Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Ilkovičova2, 842 16 Bratislava 4

3D UML

dokumentácia k produktu - prvé tri šprinty, big picture

Vedúci tímu: Ing. Ivan Polášek Phd.

Členovia tímu: Bc. Adam Kulíšek, Bc. Matej Jenis, Bc. Rami Mtier, Bc. Tomas Hnojčík, Bc. Patrik Kolek, Bc. Boris Buček

Akademický rok: 2015/2016

1 Podiel na vytváraní dokumentu

Názov kapitoly	Autor
Úvod	Bc. Patrik Kolek, Bc. Adam Kulíšek
Moduly systému	všetci členovia tímu
Testovanie	Bc. Boris Buček
Príručky - inštalačná príručka	všetci členovia tímu
Technická dokumentácia	všetci členovia tímu

Tab. 1: Autori jednotlivých kapitol dokumentu

2 Úvod

Tento dokument bol vytvorený za účelom zdokumentovania výstupu tímového projektu č. 13 s názvom Triskaidekaphobiacs v akademickom roku 2015/2016 za zimný semester na predmete *Tímový projekt*. Dokument je rozdelený na niekoľko celkov. V úvode sa čitateľ oboznámi s cieľmi projektu, ktoré boli spolu s vlastníkom projektu stanovené v priebehu prvých týždňov zimného semestra a získa základný prehľad o vyvíjanom systéme pomocou náčrtu jeho architektúry. Nasledujúca kapitola *Moduly Systému* je rozdelená na štyri celky (analýza, návrh, implementácia a testovanie), a obsahuje hlavne podrobnejšie opisy jednotlivých komponentov a diagramy, ktoré zobrazujú ich štruktúru a väzby. V prílohe sa nachádza inštalačná príručka, ktorú sme v prvých týždňoch museli spracovať na to, aby si každý člen tímu individuálne mohol spustiť projekt na vlastnom stroji.

2.1 Globálne ciele projektu

V rámci zimného a letného semestra je naším hlavným cieľom integrácia existujúcich riešení(prototyp sekvenčného diagramu, prototyp kombinovaných fragmentov a prototyp diagramu aktivít) do navrhnutej architektúry MMVCC. Práve podľa tejto architektúry bol vytvorený prototyp pre diagram aktivít, do ktorého budeme pridávať riešenie pre sekvenčný diagram. Projekt je momentálne v takom stave, že viacero jeho častí je implementovaných ako samostatné nezávislé prototypy, ktoré je potrebné osobitne nakonfigurovať vo vývojovom prostredí a spustiť ako osobitnú aplikáciu. Od stiahnutia zdrojových kódov aplikácie po ich buildovanie do spustiteľnej formy môže nastať množstvo komplikácií, čím sa tento proces značne a zbytočne predĺži. Pokiaľ by používateľ chcel spustiť iný prototyp projektu, musel by takýto proces vykonať pre každý z prototypov separátne. Práve tento problém nás motivoval pre vytvorenie jednotného riešenia, ktoré bude podporovať prácu s viacerými typmi diagramov v 3D prostredí. Spájaním prototypov diagramu aktivít, sekvenčného diagramu a kombinovaných fragmentov chceme vytvoriť prehľadný, jednoducho inštalovateľný systém, do ktorého je možné jednoducho zapracovať akúkoľvek ďalšiu funkcionalitu. Pre tento účel je potrebné implementovať `Metamodel` a `MetamodelController` pre jednotlivé diagramy tak, aby spadali do architektúry prototypu diagramu aktivít.

2.1.1 Produktový backlog - Big picture

- Zobrazenie sekvenčného diagramu v prostredí a architektúre aktivity diagramu
- Dorobenie fragmentov do novej architektúry sekvenčného diagramu
- Dorobenie editačných funkcií

- Code refactoring
    - vylepšenie insertu
    - lepšie komentáre
    - zrušenie nepotrebných namespaceov
- Podrobná dokumentácia
    - Diagramy tried zachytávajúce štruktúru prototypu
- webová stránka 3D LAB
- Eliminácia branchov na bitbuckete (z troch vznikne jeden)

2.1.2 Prehľad definovaných používateľských príbehov:

Ako používateľ 3D UML

- **Základné funkcie editovania**

  1. Ako používateľ 3D UML prototypu potrebujem pri vstupe do systému vidieť základné menu vstupu do jednotlivých druhov diagramov, kde sa mi hneď po stlačení voľby „Class diagram“, „Sequence diagram“ alebo „Activity diagram“ zobrazia ich editory.

  2. Pri vstupe do editora “Sequence diagram” chcem vidieť ponuku jednotlivých možných elementov na vloženie do diagramu (čiary života - lifelines a interakcie - interactions).

  3. Musím mať možnosť vkladať nové prvky (lifelines a interactions).

  4. Potrebujem zvolené prvky aj vyraďovať.

- **Presuny**

  5. Chcem aj presúvať čiary života v rámci jednej vrstvy, pričom ťahajú so sebou aj interakcie.

  6. Potrebujem presúvať čiary života aj medzi vrstvami, pričom ťahajú so sebou aj interakcie.

  7. Chcem aj presúvať interakcie v rámci jednej vrstvy hore/dole.

  8. Potrebujem presúvať aj interakcie medzi vrstvami.

  9. Chcem aj presúvať interakcie v rámci jednej vrstvy zmenou source lifeline a/alebo destination lifeline.

  10. Potrebujem presúvať interakcie zmenou source lifeline a/alebo destination lifeline aj medzi vrstvami.

- **Fragmenty**

  11. Musím mať možnot pridávať nielen objekty a interakcie, ale aj jednoduché fragmenty typu Loop a Opt.

  12. Potrebujem pridávať aj zložené fragmenty typu Alt a Par (ako počet operandov).

  13. Chcem vnárať fragmenty jeden do druhého.

14. Chcem aby sa zložené fragmenty typu Alt a Par vedeli v animácii zobraziť za sebou v priestore, alebo pod sebou v rámci jednej vrstvy.
15. Okrem pridania fragmentov ich potrebujem aj odoberať.
16. Potrebujem ich aj presúvať hore dole po vrstve.
17. Potrebujem ich aj rozširovať alebo zužovať vo vrstve.

Ako správca kódu:

18. Potrebujem premigrovať sekvenčný diagram do architektúry diagramu aktivít po minuloročnom tíme 04/2014, čo znamená MMVCC.
19. Páčilo by sa mi owrapovať pomocou adapter/strategy 3D Controller, View a možno aj 3D Model kvôli perspektívnej zmene.
20. Je potrebné aj rozbehať metamodel sekvenčného diagramu ako aj pridať správu fragmentov (Richard Belan).
21. Bolo by pekné refaktorovať kód, vylepšiť insert, opraviť Popup okno.

Ako správca dokumentácie k produktu:

22. Potrebujem podrobnú technickú dokumentáciu novej architektúry.
23. Potrebujem inštalačnú príručku.
24. Potrebujem podrobnú technickú dokumentáciu jednotlivých funkcií.

Ako správca 3D laboratória

25. Potrebujem 3D LAB webovú stránku.
26. Chcel by som mať možnosť aktualizovať obsah.

2.1.3 Ciele pre zimný semester

Hlavným cieľom zimného semestra je návrh metamodelu pre sekvenčný diagram, pomocou ktorého budeme vykreslovať jednotlivé elementy v 3D priestore. Aby bolo možné vykreslovať tieto elementy, je potrebná implementácia pre zobrazenie sekvenčného diagramu v rámci hore uvedenej architektúry (obrázok 1). Zobrazenie spočíva vo vykreslení scény, do ktorej bude možné vkladať elementy pre sekvenčný diagram (životná čiara, správa a kombinované fragmenty - ALT, OPT, LOOP). Tieto elementy budú k dispozícii v rámci menu, ktoré je súčasťou vykreslenia prázdnej scény.
Medzi ciele pre zimný semester patrí aj refaktorovanie projektu pre aktivity diagram, do ktorého chceme pridať existujúce prototypy pre sekvenčné diagramy (editačné funkcie a fragmenty). Refaktorovanie je potrebné pre vytvorenie nezávislých komponentov pre prácu v 3D priestore. Tieto komponenty by mali byť nezávislé od typu diagramu. Mali by sa starať len o prácu a vykreslovanie elementov v 3D priestore. Výsledkom refaktorovania by mal byť aj prehľadnejší a čitateľnejší kód, do ktorého budeme implementovať prototyp sekvenčného diagramu.

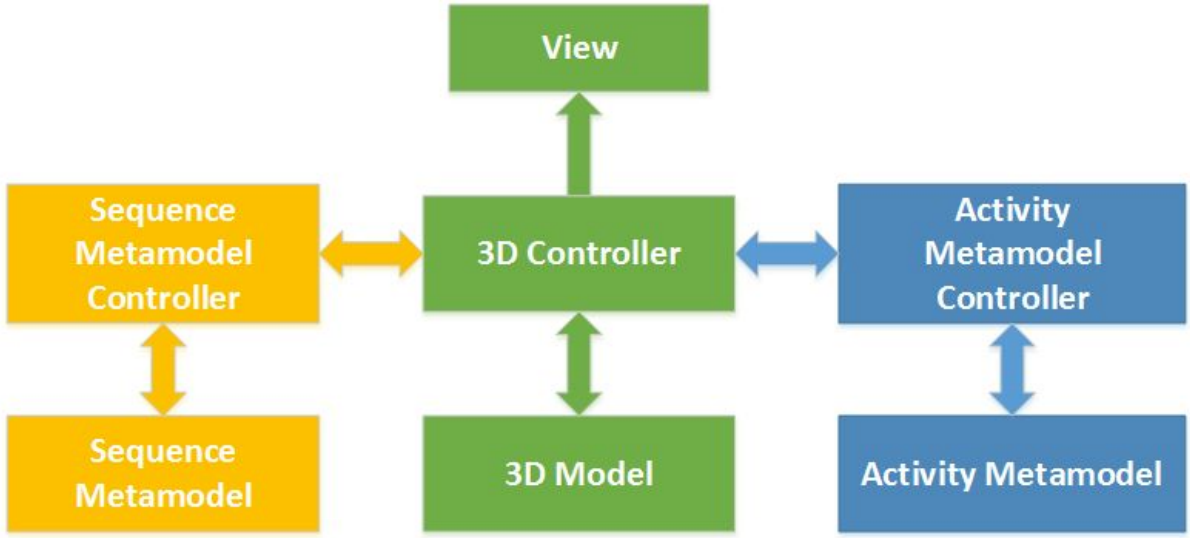
2.1.4 Ciele pre letný semester

V rámci letného semestra chceme dorobiť prácu s editačnými funkciami pre sekvenčný diagram. Doplnenou funkcionalitou bude možné pohybovanie životných čiar na danej vrstve, pohybovanie sa pomocou asynchrónnej správy a rovnako aj medzivrstvový pohyb. Ďalším cieľom bude úprava a finálna prehliadka kódu vytvoreného prototypu. Výsledkom by malo byť spojenie troch prototypov do finálneho projektu podporujúceho prácu s diagramom aktivít a rovnako aj sekvenčného diagramu v rámci jednej aplikácie. Hlavnou motiváciou pri vytváraní finálneho prototypu je eliminácia vetiev pre každý existujúci prototyp. Momentálne je každý jeden prototyp vytvorený v rozličných vetvách. Integráciou prototypov pre sekvenčný diagram do architektúry diagramu aktivít nám vznikne iba jedna vetva pre jeden finálny prototyp.

Počas práce v obidvoch semestroch je cieľom aj priebežné dokumentovanie a refaktorovanie prototypov pre dosiahnutie lepšej prehľadnosti a čitateľnosti kódu.

2.2 Celkový pohľad na systém

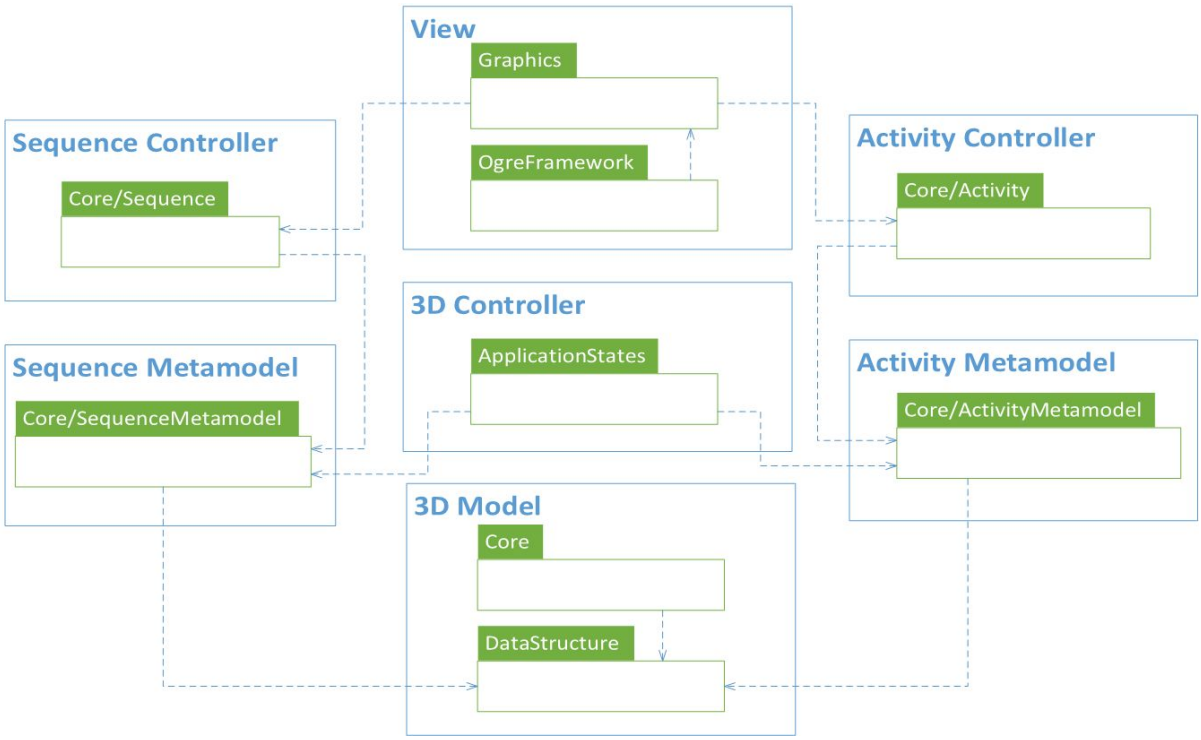
Na obrázku 1 je znázornený jednoduchý náhľad do architektúry prototypu, ktorú chceme dosiahnuť po integrácii prototypov sekvenčného diagramu (kombinované fragmenty a editačné funckie) do prototypu diagramu aktivít. Obrázok je rozdelený do troch farebne označených častí. Zelenou farbou sú znázornené komponenty, ktoré sú spoločné pre všetky typy diagramov. Ich komunikáciou a spoluprácou zabezpečíme vykreslenie jednotlivých diagramov. V pravej časti obrázka sú komponenty špecifické pre diagram aktivít. Návrh a implementácia týchto komponentov je výsledkom práce minuloročného tímového projektu. Podobným spôsobom by sme chceli navrhnúť komponenty pre sekvenčný diagram (oranžová časť). Pre implementáciu tejto myšlienky je potrebné modifikovať časti existujúceho prototypu diagramu aktivít.



Obr. 1: Architektúra prototypu po integrácii sekvenčného diagramu

3 Moduly Systému

Systém je zložený z niekoľkých balíkov, ktoré zoskupujú zdrojové kódy do logických celkov.



Obr. 2: Balíková štruktúra zdrojového kódu s hlavnými komponentami

- **View:** úlohou `View` komponentu je vykresľovanie elementov diagramu a vrstiev v 3D priestore. Tento 3D priestor je definovaný scénou, v ktorej sa môže používateľ pohybovať. Rovnako do tejto scény môže pridávať elementy, ktoré má k dispozícii v rámci menu pre konkrétny typ diagramu. Aby sme boli schopní tieto elementy vykresliť, musí mať každý element definovaný svoj vlastný algoritmus a infomácie o vykreslení ako napríklad typ použitého písma. Vykresľovanie elementov sekvenčného diagramu definované a implementované v triedach v balíku `Graphics`. Pre samotné grafické vykreslenie elemetov do scény sa využíva knižnica Ogre, ktorej základné grafické operácie sú definované v balíku `OgreFramework`.
- **3D Controller:** stará sa o vytváranie elementov diagramu. Rovnako jeho úlohou je aj kontrolovať stav aplikácie, v ktorej sa nachádzame. Všetky triedy, ktoré sa starajú o riadenie a kontrolu stavu aplikácie, v ktorej sa nachádza, sú zahrnuté v balíku `ApplicationStates`.
- **3D Model:** nakoľko pracujeme v 3D priestore, musíme poznať štruktúru dát elementov pre túto dimenziou. Všetky modely elemetov sú zachytené v balíkoch `Core` a `Datastructure`.
- **Activity/Sequence Controller:** ich úlohou je vytvárať a modifikovať model elementy v rámci UML diagramu (pridávanie, mazanie a aktualizácia elementov).
- **Activity/Sequence Metamodel:** určujú vzťahy medzi komponentami jednotlivých diagramov UML diagram.

3.1 Prototyp diagramu aktivít

Táto vetva projektu bola vytvorená v rámci tímového projektu v minulom akademickom roku.

3.1.1 Analýza

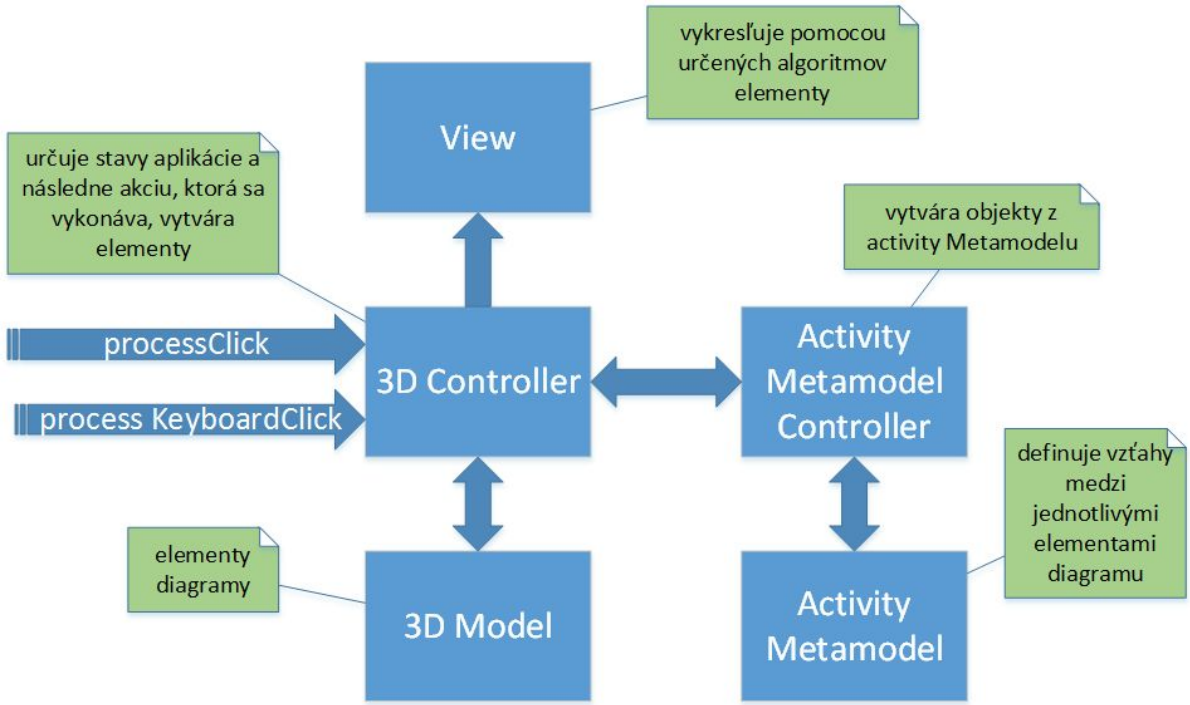
V tejto časti je opísaný existujúci prototyp diagramu aktivít. Tento prototyp je funkčný program písaný v jazyku C++ s použitou grafickou knižnicou Ogre[1]. Použitý architektonický štýl je modifikovaný MVC.

3.1.1.1 Opis architektonického štýlu

Modifikácia MVC architektonického štýlu spočíva v pridaní jedného extra kontrolera a jedného modelu. Pridaný model nazývame metamodel a slúži na určenie vzťahov medzi jednotlivými elementami. Údaje z metamodelu môžu byť využité napríklad na automatické generovanie diagramov z prototypu do nástroja Enterprise Architect a naopak. Pridaný kontroler je Activity Metamodel Controller, ktorý vytvára a modifikuje metamodel elementy v aktivity diagrame. Druhým modelom, s ktorým pracujeme je

[1] http://www.ogre3d.org/

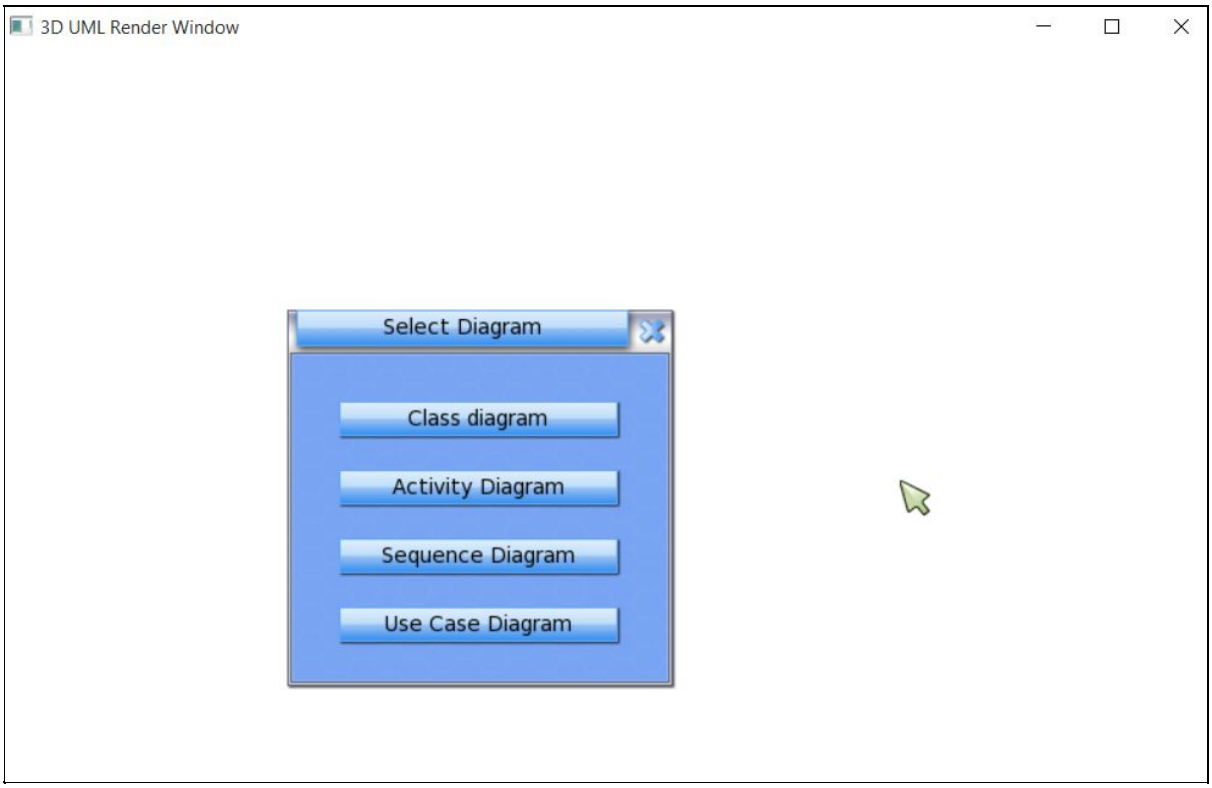
3D Model, ktorý definuje údaje v elemente, informácie o umiestnení elementov do scény a ich grafickú reprezentáciu. Obrázok zobrazuje prepojenie popísaných elementov vzniknutého architektonického štýlu MMVCC.



Obr. 3: Architektúra prototypu diagramu aktivít

3.1.1.2 Komponent 3D controller

Aplikácia používa na definovanie akcií stavy v ktorých sa môže nachádzať. Všetky stavy sú uložené v triede `ApplicationStateManager`, ktorá používa architektonický vzor *Singleton*. Pri spustení programu sa vyvolá v triede `Main` metóda `CreateScene`, ktorá určí aplikácií stav `SelectDiagramContext`. Tento stav vytvorí úvodnú obrazovku z ktorej je možné vyberať typ vytváraného diagramu (obrázok 4). Po zvolení vhodného diagramu sa vykreslí prázdna scéna, v ktorej je možné vytvárať 3D UML diagram. V prototype z minuloročného tímového projektu je funkčný len aktivity diagram. Jednotlivé stavy sú definované v balíku `ApplicationStates`.

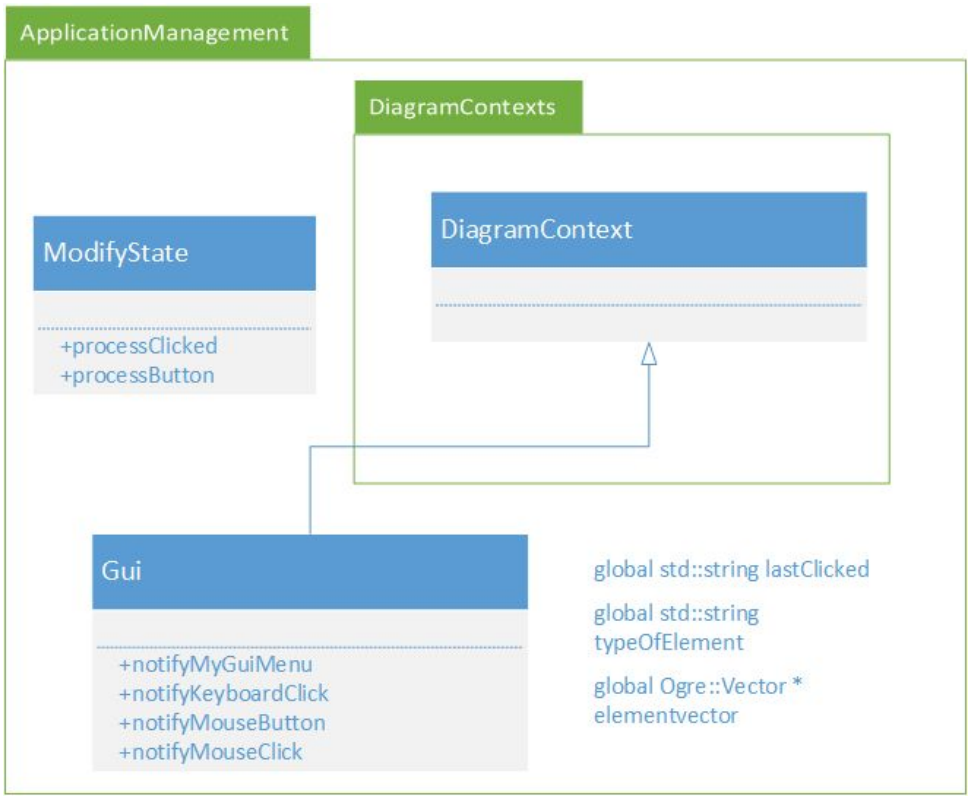


Obr. 4: Používateľské menu

Pri interakcií používateľa s aplikáciu sa nachádza v editačnom stave. V editačnom móde je možné vytvárať potrebné diagramy. Tento proces sa v prototype deje pomocou triedy `ModifyState`. Táto trieda inicializuje akciu pomocou metód `procesClick` (kliknutie tlačidlom na myši) a `procesButton` (kliknutie klávesou). V metódach nájde vytvorený objekt typu `DiagramContext`, ktorý v prípade diagramu aktivít je trieda `Gui`. Následne vyvoláva metódy podľa typu akcie

- **`notifyMyGuiMenu()` -** reaguje pri kliknutí na Menu v aktivity diagrame
- **`notifyKeyboardClick()`** - reaguje pri kliknutí na tlačidlo v klávesnici
- **`notifyMouseButtonClick()`** - reaguje pri kliknutí na tlačidlo
- **`notifyMouseClick()`** - reaguje pri kliknutí na vykreslený priestor

Aplikácia si v globálnych premenných pamätá posledné kliknuté tlačidlo, súradnice bodu na vykreslenie a typ elementu s ktorým sa pracuje. Vyššie opísaná štruktúra v aplikácií sa nachádza na obrázku 5



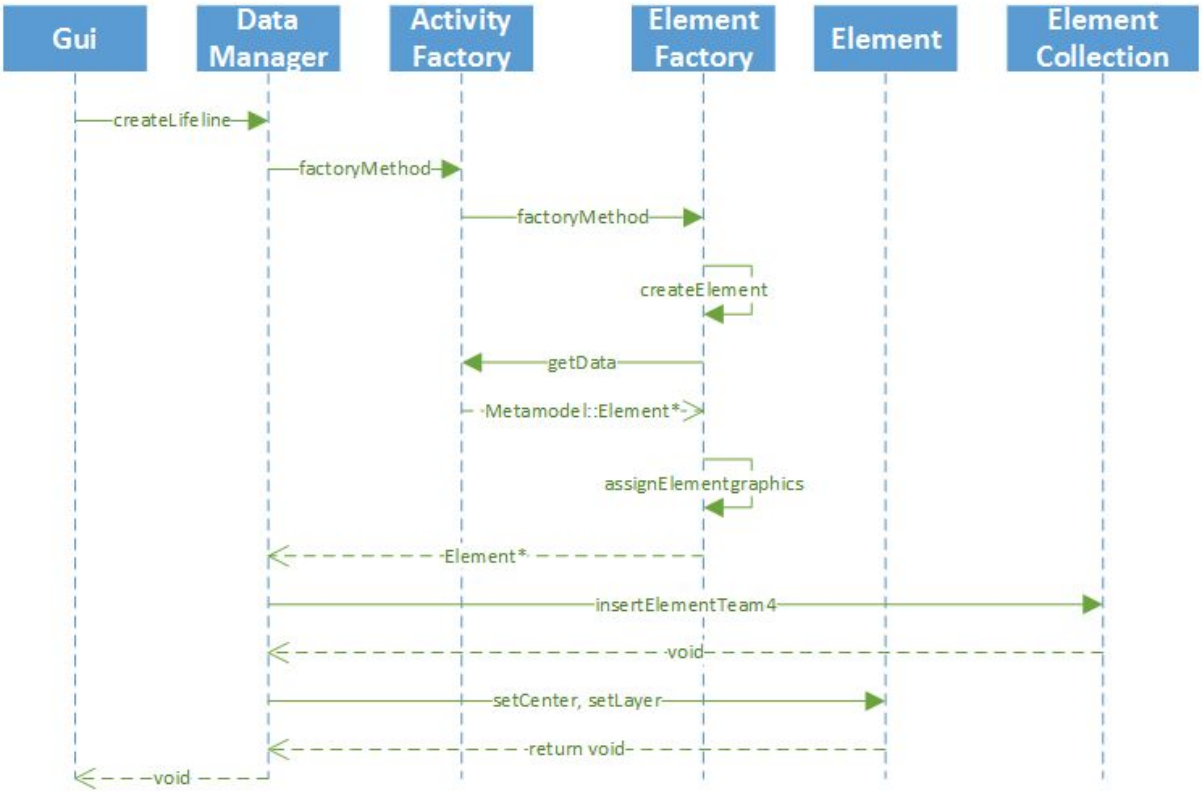
Obr. 5: Štruktúra aplikácií

3.1.1.3 Vkladanie elementu

Elementy analyzovaného prototypu sa ukladajú do pamäte vo forme dvojíc(`Element`, `MetamodelElement`):

- **`Element`** - triedy v štruktúre `Core/activity`, ktoré dedia od triedy Element
- **`MetamodelElement`** - elementy v štruktúre `Core/Metamodel`, ktoré dedia od triedy `Metamodel`

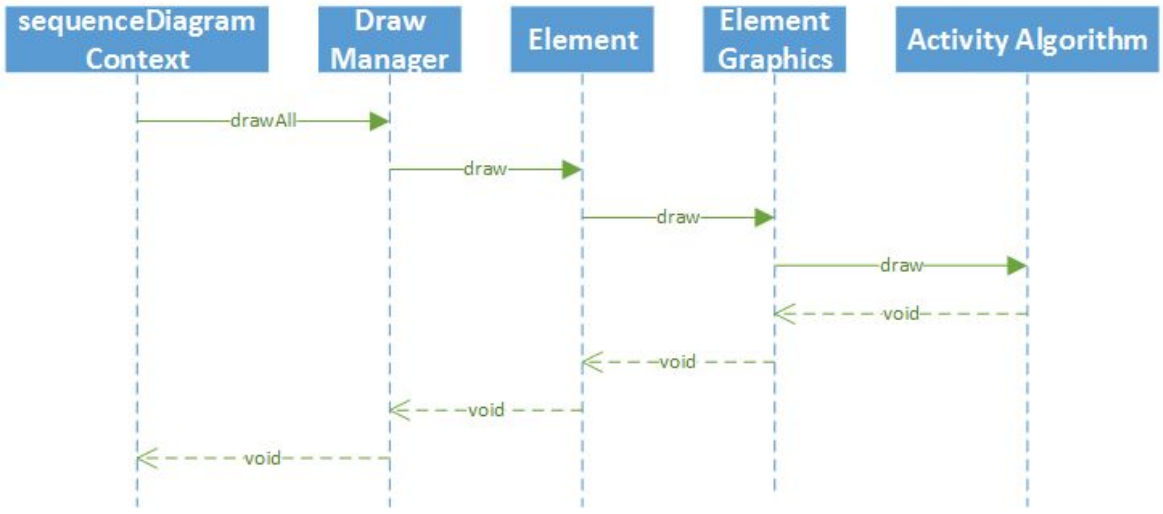
Dvojice objektov sa ukladajú do singleton štruktúry triedy `ElementCollection`. Triedy dediace od triedy `ElementFactory` vytvárajú elementy aktivity diagramu. Všetky komponety sa nachádzajú v statickej štruktúre obsiahnutej v triede `ElementCollection`. Zjednodušený proces ukladania je zobrazený na obrázku 6 vo forme sekvečného diagramu.



Obr. 6: Vkladanie aktivity do Model komponentu

3.1.1.4 Vykresľovanie komponetov

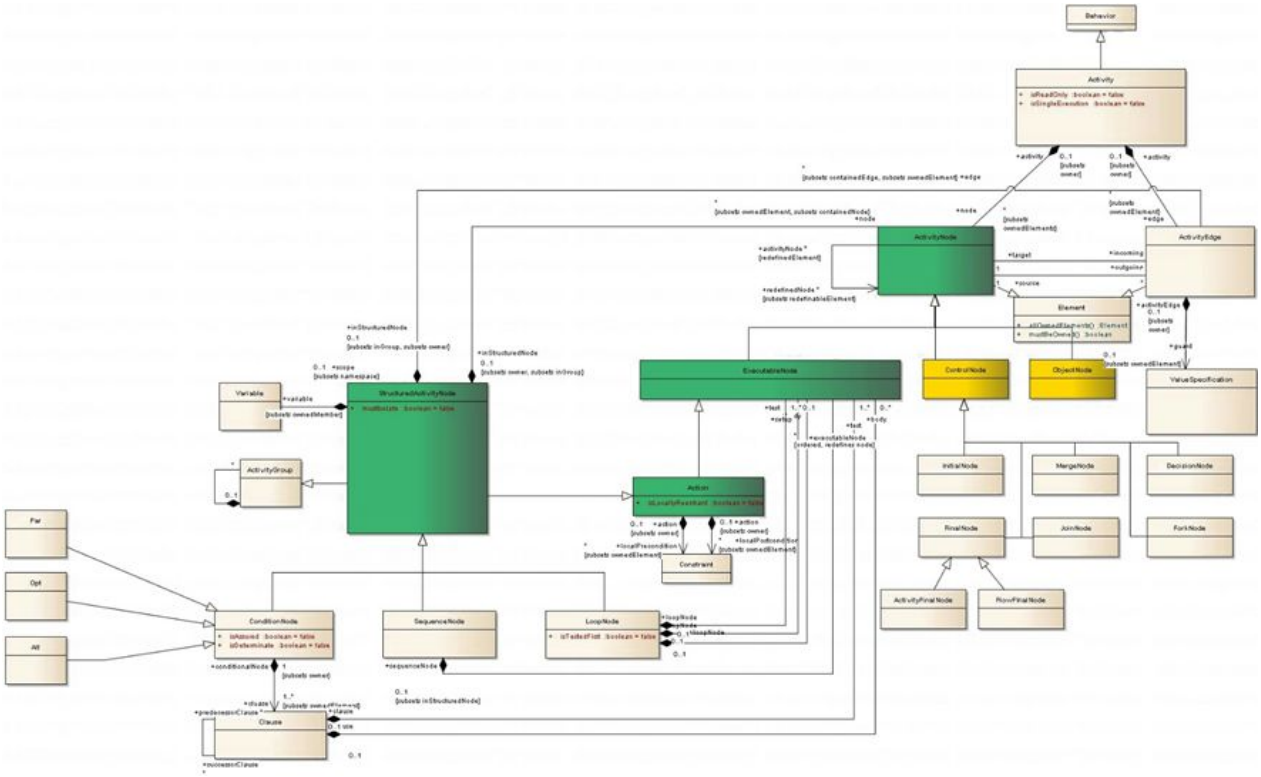
Po vložení komponentov sa v objekte typu `DrawManager` vyvolá metóda, ktorá vykreslí všetky objekty uchovávané v štruktúre `ElementCollection`. Pre každý pár (`Element`, `MetamodelElement`) sa vyvolá metóda draw v objekte typu `Element`. Každý `Element` má agregovaný `ElementGraphics`. Ten obsahuje kresliaci algoritmus. Proces je zobrazený na obrázku 7. Nevýhodou zostáva vykresľovanie všetkých objektov, čo nie je veľmi efektívne.



Obr. 7: Sekvenčný diagram algoritmu vykresľovania

3.1.1.5 Metamodel prototypu diagramu aktivít

Metamodel je štandartne definovaný pre aktivity diagram vo forme definovanej na obrázku 8. Ten je použitý v prototype diagramu aktivít.



Obr. 8: Diagram metamodelu diagramu aktivít

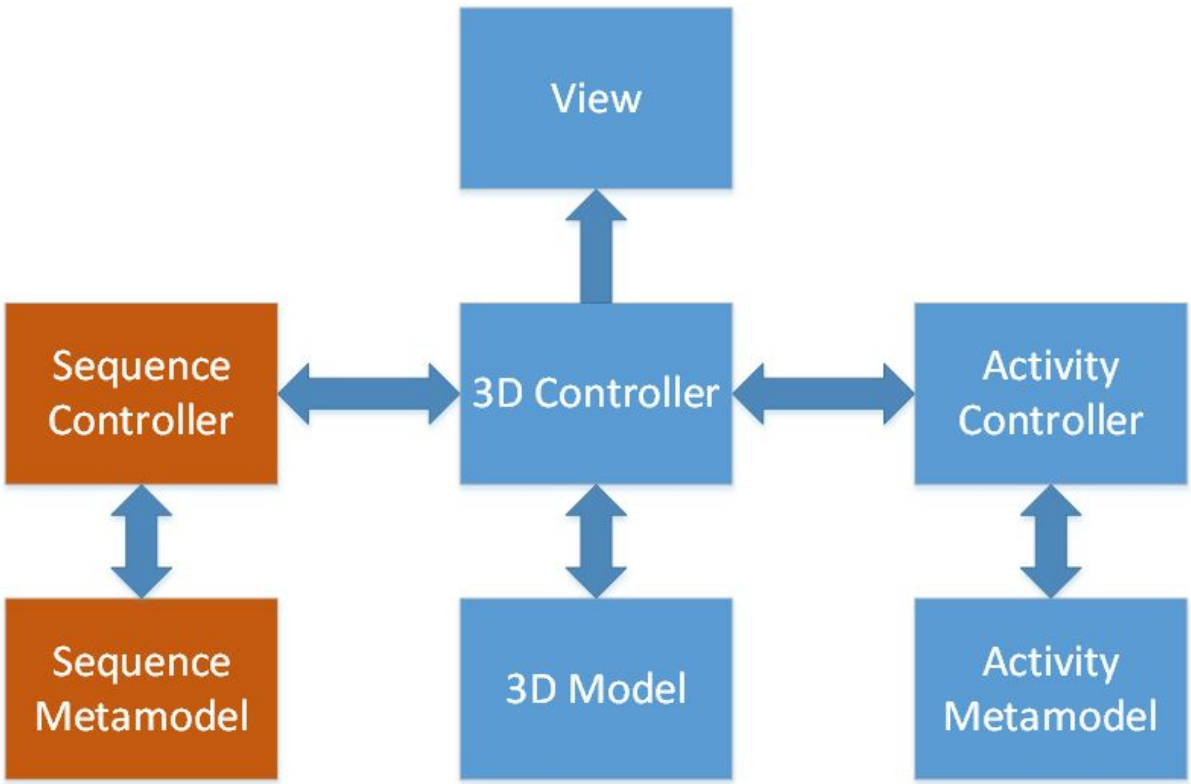
3.1.2 Analýza prototypov sekvenčného diagramu

Architektúra sekvenčného diagramu je veľmi podobná aktivity diagramu. Narozdiel od aktivity diagramu používa klasickú architektúru MVC. Sekvenčný diagram obsahuje jeden kontroler, ktorý pracuje s model komponentom definovaným v `Core` ako triedy dediace od triedy `Element`. Informácia o štruktúre diagramu sa narozdiel od diagramu aktivít neuchováva v metamodeli ale v elementoch jednotlivých tried modelu. Triedy dediace od `ElementFactory` vytvárajú elementy z jednotného modelu. Analyzované procesy ako vykresľovanie prvkov, prechádzanie stavov sú identické s diagramom aktivít.

3.2 Návrh

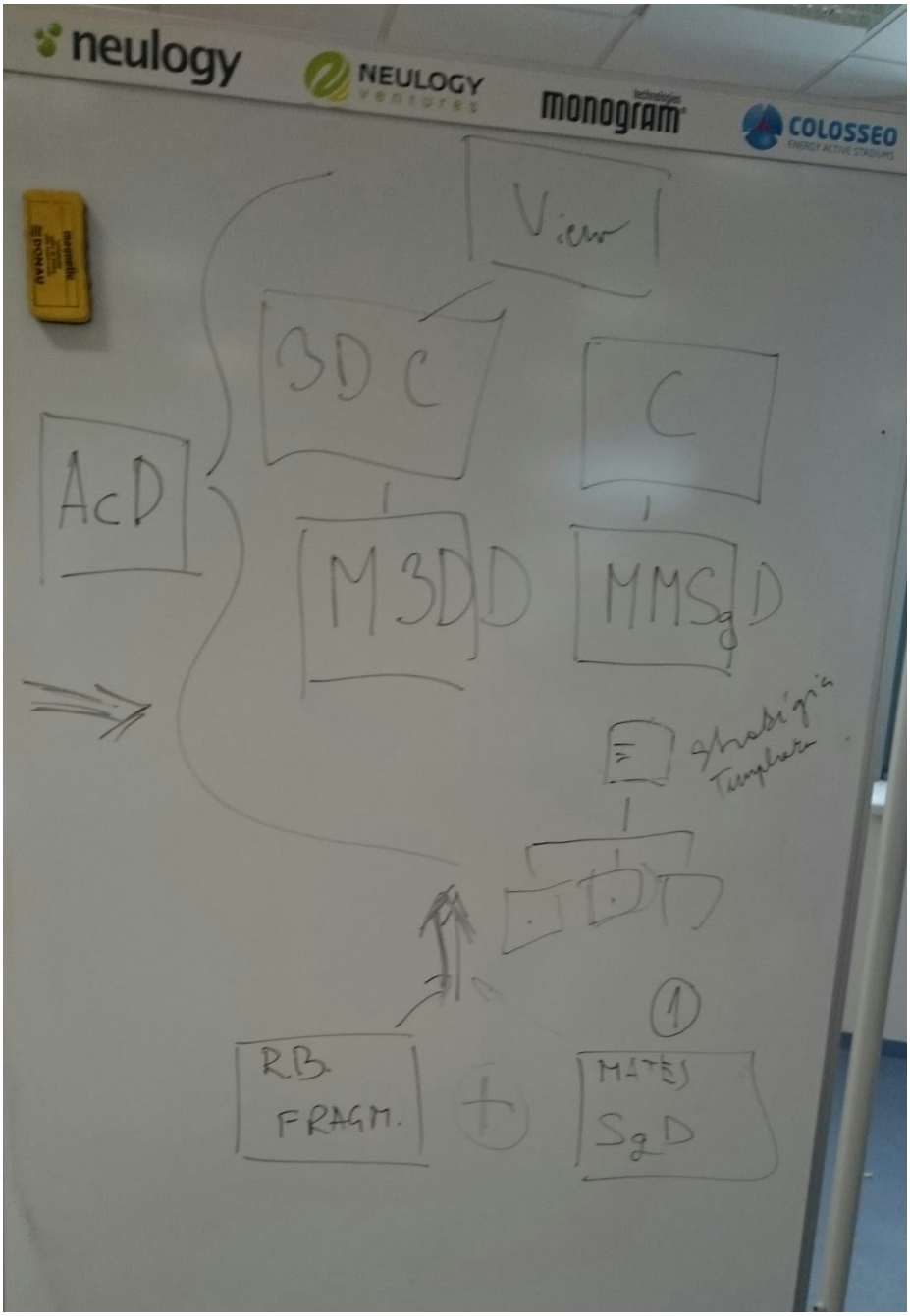
Pri analyzovaní diagramov bola zistená základná funkcionalita aplikačných prototypov. Nasledoval proces spájania prototypov. Prvotným princípom bolo vytvorenie spoločnej aplikácie, ktorá by obsahovala funkcionalitu všetkých troch existujúcich prototypov, keďže doteraz fungovali ako samostatné aplikácie.

Využili sme existujúcu implementáciu návrhu novej architektúry diagramu aktivít a snažili sa navrhnúť najefektívnejšiu cestu integrovania sekvenčného diagramu do tejto architektúry.
Keďže pre tvorbu pôvodného sekvenčného diagramu bola použitá architektúra MVC, bolo potrebné túto architektúru zmeniť. S pomocou vedúceho projektu sme navrhli migráciu sekvenčného diagramu do novej architekúty podľa obrázku 9.
Na obrázku je vidieť architektúra diagramu aktivít spolu s pridanými komponentami pre sekvenčný diagram. Pozostáva z jedného view komponentu ktorý slúži na zobrazovanie jednotlivých elementov na obrazovku, 3D controlleri slúži na uchovávanie stavov v aplikácií a vytváranie elementov diagramu. V 3D modely sa nachádzajú všetky prvky ktoré popisujú ako by mali vyzerať jednotlivé elementy ako vrstvy, aktivity a podobne. V štandardnom controlleri sa vytvárajú metamodel objekty definované v metamodely pre sekvenčný diagram. Tie určujú vzťahy medzi komponetami sekvenčného diagramu.
V našom návrhu implementácie vytvárame do existujúcej architektúri aktivity prototypu prídavné moduly `SequenceController` a `SequenceModel`. Pre tieto komponenty budeme používať vytvorené triedy v prototypoch sekvenčných diagramov. V prípade nájdenia nedostatkov v architektúre odkonzultujeme s produktovým vlastníkom refaktorizáciu.



Obr. 9: Cieľová architektúra výsledného prototypu

Obrázok nižšie, ktorý sme vytvorili počas brainstormingu na stretnutí uvádza náhľad na štandartné pripojenie controller a metamodel komponentu k už existujúcemu aktivity diagramu.

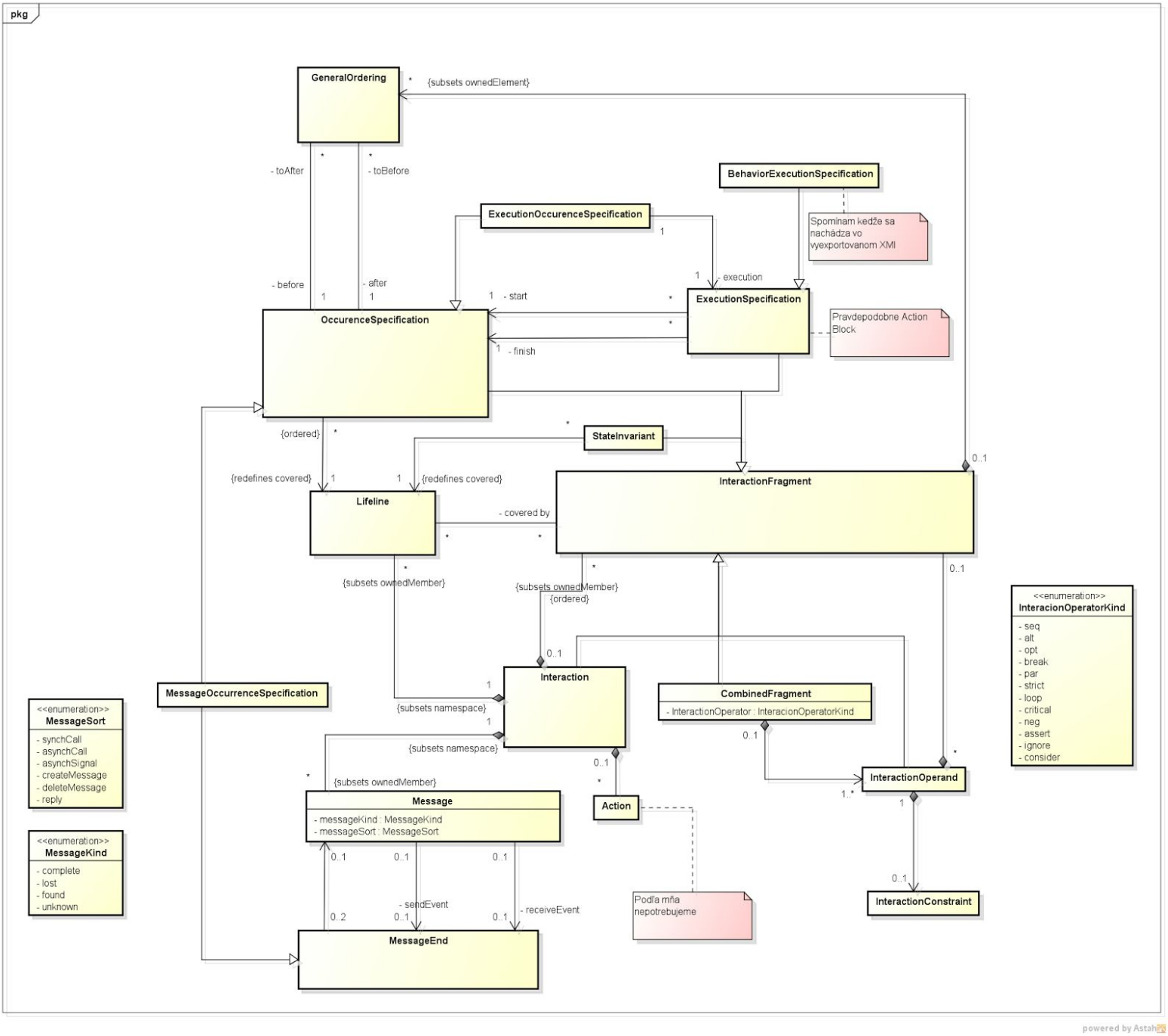


Obr. 10: Náčrt architektúry prototypu diagramu aktivít a spôsobu naviazania zvyšných protypov

3.2.1 Návrh pridávaného Metamodelu sekvenčného diagramu

Ako je spomenuté vyššie metamodel určuje základnú štruktúru prepojení elementov v prototype. Je odvodený od metamodelu sekvenčného diagramu v UML

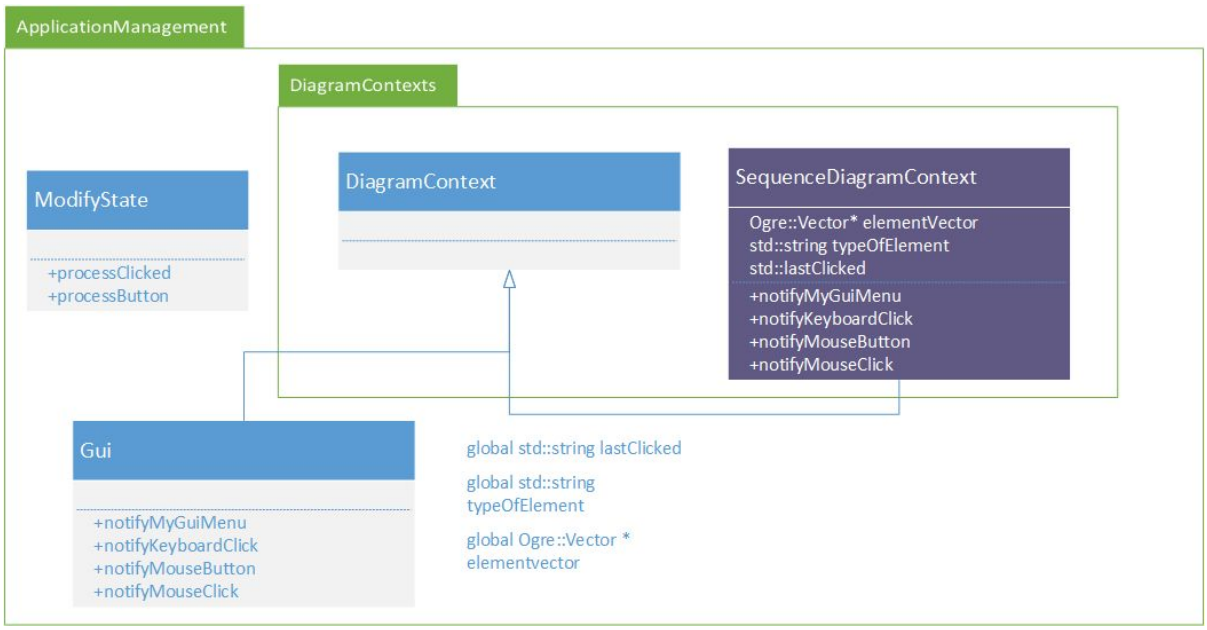
Superstructure. Slúži na ukladanie údajov zo scény aby sme k nim vedeli správne pristupovať a korektne s nimi pracovať. Neskôr môže slúžiť taktiež na import a export údajov z/do prototypu. Jeho štruktúru zobrazuje diagram Y.



Obr. 11: Metamodel sekvenčného diagramu

3.2.2 Rozšírenie časti 3D controller

Pre rozšírenie komponentu bolo potrebné vytvoriť novú triedu `SequenceDiagramContext`. Jej názov určuje, že sa jedná o kontext sekvenčného diagramu. Na rozdiel od aktivity časti view sa nevytvárajú žiadne globálne premenné. Všetky sú definované ako premenné v triede `SequenceDiagramContext`. V prípade práce so sekvenčným UML diagramom sa volajú metódy `SequenceDiagramContext`-u.



Obr. 12: Návrh časti rozšírenia 3D controlleru

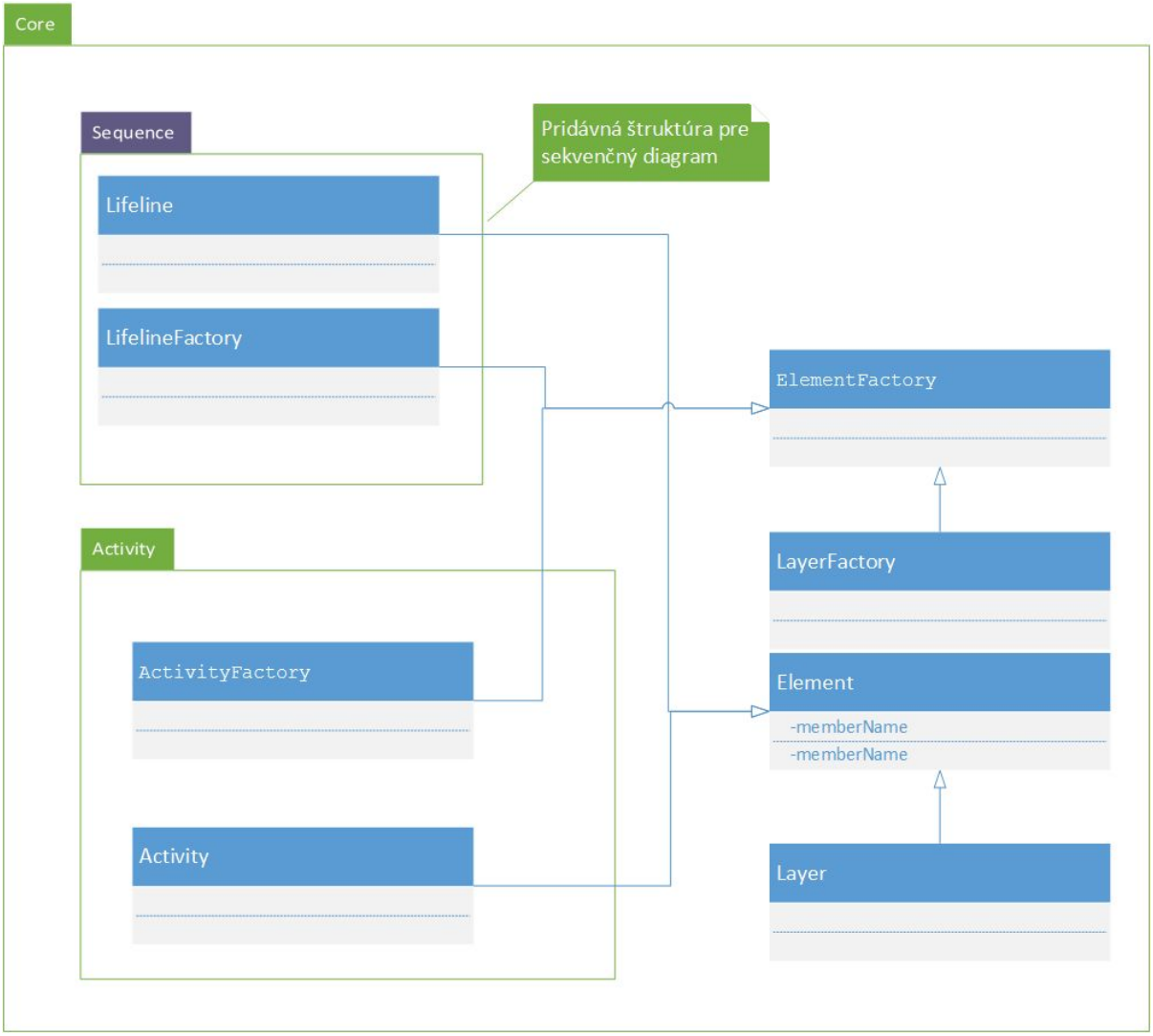
3.2.3 Rozšírenie komponentu 3D model

V časti model je potrebné pridávať elementy sekvenčného diagramu. Kedže metamodel určuje vzťahy medzi komponentami sekvenčného diagramu model bude rozšírený o triedy sekvenčného diagramu. Tieto triedy budú definovať údaje pre jednotlivé komponenty diagramu. Triedy budú dediť vlastnosti od triedy `Element` (obrázok 13). Pre veľkú zložitosť na diagrame sú uvedené len príklady tried pre pochopenie princípu dopĺňanej štruktúri.

3.2.4 Návrh pridávanej kontrolnej jednotky

Pridávanú kontrolnú jednotku (controller) pre sekvenčný diagram budú tvoriť triedy, ktoré vytvárajú metamodel elementy sekvenčného diagramu. Taktiež prideľujú vykreslovacie algoritmy a grafické informácie. Triedy sú definované v štruktúre `Core/Sequence`. Trieda `ElementFactory` vytvára element, ktorý obsahuje pointer na vytvorený metamodel element. Ten je vytvorený factory triedou sekvenčného

diagramu. Na obrázku nižšie (obrázok 13) je uvedená štruktúra s príkladmi factory tried vo forme diagramu tried. Pre veľkú zložitosť na diagrame sú uvedené len príklady tried pre pochopenie princípu. Prídavný controller sa bude nachádzať v štruktúre `Core/Sequence`.

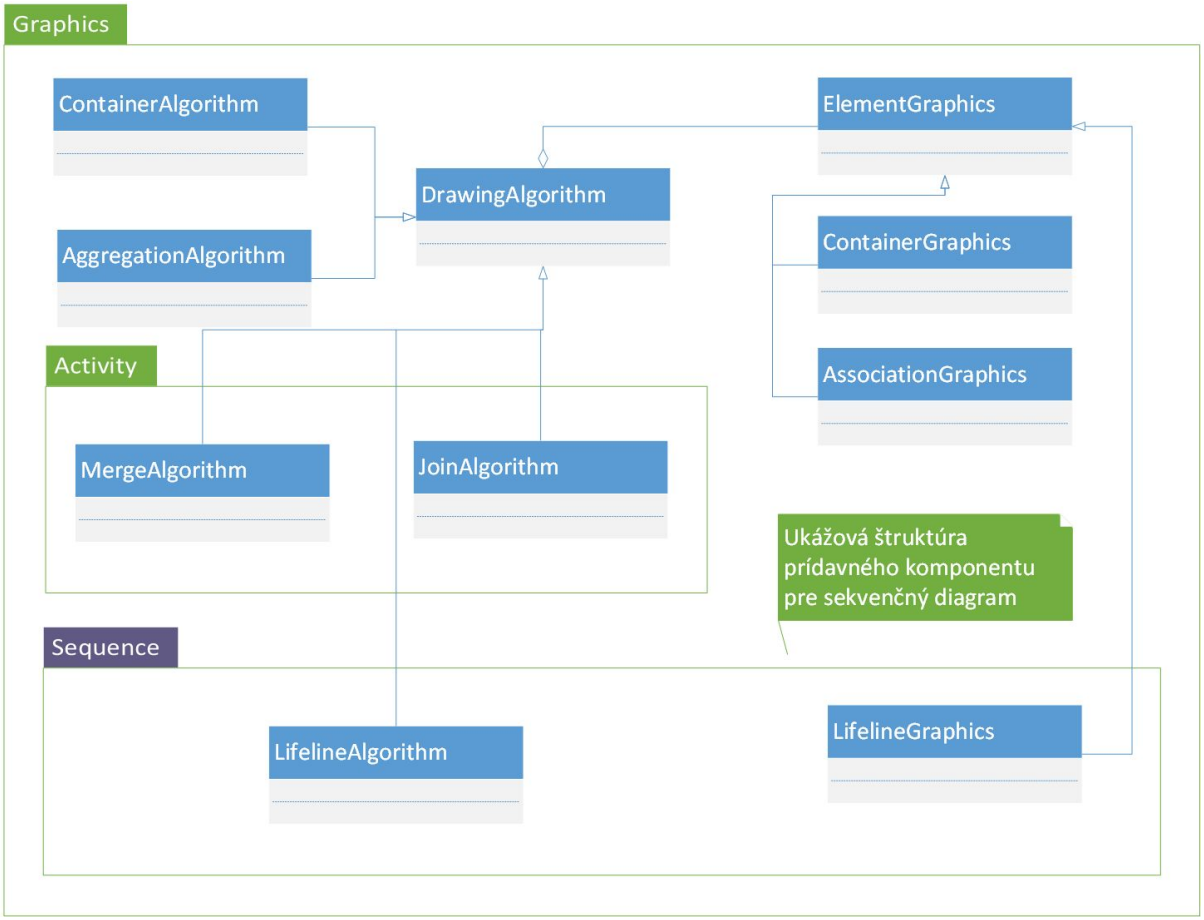


Obr. 13: Diagram tried pre `Factory` triedy

3.2.5 Modifikácia 3D View komponetu

Súčasťou View komponentu sú triedy dediace od triedy `DrawingAlgorithm`. Určujú spôsob vykreslenia tvaru elementu. V nich sú agregované informácie ohľadom grafického spôsobu vykreslenia (napr. použitie typu písma, šírka textu) vo forme tried dediacich od triedy `ElementGraphics`. V prototype je potrebné postupne pridávať pre každý element vlastné triedy pre vykreslenie implementovaných elementov. Tie sa búdú vkladať do štrukúry

`Graphics/sequence`. Na obrázku 14 je uvedený príklad, kde je možné vidieť štruktúru jednotlivých vykresľovacích tried a grafických informácií. Kôli veľkej zložitosti je diagram tried len ukážkový a neobsahuje všetky triedy zahrnuté v štruktúrach.



Obr. 14: Diagram tried algoritmov

3.3 Implementácia

Vývoj cieľového produktu je vytváraný metódou modifikovaného SCRUM-u. Používa sa agilný vývoj a vytvárajú sa funkčné prototypy, ktoré spĺňajú jednotlivé položky z produktového backlogu.

Našim hlavným cieľom na zimný semester bolo vytvoriť prototyp sekvenčného diagramu s dodržaním architektúry vytvorenej minuloročným tímom a základnou funkcionalitou vkladania elementov do diagramu. Do prototypu sme ako prvé natiahli metamodel sekvenčného diagramu, ktorý je miernou úpravou metamodelu definovaného v UML Superstructure. Jeho štruktúra je zobrazená na obrázku 11. Metamodelu bol priradený samostatný balíček v prototype, podobne ako tomu je aj v metamodeli aktivity diagramu. Na základe definovaných prepojení medzi triedami metamodelu v diagrame sme vedeli definovať vzťahy agregácie či dedenia aj

implementačne. Implementáciou metamodelu sme si vytvorili podmienky pre začiatok grafickej a logickej implementácie jednotlivých scenárov.
Ako sme spomenuli, pre nás dôležitým používateľským príbehom je vloženie elementu do scény. Tento proces bol však vzhľadom na rozsiahlosť prototypu príliš komplexný a preto sme ho rozdelili na niekoľko podúloh, ktorých priebeh a implementácia sú opísané nižšie.

3.3.1 Algotimus vykreslenia grafickej plochy s nefunkčným menu

Logicky prvým krokom, ktorý používateľ po spustení aplikácie vykoná je zvolenie typu diagramu, s ktorým chce pracovať. Naši predchodcovia vytvorili prostredie, v ktorom je možnosť výberu sekvenčného diagramu priamo poskytnutá, no nefunkčná. Na tento fakt sme nadviazali a tlačidlo sekvenčného diagramu sme oživili. Prvým krokom bolo vytvorenie prázdnej scény po kliknutí na tlačidlo. Nakoľko vytvorenie kompletnej scény je implementované v “aktivity” časti prototypu, jediné čo bolo treba zmeniť, boli jednotlivé elementy aktivity diagramu, za elementy sekvenčného diagramu. V menu sa teda nezobrazia elementy ako aktivity ale budú nahradené napríklad čiarou života a správou. Za týmto účelom sme vytvorili XML súbory, v ktorých definujeme vlastnosti tlačidiel úvodného menu. V týchto súboroch sme definovali pozíciu, veľkosť a názov tlačidiel pre naše čiary života, správy a vrstvy. Rovnako sme týmto tlačidlám určili nadelement, v ktorom sú obsiahnuté. Pre všetky tieto tlačidlá to bolo menu v pravom hornom rohu obrazovky. Pri inicializácií scény tieto XML súbory spracujeme a jednotlivé tlačidlá si uložíme do vhodných dátových štruktúr a priradíme im controlleri. Po vykonaných spomenutých činností máme vykreslenú prázdnu scénu, v ktorej pravom hornom rohu sa nachádza menu obsahujúce nefunkčné tlačidlá elementov sekvenčného diagramu.

3.3.2 Algoritmus vykreslenia vrstiev

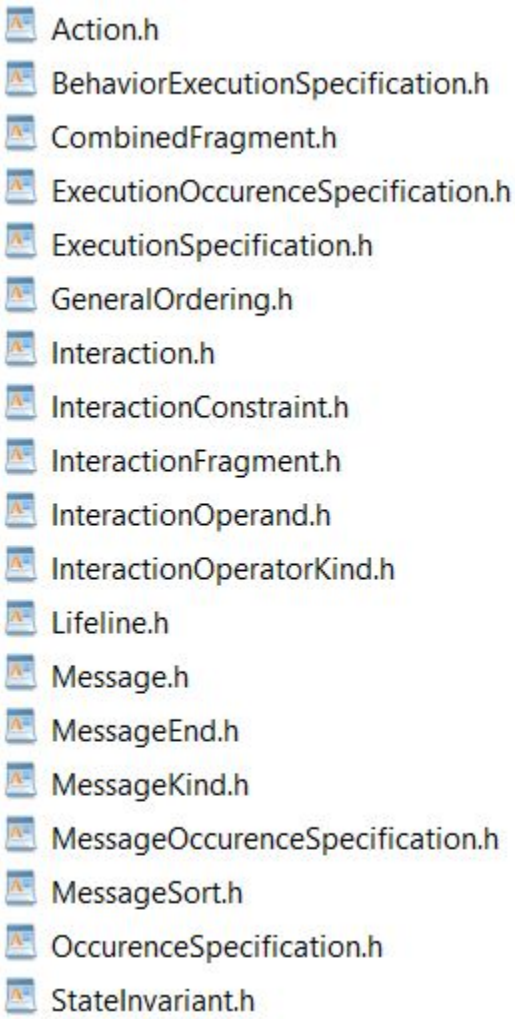
Vykreslenie vrstiev, pridávanie ďalších vrstiev je v prototype korektne vytvorené. Rozhodli sme sa používať už vytvorené komponenty prototypu diagramu aktivít. Vykreslenie jednotlivých vrstiev sme implementovali podľa vzoru triedy `Gui`, ktorá inicializuje vykreslenie vrstiev pre aktivity diagram. Nasledujúci obrázok (obrázok 15) zobrazuje reálnu implementačnú časť, ktorá zodpovedá za vykreslenie prázdnej scény. Pohyb medzi vrstvami a prepínanie aktuálnej scény bolo dotvorené podľa implementácií v triede `Gui`.

```
242     if (_sender->getName() == "button_layer")
243     {
244         lay1 = dataM->createLayer();
245     }
```

Obr. 15: Volanie vykresľovacieho algoritmu

3.3.3 Prototyp s implementovaným Modelom komponentom

Po navrhnutí `Metamodel` komponentu pre sekvenčný diagram sme implemetovali, do štruktúry `Core`/`SequenceMetamodel`/, určené triedy a ich dediacu štruktúru. Kompozícia bola dotváraná pomocou štruktúry `std::map<std::string, Trieda*>`. Agregácia je zabezpečená pomocou ukazovateľa na objekt. Na obrázku nižšie sú zobrazené hlavičkové súbory vytváraných tried.



Obr. 16: Hlavičkové súbory vytváraných tried

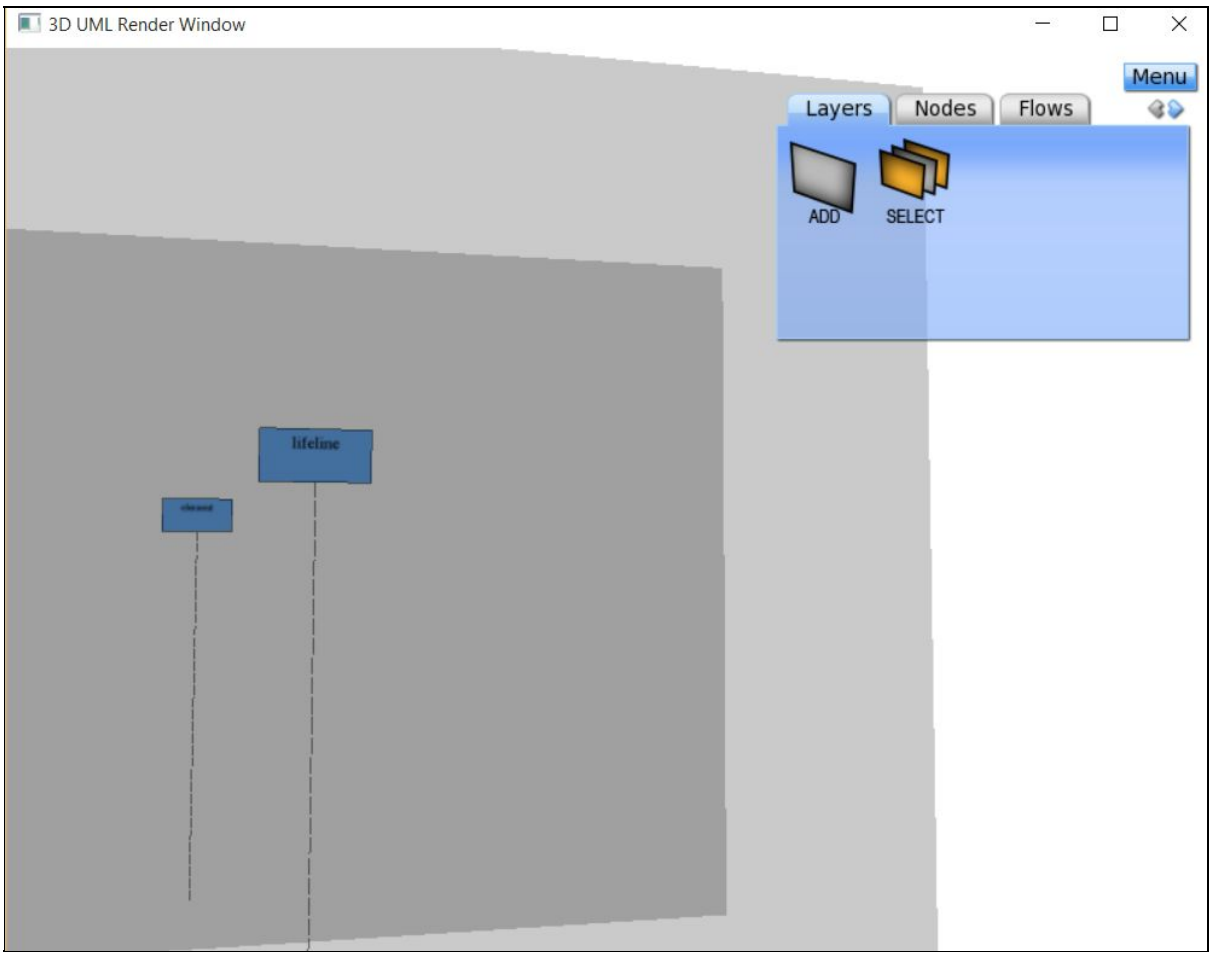
3.3.4 Algoritmus vloženia a vykreslenia Lifeline komponentu

Pre vloženie `Lifeline` komponentu bolo potrebné modifikovať časti `3D controller-a`, `3D model-u`, a `View` komponentu. Následne boli vytvárané potrebné triedy pre správne uloženie a následné vykreslenie čiary života. Pre vytvorenie bolo potrebné vykonať nasledujúce akcie:

1. definovať v metamodeli a modeli `Lifeline` triedu.

2. vytvoriť `LifelineFactory` triedu, ktorá by pracovala s a vytvárala metamodelový komponent lifeline a prideľovala grafické informácie.
3. vytvoriť triedu `LifelineGraphics`, ktorá obsahuje informácie ohľadom vykreslenia čiary života.
4. vytvortiť `LifelineAlgorithm`, v ňom je implementovaná metóda `draw(std::string)`, ktorá vykreslí čiaru života. Ten bol vytvorený podľa prototypov sekvenčného diagramu.
5. modifikovať `DataManager`, pridať metódu `createLifeline`, ktorá inicializuje `LifelineFactory` a `ElementFactory`. Tie vytvoria dvojicu `<Element, MetamodelElement>` a uložia záznam do statickej singleton štruktúri v `ElementCollection`.
6. modifikovať `SequenceDiagramContext` aby inicializoval vytváranie komponentu a zobrazenie korektného editovacieho okna.

Výsledné vykreslenie je zobrazené na obrázku 17. Vkladanie je možné na rôzne vrstvy.



Obr. 17: Vykreslenie `Lifeline`

4 Testovanie

Priebežne s implementáciou prebiehalo aj testovanie prototypu. Pre otestovanie implementovaných častí sme si vytvorili niekoľko nižšie uvedených testovacích scenárov. Niektoré scenáre sú určené aj pre funkcionalitu, ktorá mala byť implementovaná už pred našimi zmenami a v prípade, že test nefunguje pre elementy v scéne sekvenčného diagramu, tak nefunguje ani pre ostatné (diagram aktivít elementy).

- **Zobrazenie prázdnej scény pre sekvenčný diagram**
    1. Spustenie aplikácie.
    2. Výber sekvenčného diagramu v menu.
    3. Finálny stav - Vytvorí sá prázdna scéna sekvenčného diagramu.
- **Zobrazenie elementu Lifeline v menu**
    1. Výber sekvenčného diagramu v menu.
    2. Otvorenie záložky “Nodes” v pravom hornom menu, ktoré sa zobrazí po načítaní scény.
    3. Finálny stav - Záložka obsahuje element s názvom “Lifeline”.
- **Načítanie okna pre vytvorenie elementu Lifeline**
    1. Výber sekvenčného diagramu v menu.
    2. Pridanie aspoň jednej vrstvy v záložke “Layers”.
    3. Výber elementu Lifeline zo záložky “Nodes”.
    4. Kliknutie do pridanej vrstvy.
    5. Finálny stav - zobrazí sa okno pre vytvorenie elementu Lifeline.
- **Nenačítanie okna pre vytvorenie elementu** L**ifeline v prípade, že neexistuje ešte žiadna vrstva**
    1. Výber sekvenčného diagramu v menu.
    2. Výber elementu Lifeline zo záložky “Nodes” v Menu.
    3. Kliknutie na akékoľvek miesto na scéne.
    4. Finálny stav - Okno pre vytvorenie Lifeline sa nezobrazí.
- **Pridanie elementu Lifeline do scény**
    1. Výber sekvenčného diagramu v menu.
    2. Pridanie aspoň jednej vrstvy v záložke “Layers”.
    3. Výber elementu Lifeline zo záložky “Nodes”.
    4. Kliknutie do pridanej vrstvy.
    5. Vyplnenie poľa “name” v okne pre pridanie elementu Lifeline.
    6. Kliknutie na “OK” .
    7. Finálny stav - Na vrstve sa zobrazí nový element Lifeline so zvoleným menom.
- **Nepridanie elementu Lifeline do scény mimo vrstvy**
    1. Výber sekvenčného diagramu v menu.

2. Pridanie aspoň jednej vrstvy v záložke “Layers”.
3. Výber elementu Lifeline zo záložky “Nodes”.
4. Kliknutie mimo akejkoľvek pridanej vrstvy.
5. Vyplnenie poľa “name” v okne pre pridanie elementu Lifeline.
6. Kliknutie na “OK”.
7. Finálny stav - element Lifeline sa nevytvorí. Možná alternatíva - Nezobrazí sa už ani okno pre vytvorenie elementu.

- **Pridanie elementu Lifeline do scény do práve zvolenej vrstvy**
  1. Výber sekvenčného diagramu v menu.
  2. Pridanie aspoň jednej vrstvy v záložke “Layers”.
  3. Výber jednej z vrstiev cez menu alebo tlačidlami “R” a “F” (bez výberu sa predvolene použije najnovšie vytvorená vrstva).
  4. Výber elementu Lifeline zo záložky “Nodes”.
  5. Kliknutie do pridanej vrstvy.
  6. Vyplnenie poľa “name” v okne pre pridanie elementu Lifeline.
  7. Kliknutie na “OK” .
  8. Finálny stav - Na vybranej vrstve sa zobrazí nový element Lifeline so zvoleným menom.
- **Zrušenie okna pre vytvorenie elementu lifeline**
  1. Výber sekvenčného diagramu v menu.
  2. Pridanie aspoň jednej vrstvy v záložke “Layers”.
  3. Výber elementu Lifeline zo záložky “Nodes”.
  4. Kliknutie do pridanej vrstvy.
  5. Vyplnenie poľa “name” v okne pre pridanie elementu Lifeline.
  6. Kliknutie na “Cancel”.
  7. Finálny stav - Okno sa zruší a element Lifeline sa nevytvorí.
- **Vymazanie elementu lifeline**
  1. Výber sekvenčného diagramu v menu.
  2. Pridanie aspoň jednej vrstvy v záložke “Layers”.
  3. Výber elementu Lifeline zo záložky “Nodes”.
  4. Kliknutie do pridanej vrstvy.
  5. Vyplnenie poľa “name” v okne pre pridanie elementu Lifeline.
  6. Kliknutie na “OK”.
  7. Kliknutie pravým tlačidlom myši na vytvorený element.
  8. Kliknutie na možnosť “Delete” v novozobrazenom menu.
  9. Finálny stav - element je vymazaný.

4.1 Výsledky testovacích scenárov

V tejto časti sa nachádzajú výsledky z vyššie uvedených testovacích scenárov.
Každý neúspešný testovací scenár má zároveň uvedený aktuálny finálny stav.

#	Názov testovacieho scenára	Stav testu	Poznámka
1.	Zobrazenie prázdnej scény pre sekvenčný diagram	Úspech	Sekvenčná scéna sa úspešne zobrazí
2.	Zobrazenie elementu Lifeline v menu scény	Úspech	Element je zobrazený v menu v záložke “Nodes”
3.	Načítanie okna pre vytvorenie elementu lifeline	Úspech	Okno sa zobrazí a je editovateľné
4.	Nenačítanie okna pre vytvorenie elementu lifeline v prípade, že neexistuje ešte žiadna vrstva	Neúspech	Pri kliknutí do prázdnej scény po vybratí elementu prestane program pracovať (program očakáva aspoň jednu vrstvu). Tento testovací scenár nefunguje pre žiadny element
5.	Pridanie elementu Lifeline do scény	Úspech	Element sa úspešne vykreslí
6.	Nepridanie elementu Lifeline do scény mimo vrstvy	Neúspech	Element sa zobrazí mimo vrstvy. Tento testovací scenár nefunguje pre žiadny element
7.	Pridanie elementu Lifeline do scény do práve zvolenej vrstvy	Úspech	Element sa vloží do práve zvolenej vrstvy
8.	Zrušenie okna pre vytvorenie elementu lifeline	Neúspech	Tlačidlo “Cancel” nefunguje
9.	Vymazanie elementu lifeline	Neúspech	Tlačidlo “Delete” v menu nefunguje. Tento testovací scenár nefunguje pre žiadny element

4.2 Identifikované problémy mimo testovacích scenárov

V tejto časti sa nachádzajú niektoré identifikované problémy, ktoré nie sú pokryté testovacími scenármi.

1. Vypnutie projektu iným spôsobom než kliknutím na tlačidlo “Esc” môže spôsobiť, že program prestane pracovať.
2. Pravým kliknutím do scény by sa malo zobraziť menu obsahujúce funkcie Insert,Modify a Delete. Žiadna z týchto funkcií momentálne nefunguje a prvé dve sa ani nezobrazia.
3. Pri vkladaní sa element síce zobrazí na zvolenej vrstve, ale nemusí sa zobraziť presne na kliknutom mieste.

5 Príručky

5.1 Inštalačná príručka

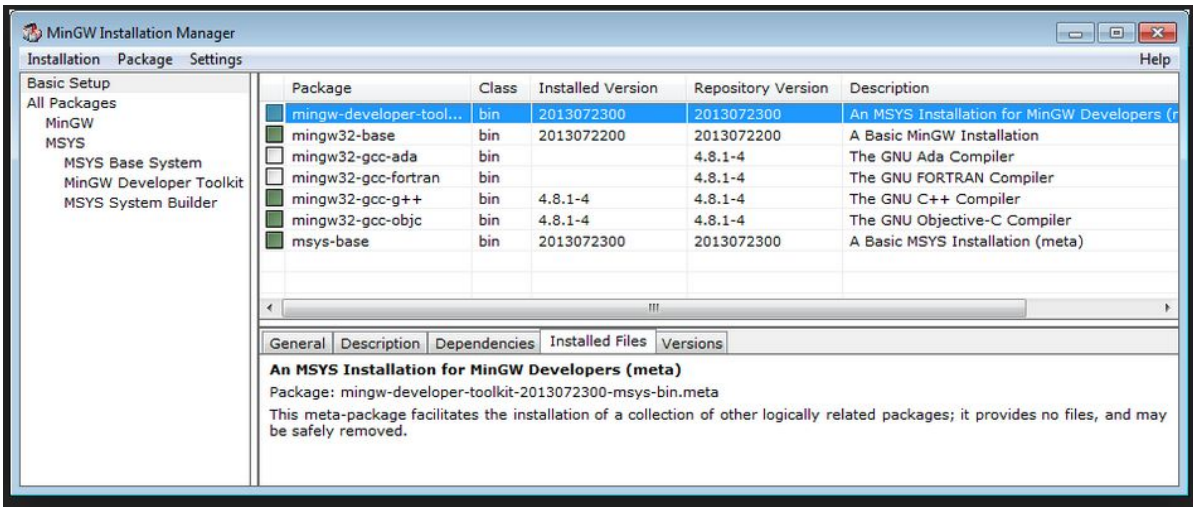
V úvodnej fáze projektu bolo potrebné zistiť aké kroky je nutné pri spúšťaní projektu na vlastnom počítači vykonať. Preto sme vytvorili nasledujúcu inštalačnú príručku.

5.1.1 Inštalácia vývojového prostredia Eclipse pre C/C++

Prijateľná je akákoľvek 32-bitová verzia tohto prostredia. Pri 64-bitovej verzii môžu nastať problémy pri kompilácii 3D UML prototypu a preto sa jej používanie neodporúča.

5.1.2 Inštalácia kompilačného systému MinGW

1. Pri inštalácii je odporúčané vybrať zložku “C:/MinGW”
2. Po dokončení úvodnej inštalácie sa kliknutím na tlačidlo “Continue” otvorí okno s dostupnými balíkmi. Potrebné balíky sú zobrazené na obrázku 18 nižšie (označené zelenou farbou).
3. Potvrdením výberu sa balíky nainštalujú.



Obr. 18: Potrebné balíky pre korektnú inštaláciu MinGW

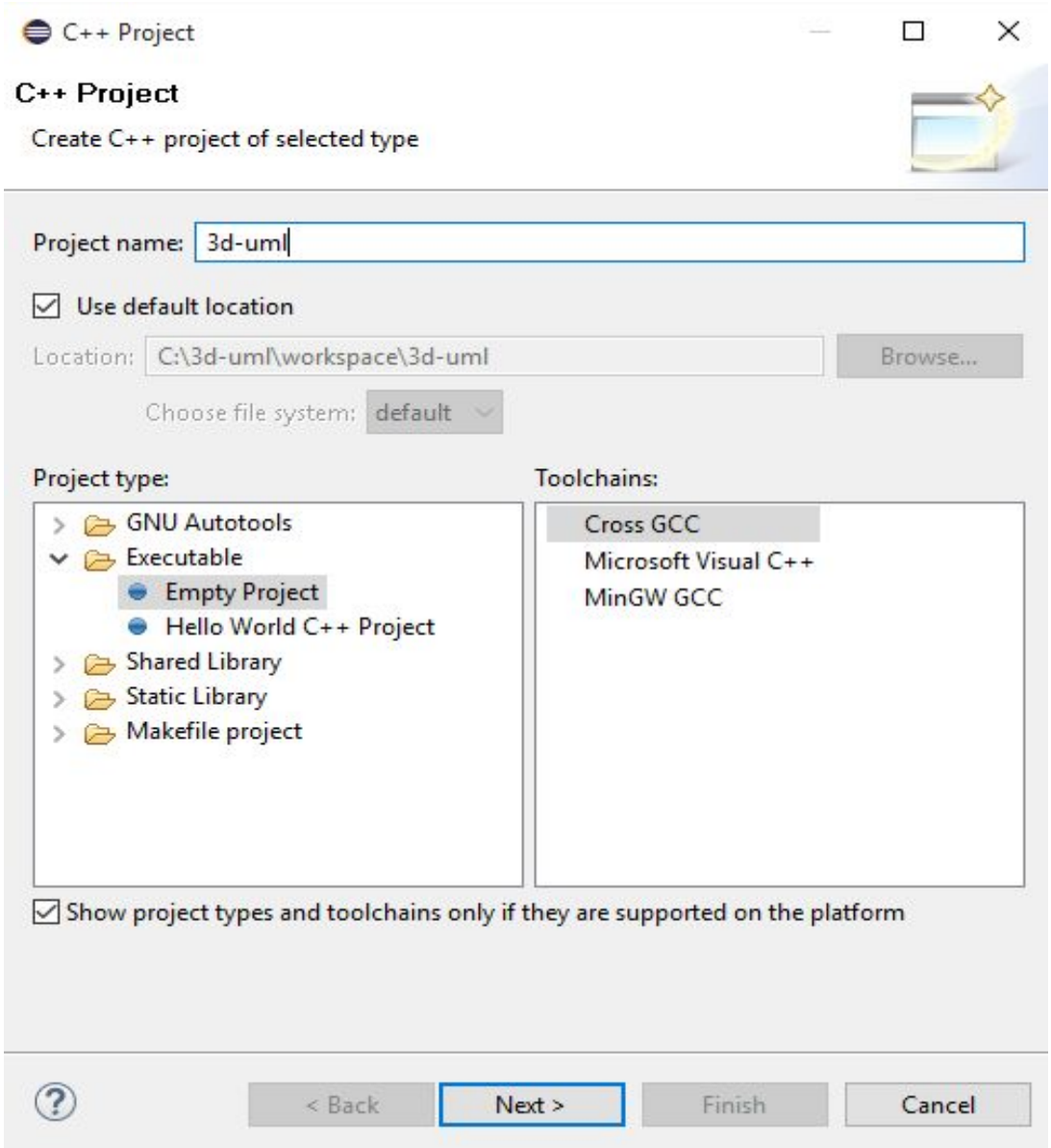
5.1.3 Stiahnutie prototypu z Bitbucketu

1. Najjednoduchší spôsob stiahnutia projektu je využitie open source programu SourceTree.
2. Po nainštalovaní SourceTree je potrebné v Bitbuckete zvoliť konkrétnu vetvu.
3. V novom okne potom stačí len kliknúť na “Check out -> Check out in SourceTree” v pravom hornom rohu. Tým sa spustí SourceTree.
4. V SourceTree je nasledovne potrebné vybrať zložku, kam sa má prototyp stiahnuť. Odporúčaná cesta je “C:\3d-uml”

5.1.4 Úprava nastavení v Eclipse

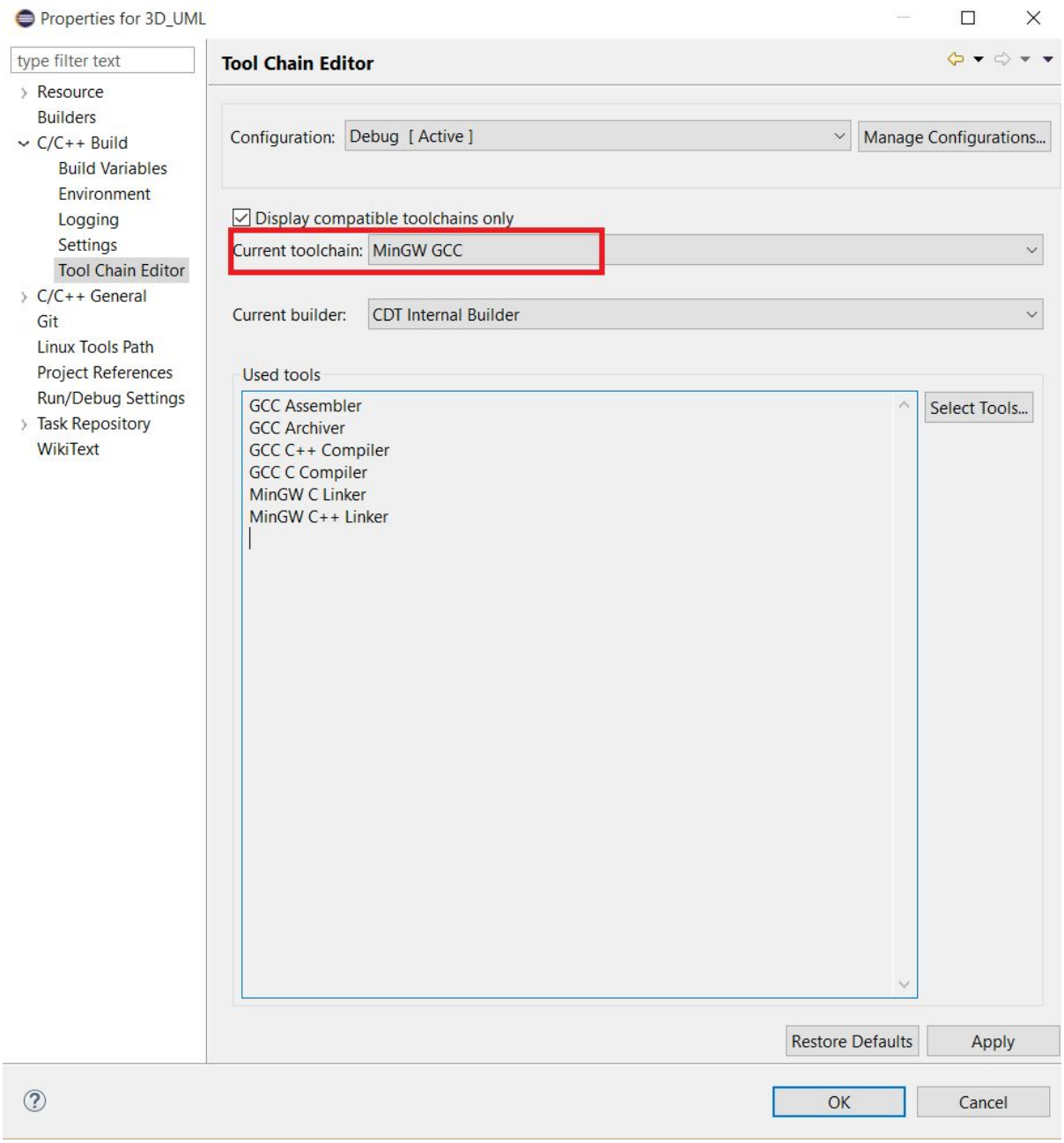
1. Pri spustení Eclipse je potrebné vybrať workspace. V každom prototype sa workspace už nachádza, takže je potrebné ho len nájsť v zložke, kam sa celý prototyp stiahol.
2. V prípade, že sa projekt v Eclipse zobrazí, preskočte na krok 4. V prípade, že po zvolení existujúceho workspacu sa v Eclipse nezobrazí projekt, je potrebné

vytvoriť nový. Vytvorenie projektu je zobrazené na obrázku nižšie.



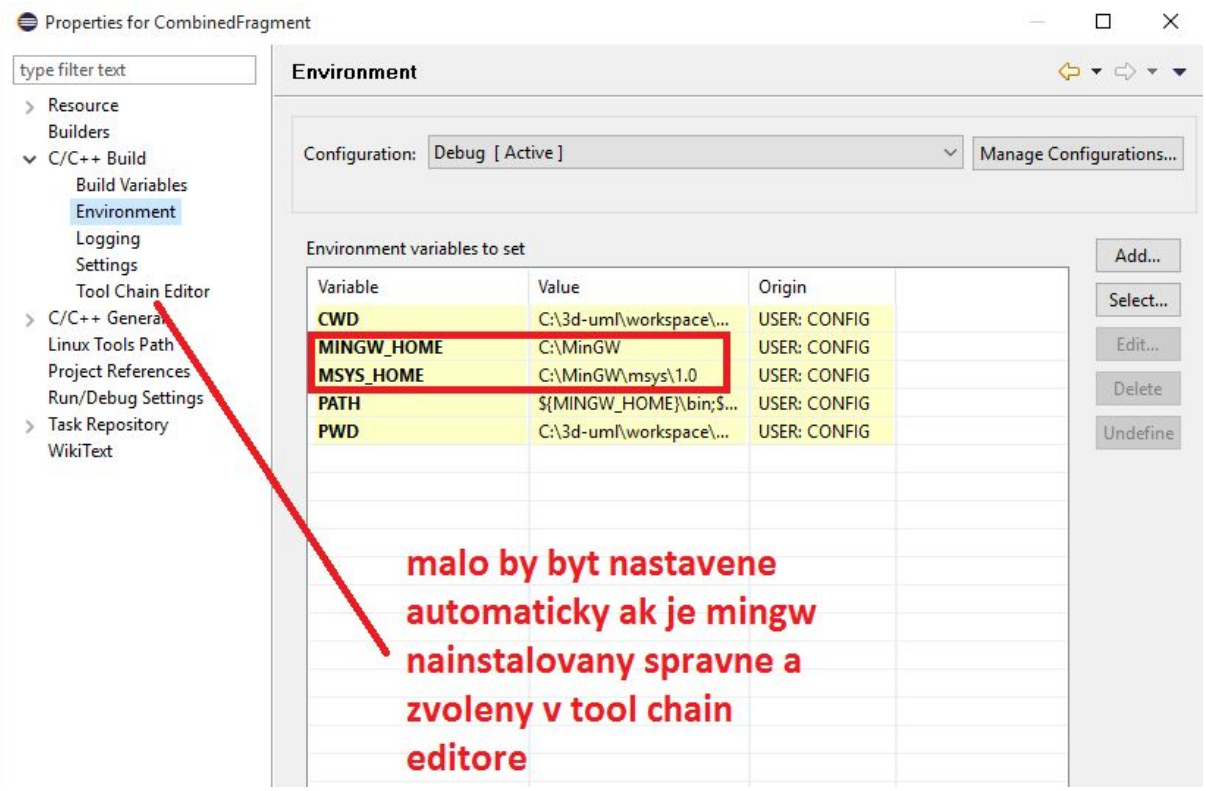
Obr. 19: Ogre setup - krok 1

3. Po vytvorení projektu je potrebné prekopírovať zložky “Debug” a “src” z pôvodného projektu.
4. Nasledovne je potrebné kliknúť pravým tlačidlom na projekt a zvoliť Properties.
5. V časti “C/C++ Build → Tool Chain Editor” je potrebné zmeniť používaný kompilátor na MinGW GCC. Zmena je zobrazená na obrázku nižšie.



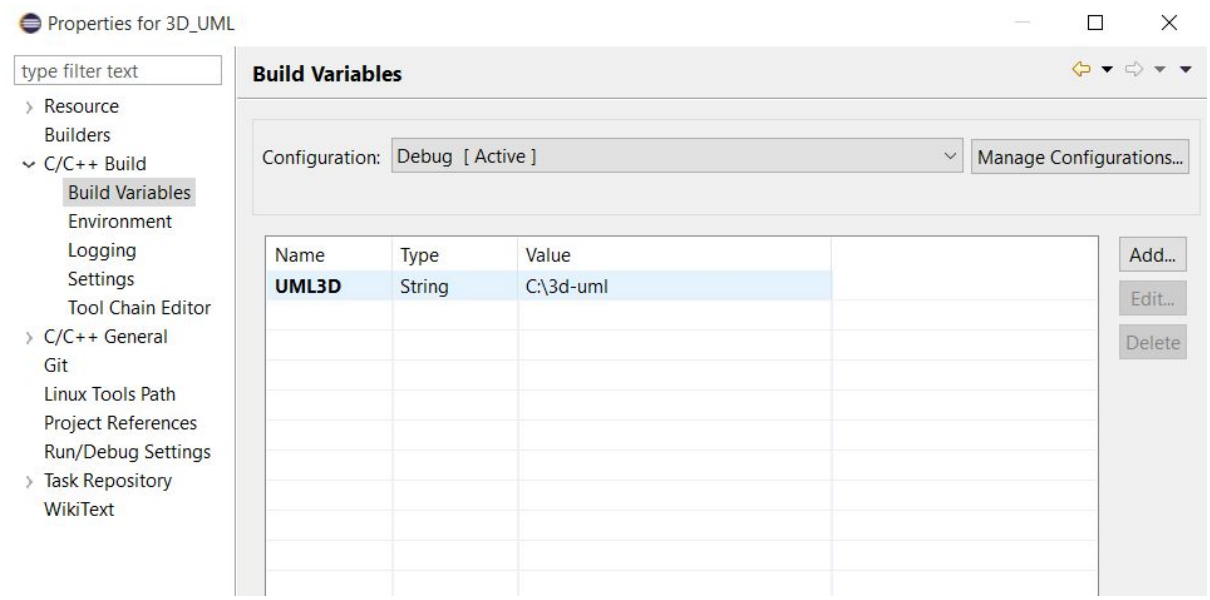
Obr. 20: Ogre setup - krok 2

6. Po zvolení kompilátora by mali byť automaticky v “C/C++ Build → Environment” tieto premenné prostredia. V opačnom prípade je potrebné ich ručne pridať.



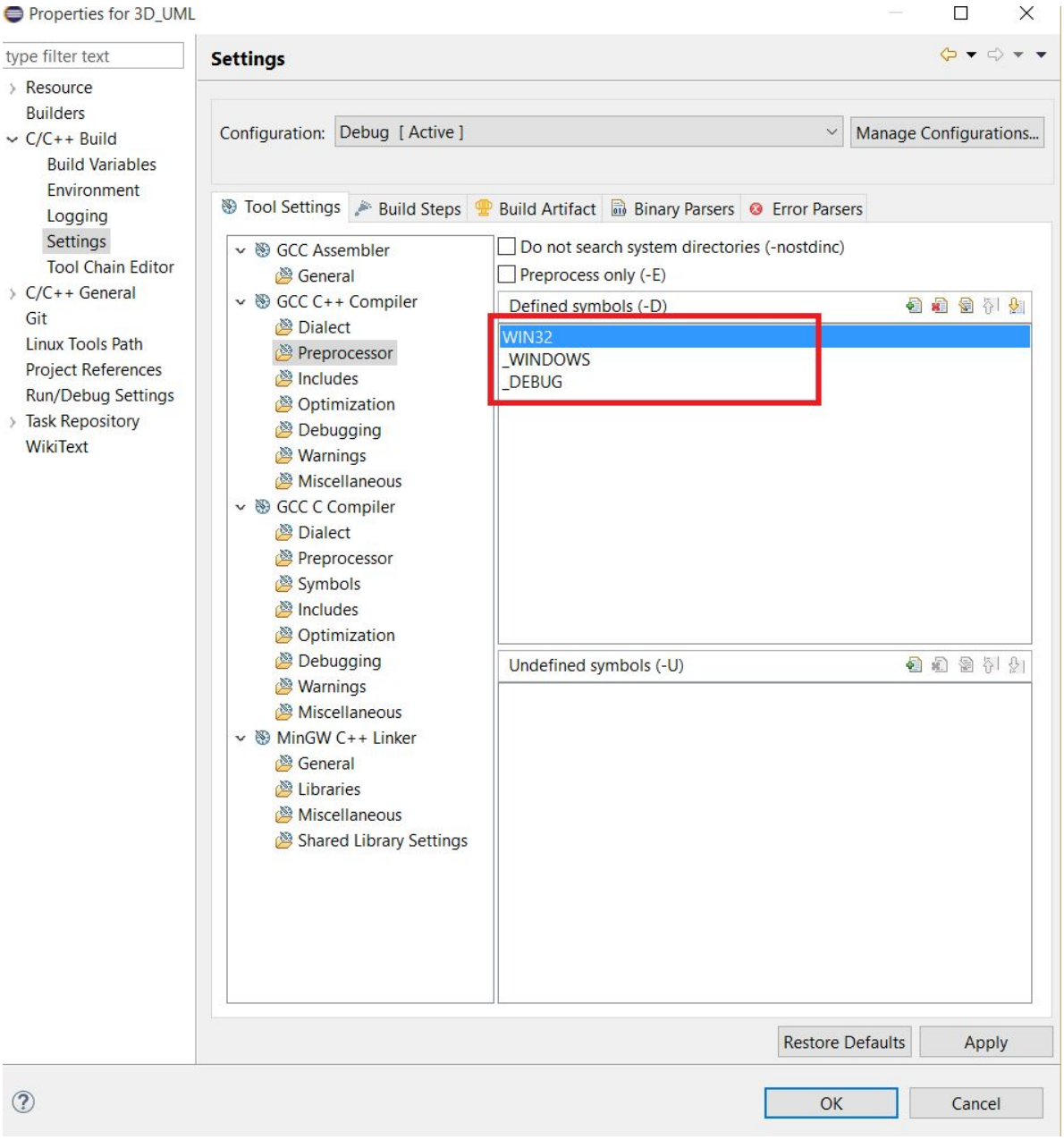
Obr. 21: Ogre setup - krok 3

7. V “C/C++ Build → Build Variables” vytvorte premennú s názvom “UML3D” a ako jej obsah dajte cestu k projektu.

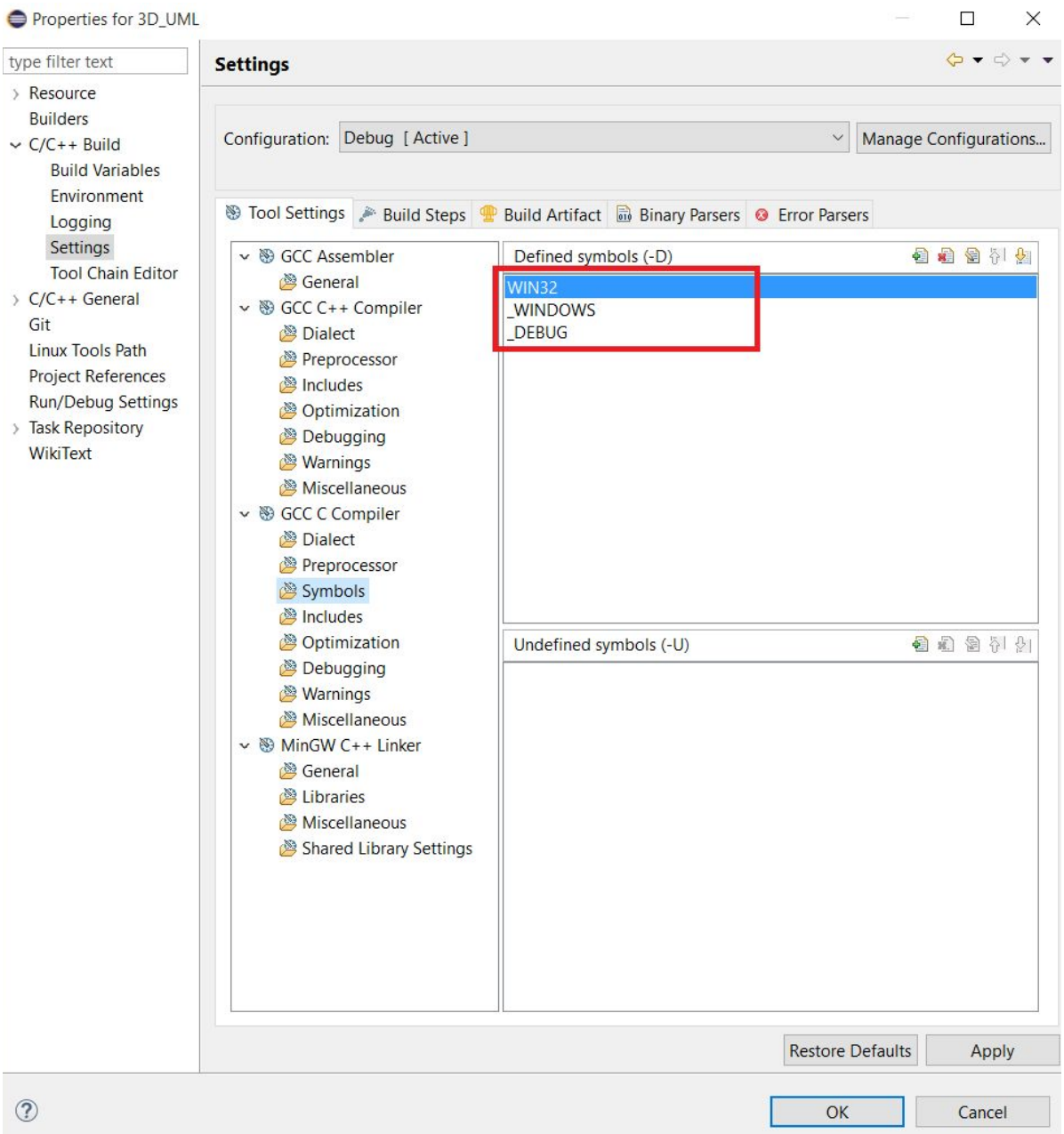


Obr. 22: Ogre setup - krok 4

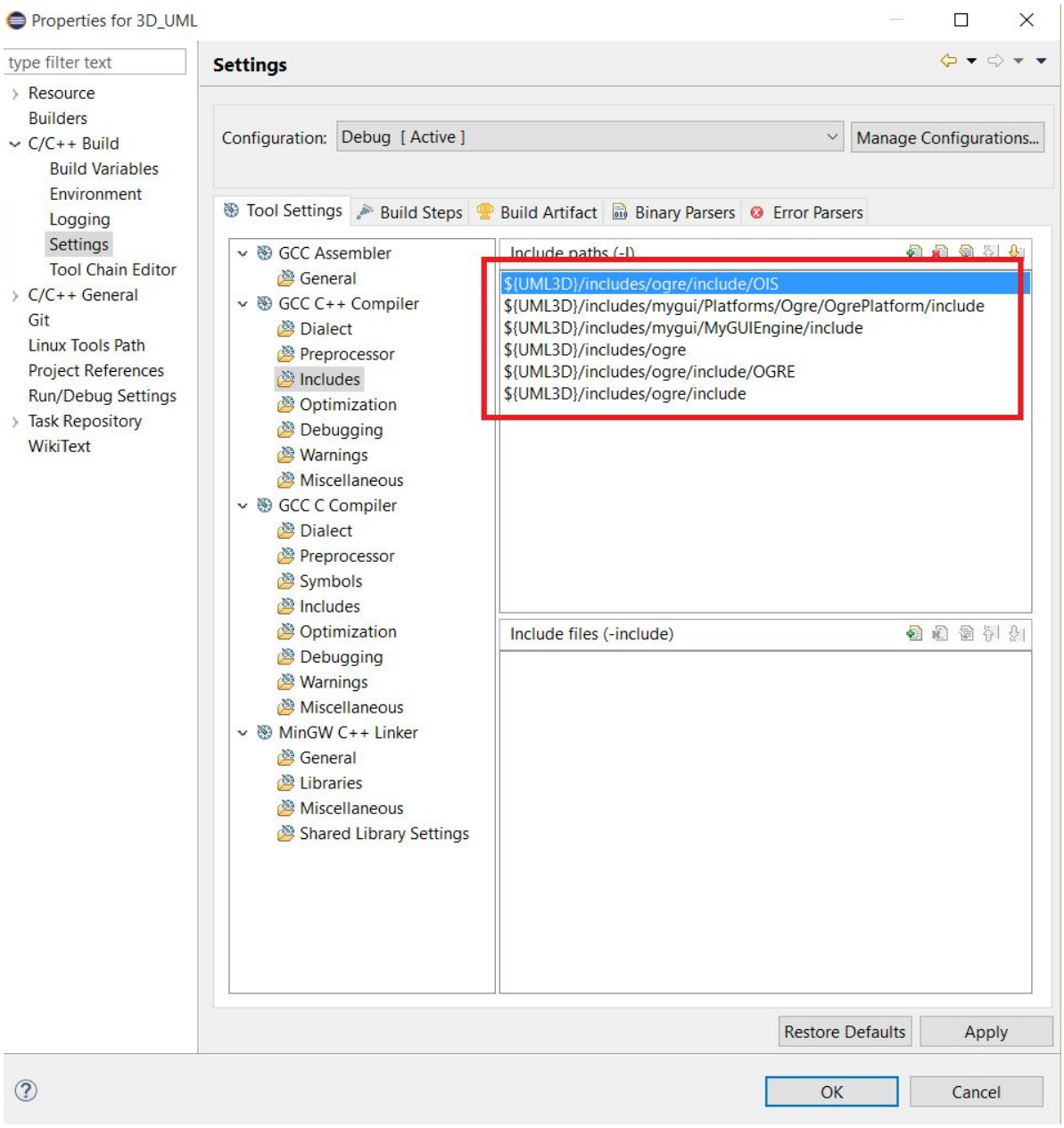
8. V časti “C/C++ Build → Settings” nasledovne vyplňte všetky údaje tak ako je zobrazené na obrázkoch nižšie.



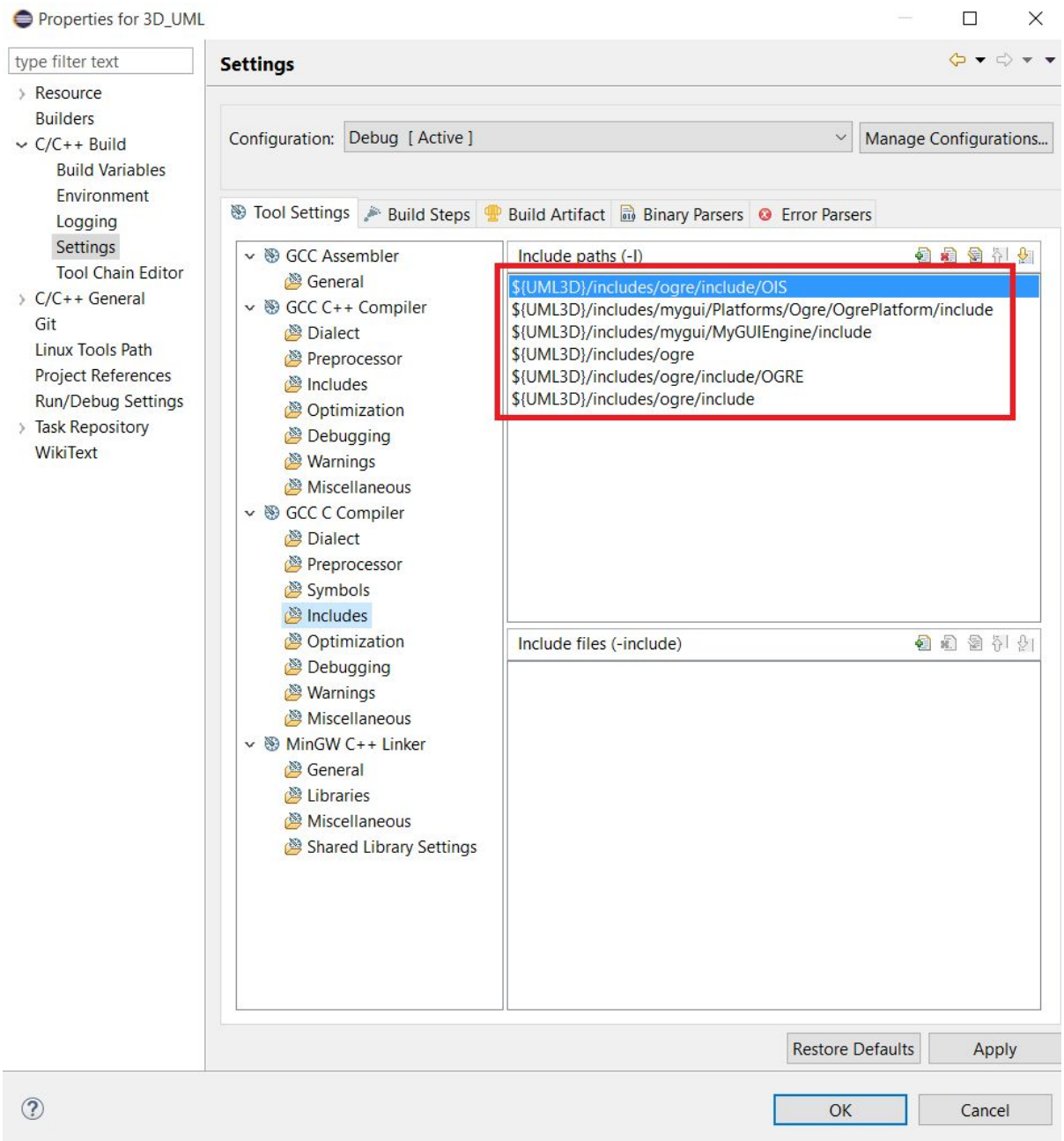
Obr. 23: Ogre setup - krok 5



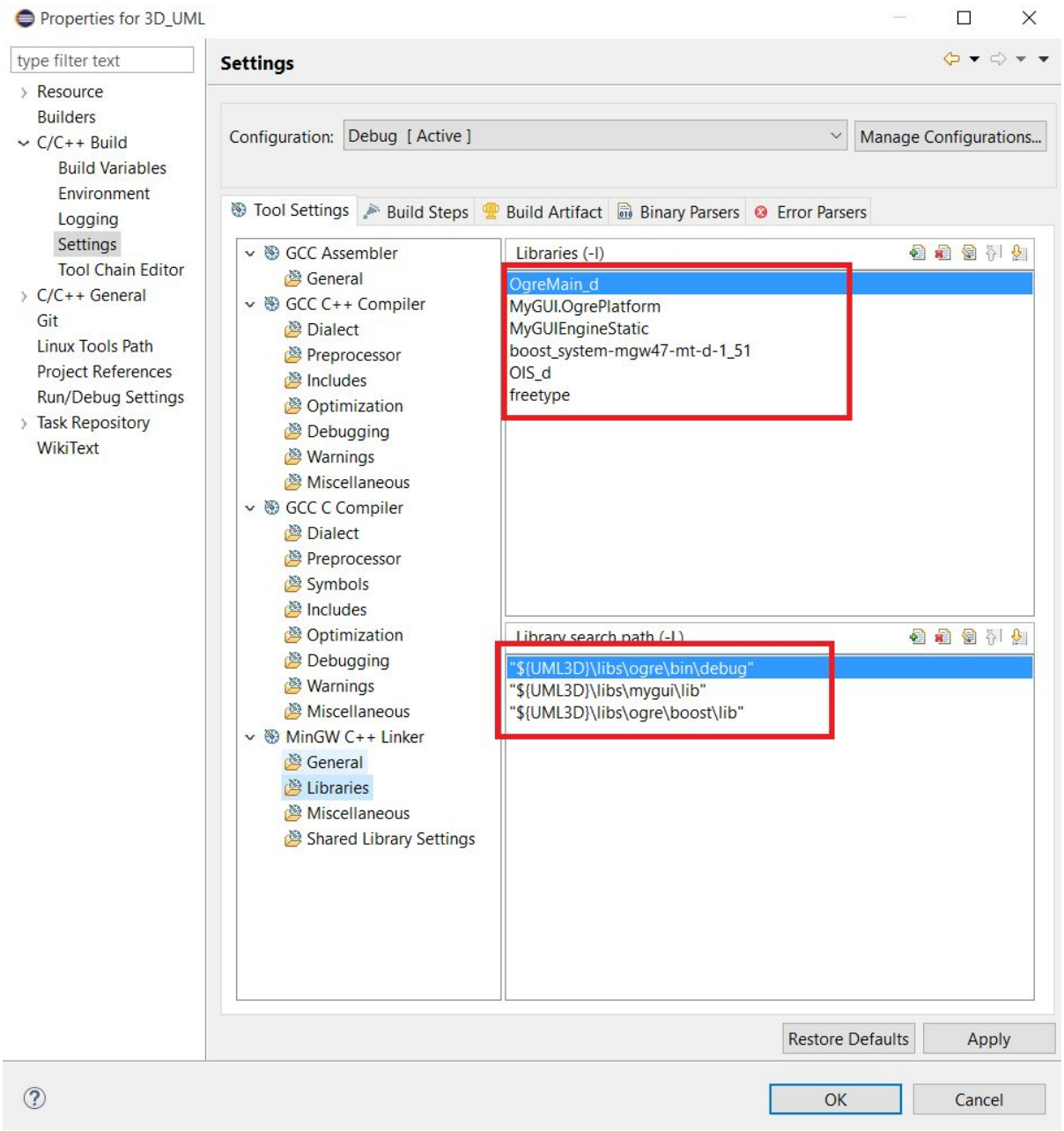
Obr. 24: Ogre setup - krok 6



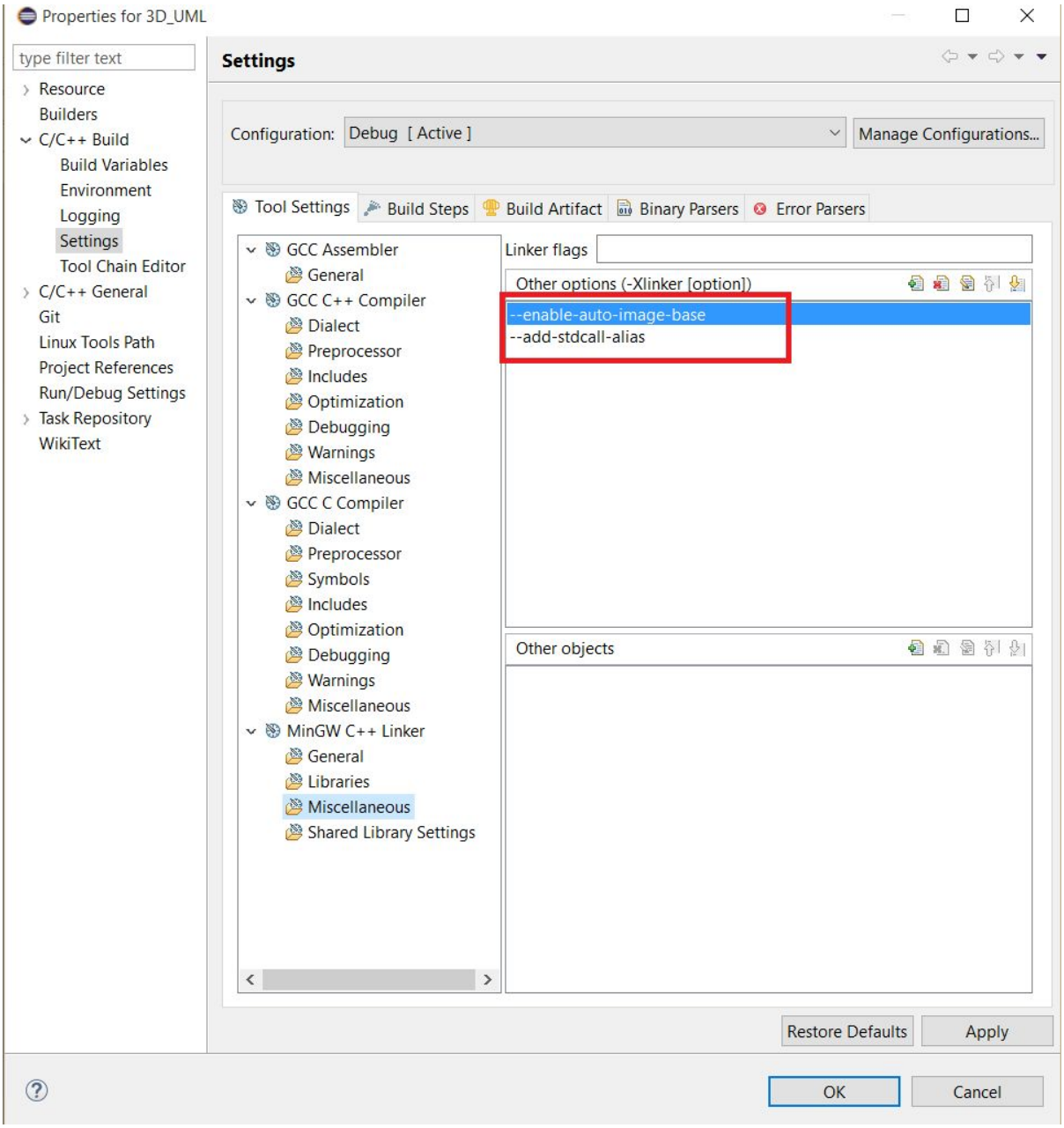
Obr. 25: Ogre setup - krok 7



Obr. 26: Ogre setup - krok 8

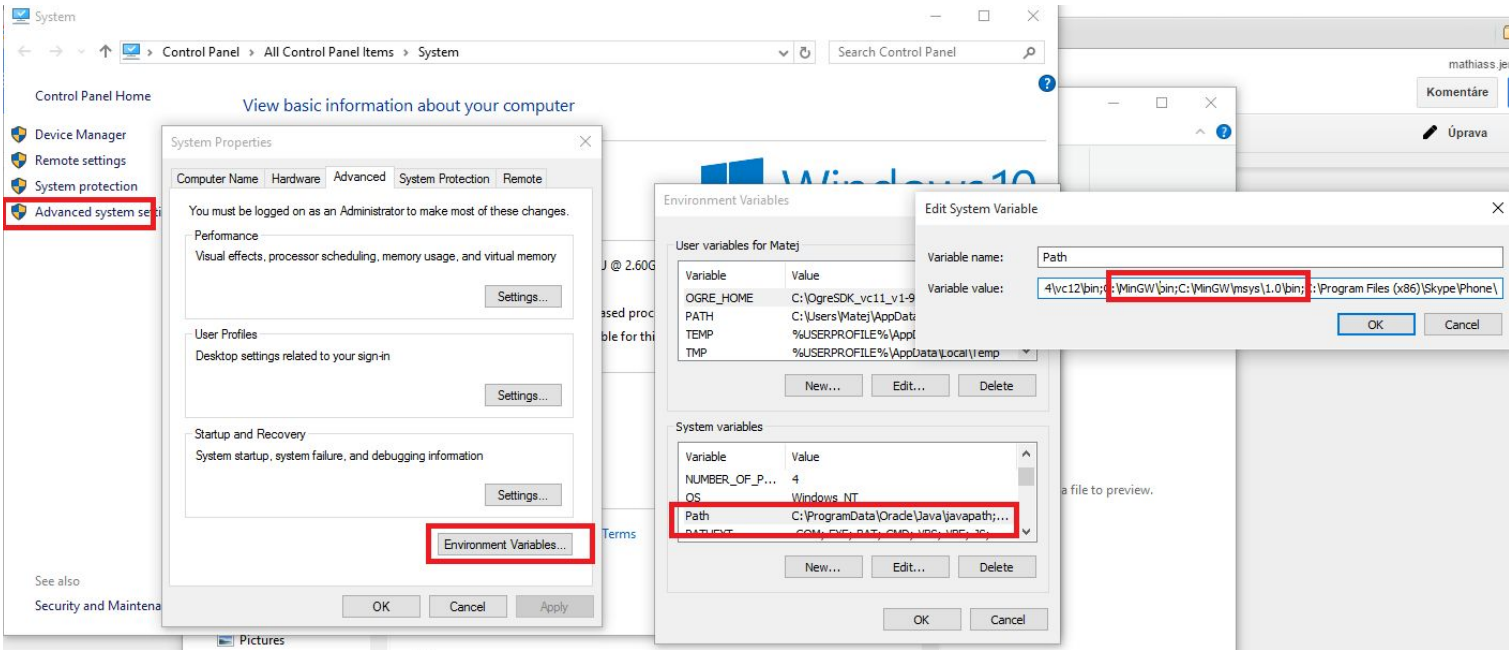


Obr. 27: Ogre setup - krok 9



Obr. 28: Ogre setup - krok 10

9. Nasledovne nastavte systémové premenné tak, ako je znázornené na obrázku nižšie.



Obr 29: Nastavenie systémovej premennej ‘Path’

10. Po nastavení všetkých údajov reštartuje Eclipse. Projekt by mal byť v tomto stave kompilovateľný. Kompilácia môže trvať rádovo niekoľko minút.

Známe problémy

V tejto časti sa nachádzajú problémy, ktoré nastali pri kompilácii / spustení skompilovaného projektu jednému z členov tímu. Na všetky identifikované problémy sa tu nachádzajú aj riešenia.

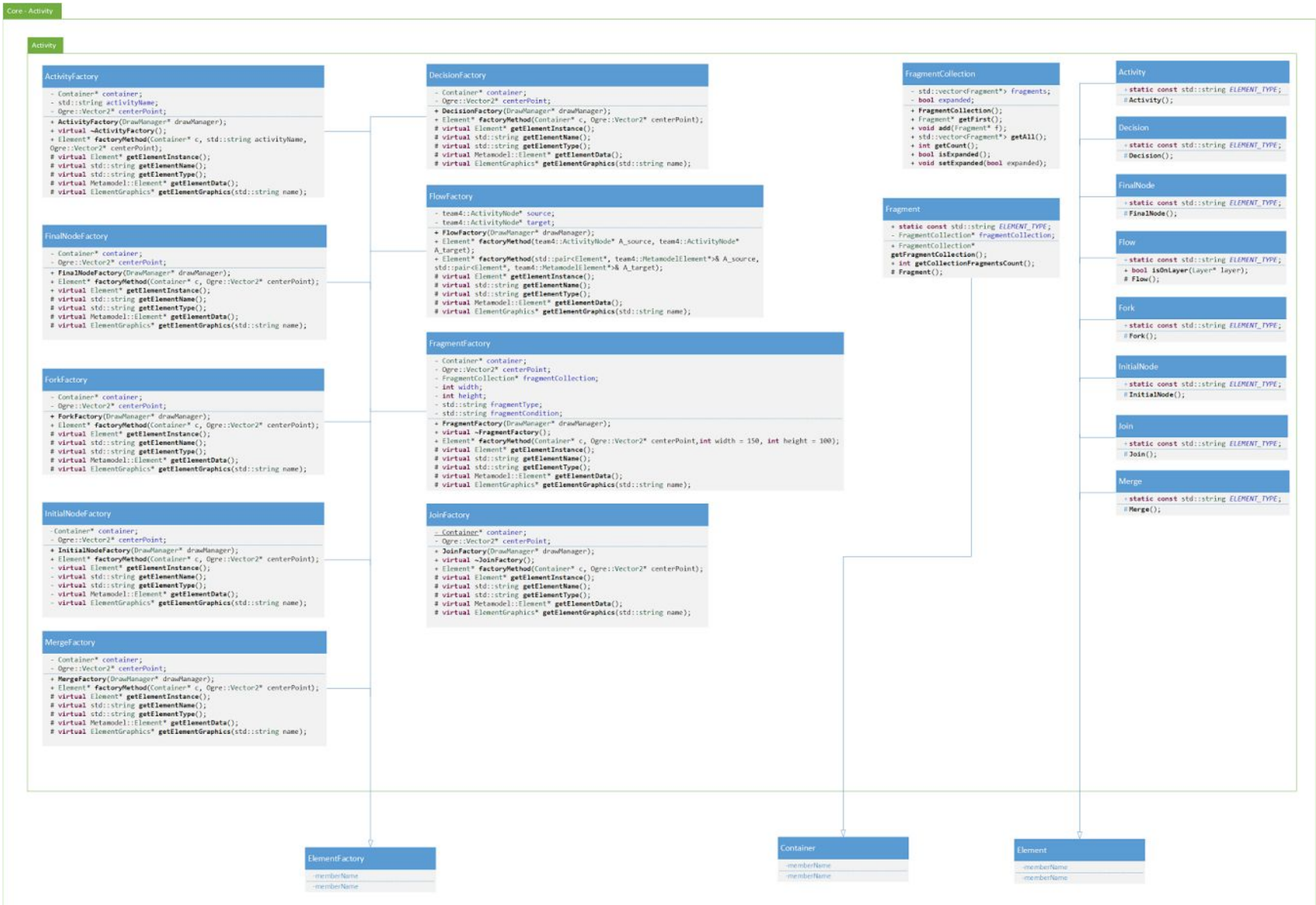
1. Pri kompilácii sa rovno na začiatku vypíše chyba typu “G++ compiler is missing” - Táto chyba znamená, že nie je správne nastavená cesta k MinGW. Musí byť preto pridaná do “Project → Settings → C/C++ Build → Environment” do premennej “Path”. Pridaná cesta by mala vyzerať nasledovne “${MINGW_HOME}\bin;${MSYS_HOME}\bin”.
2. Projekt je skompilovaný, ale po spustení exe súboru zahlási chybu typu “Program 3d-uml has stopped working”. Chyba je pravdepodobne v zložke “resources” a je potrebné ju nahradiť. Odkaz na funkčnú zložku: “http://uloz.to/x3JxhGbB/resources-zip”.
3. Projekt je skompilovaný, ale po spustení exe súboru zahlási chybu typu “Cannot run program. Missing dll”. Po kompilácii sa nemusia presunúť knižnice potrebné pre spustenie programu a preto ich treba prekopírovať ručne zo zložky “dll” do “Debug”.

Iné problémy by pri kompilácii / spustení nemali nastať. V opačnom prípade je potrebné prejsť si návod podrobne, či boli všetky kroky vykonané správne.

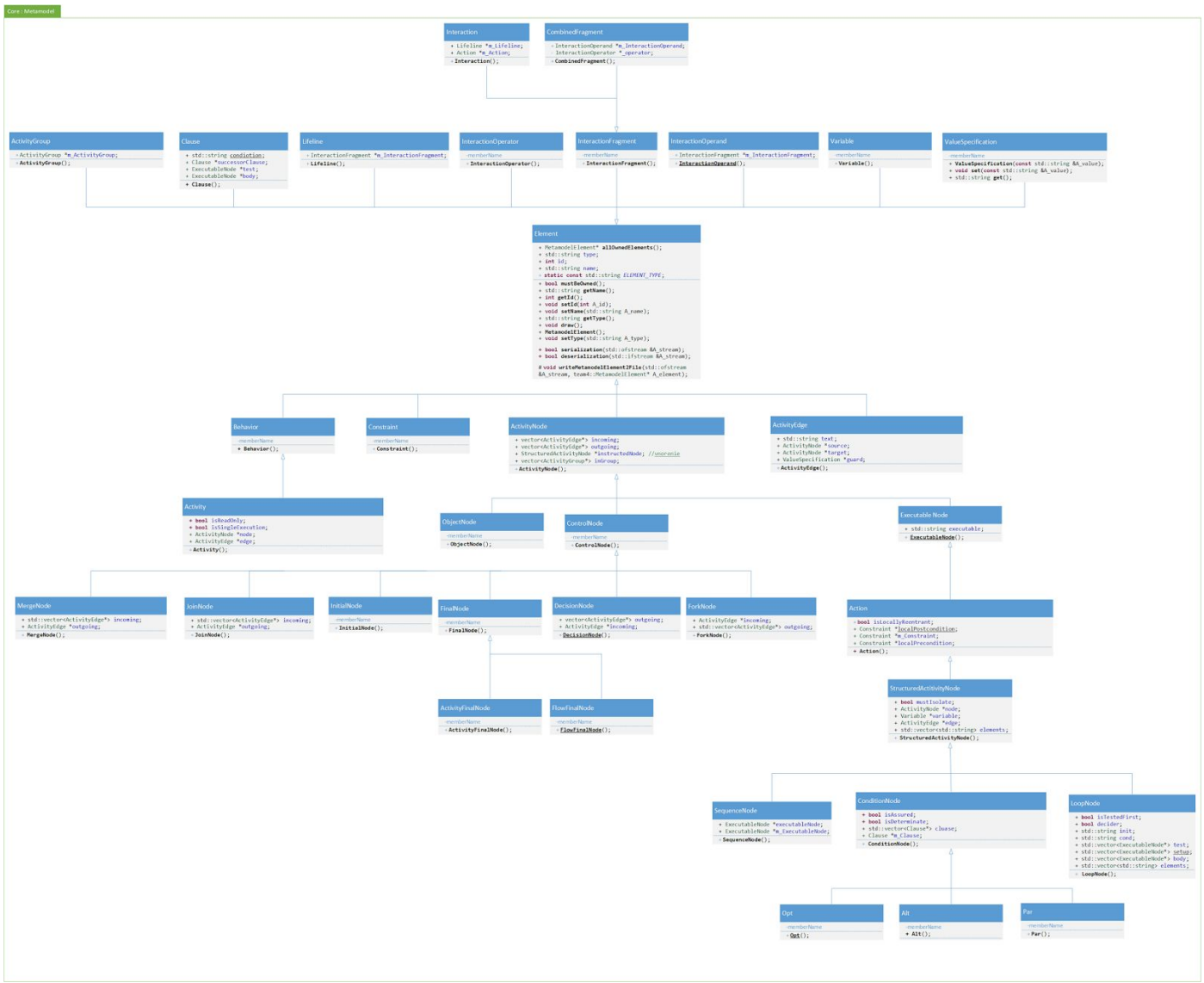
6 Technická dokumentácia

Pre lepšiu predstavu pri implementácii sme si v analytickej časti tímového projektu vytvorili diagramy pomocou Microsoft Visio[2], ktoré znázorňujú jednotlivé existujúce triedy, dedenie, premenné, metódy a prístupy k nim. Všetky vytvorené diagramy sa nachádzajú nižšie.

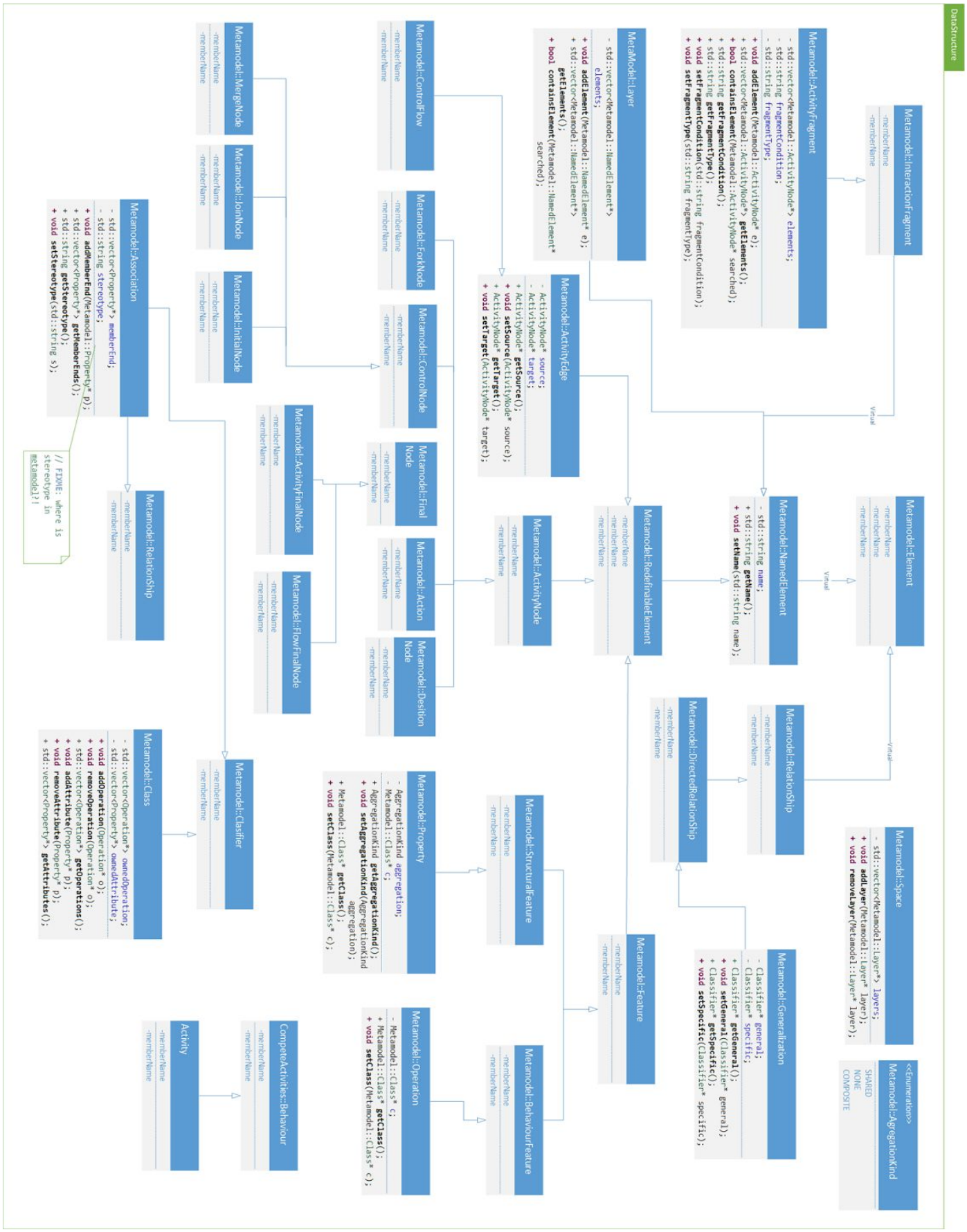
[2] https://products.office.com/en/visio/flowchart-software



Obr. 30: Core/Activity triedy

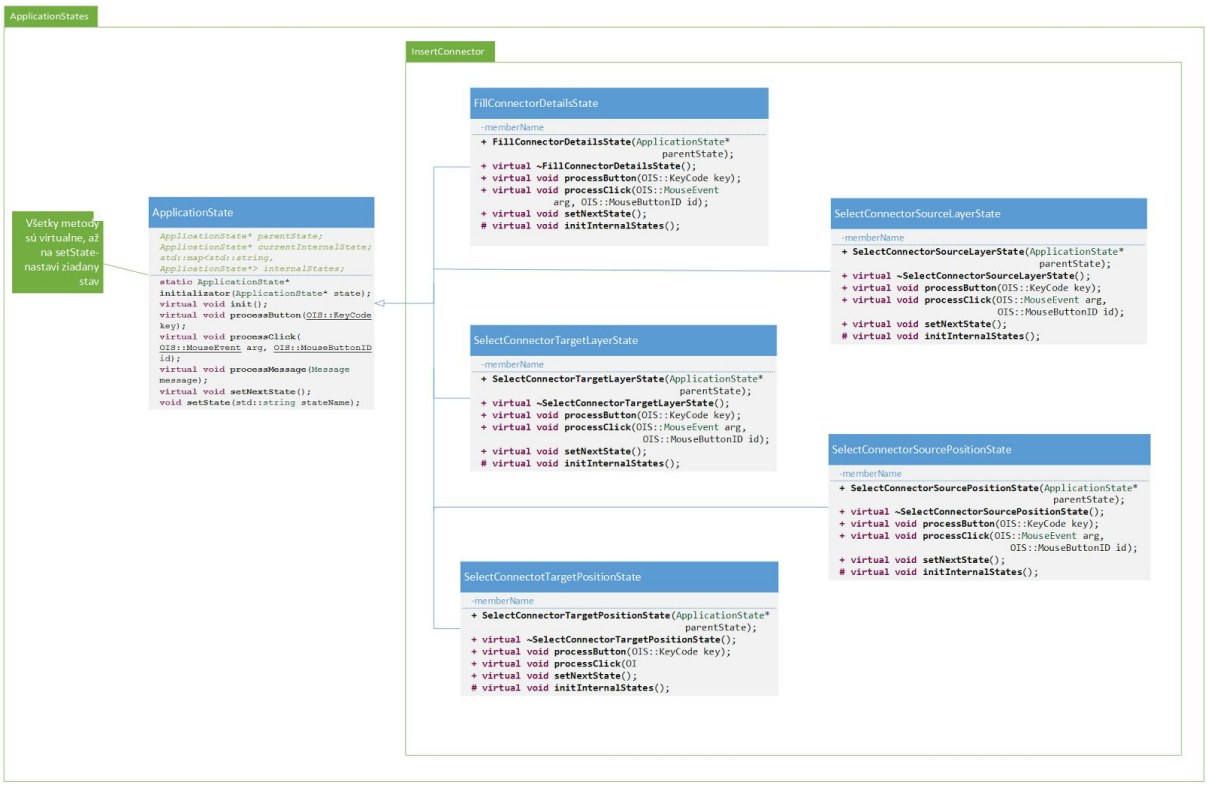


Obr. 31: Core/Metamodel triedy

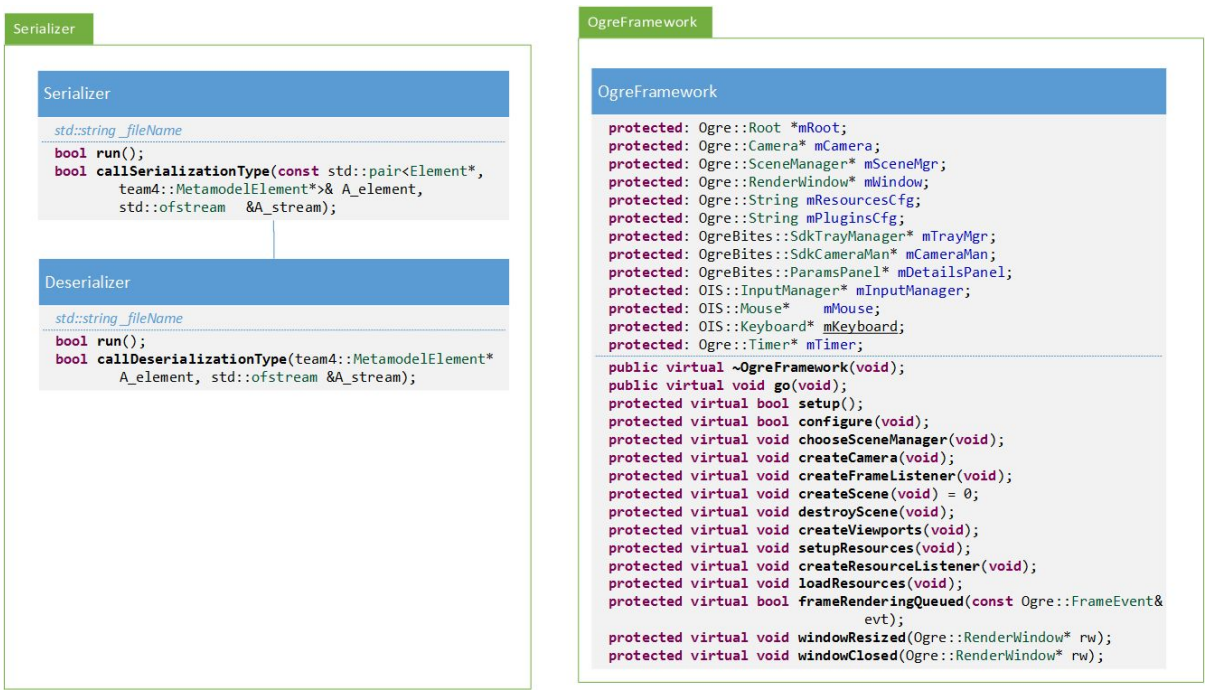


Obr. 32: DataStructure triedy

Obr. 33: Graphics triedy



Obr. 34: InsertConnector triedy



Obr. 35: OgreFramework serialization triedy