myši so stlačeným tlačidlom myši pretiahnuť cez všetky čiary života, ktoré používateľ do fragmentu chce zahrnúť. Výška a výšková pozícia (v rámci čiary života) takto vytvoreného pomyselného obdĺžnika predstavuje výšku a výškovú pozíciu vytvoreného fragmentu. Na Obr. príloha 8.6 je zobrazený fragment.



Obr. príloha 8.5: Pridanie novej vrstvy



Obr. príloha 8.6: Fragment

B Preberací protokol

PREBERACÍ PROTOKOL ZA ZIMNÝ SEMESTER

Tímový projekt 2013/2014

Tím 02 – GAMATEPI

Predmet odovzdávania:

- Dokumentácia k riadeniu projektu – verzia za zimný semester
- Dokumentáciu k inžinierskemu dielu – verzia za zimný semester

---

Vedúci projektu: Ing. Ivan Polášek, PhD
Podpisom potvrdzuje prebratie vyššie uvedených častí projektu.

V Bratislave

....

Dátum Podpis