

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Ilkovičova 2, 842 16 Bratislava 4

Tímový projekt

ArchMages

AnimArch

Vedúci tímu: doc. Ing. Ivan Polášek, PhD.

Členovia tímu: Bc. Andrej Greguš

Bc. Lukáš Radoský

Bc. Michal Zeliska

Bc. Matej Cáder

Bc. Samuel Kubala

Bc. Krištof Zubrický

Bc. Oleksij Kholodov

Kontakt na tím: *tim19.2019.fiit@gmail.com*

Akademický rok: 2019/2020

Zoznam skratiek

ANTLR - *ANother Tool for Language Recognition*; je generátor syntaktického analyzátoru

AST - *Abstract Syntax Tree*; abstraktná reprezentácia výrazov v danom jazyku

API - *Application Programming Interface*; rozhranie poskytované aplikáciou pre komunikáciu s inými aplikáciami

EA - *Enterprise Architect*; softvér pre modelovanie softvéru v UML

GUI - *Graphical User Interface*; používateľské rozhranie pozostávajúce z interaktívnych vizuálnych prvkov

JV - *Juraj Vincúr*; Ing. Juraj Vincúr poskytol svoj zdrojový kód pre vizualizáciu diagramu tried v Unity v 3D pre tento projekt

OAL - *Object Action Language*; platformovo nezávislý jazyk kompilovateľný na platformovo závislé jazyky

OOANS - *Objektovo orientovaná analýza a návrh softvéru*; predmet na FIIT STU, ktorého dvaja študenti vytvorili projekt pre animáciu diagramu tried v Unity skúmaný vo fáze analýzy

POCO - *Plain Old C# Object*; trieda predstavujúca entitu, obsahujúca len (neinfraštrukturálne) atribúty a *get/set* metódy

TFS - *Team Foundation Server*; bývalý názov nástroja pre správu úloh, dnes známeho ako Azure DevOps

UML - *Unified Modeling Language*; jazyk pre modelovanie softvéru

VS - *Visual Studio*; vývojové prostredie pre jazyk C#

XMI - *XML Metadata Interchange*; štandardizovaný jazyk pre zachytenie metadát v XML

XML - *eXtensible Markup Language*; štandardizovaný jazyk pre zachytenie dát

xtUML/xUML - *executable UML*; jazyk pre tvorbu vykonateľných modelov softvéru

Obsah

Zoznam skratiek	2
Obsah	4
Inžinierske dielo	12
Úvod	13
Globálne ciele projektu	14
Globálne ciele letného semestra	15
Zhodnotenie letného semestra	15
Celkový pohľad na systém	17
Poskytovaná funkcia	17
Architektúra	18
Dátový model	18
Moduly	19
Moduly systému	20
EA Addin	20
Analýza	20
Návrh	21
Implementácia	23
Testovanie	25
Unity modul	26
Analýza	26
Návrh	27
Implementácia	29
Testovanie	31
XMI parser	31

Analýza	31
Návrh	32
Implementácia	32
Testovanie	34
OAL parser	34
Analýza	34
Návrh	35
Implementácia	36
Testovanie	37
OAL vykonávanie	38
Analýza	38
Návrh	39
Implementácia	40
Testovanie	42
Príručky	43
Inštalačná príručka pomocou Microsoft VS	45
1. Pred použitím príručky je potrebné vlastniť:	45
2. Vytvorenie projektu vo Visual Studiu 2019 a pridanie zdrojového kódu	46
3. Nastavenie projektu	47
4. Pridanie Addinu do registrov OS	48
5. Build addinu vo VS	49
6. Overenie správnej inštalácie addinu	50
Používateľská príručka	51
1. Nastavenie Unity modulu	51
2. Spustenie addinu	51
3. Zobrazenie Unity v Add-Ins okne	52

4. Diagram tried v Unity	53
Technická dokumentácia	59
EA Add-in	59
Vykonávanie OAL kódu	61
EXEScope	61
EXEScopeCondition	62
EXEScopeLoopWhile	63
EXEScopeForEach	64
EXEScopeParallel	65
EXEThreadSychronizator	67
XMI Parser	69
Prílohy	73
Príloha A - Testovacie scenáre	73
Požiadavky na používanie Unity Addinu pre animovanie architektúr	73
Inštalácia	73
Spustenie pluginu v Enterprise Architect	75
Vytvorenie jednoduchej animácie	78
Testovanie podmienku “IF”	84
Testovanie cyklus “WHILE”	88
Testovanie vnorený “WHILE” a “IF”	92
Testovanie paralelizmu	96
Testovanie načítania animácie	100
Príloha B - Dokumentácia modifikovaného jazyka OAL	103
Vysvetlivky syntaxe	103
Biele znaky	103
Komentáre	104

Kľúčové slová	104
Názvy	105
Dopyty	105
Create class instance	106
Create relationship between class instances	106
Select class instance - by class	107
Select class instance - by relationship specification	107
Delete class instance	108
Delete relationship between class instances	108
Priradenie	108
Priradenie do premennej	109
Priradenie do atribútu inštancie triedy	109
Volanie metód	110
Volanie metód	110
Jednoduché riadiace konštrukcie	110
Break	110
Continue	111
Výrazy	111
Koncová hodnota	111
Operácia	113
Typy a operátory	113
Unárne operátory	113
Binárne operátory	114
N - árne operátory	114
Zložené príkazy	116
If - elif - else	116

While	117
Foreach	119
Parallel - thread	120
EBNF Pôvodného OAL	121
Príloha C - Protokol z externého testovania	128
Pozitíva:	128
Negatíva	128
Príloha D - Gramatika našej verzie OAL	129
Príloha E - Podklady pre katalóg vzorov	137
Postup pri ukladaní vzorov v Enterprise Architect	137
Postup pri využíti vzorov v Enterprise Architect	139
Riadenie projektu	143
Úvod	143
Role členov tímu	145
Bc. Lukáš Radoský	145
Bc. Andrej Greguš	146
Bc. Michal Zeliska	146
Bc. Matej Cáder	146
Bc. Samuel Kubala	146
Bc. Krištof Zubricky	146
Bc. Oleksij Kholodov	147
Podiel práce	147
Podiel práce na jednotlivých šprintoch	147
Podiel práce na dokumentácii k inžinierskemu dielu	148
Podiel práce na dokumentácii k manažmentu riadenia	149
Aplikácie manažmentov	151

Manažment vývoja	151
Manažment komunikácie	151
Manažment úloh	151
Manažment zdrojového kódu	152
Sumarizácia šprintov	153
Šprint 0	153
Šprint 1	153
Šprint 2	154
Šprint 3	155
Šprint 4	156
Šprint 5	156
Šprint 6	157
Šprint 7	157
Šprint 8	158
Šprint 9	158
Šprint 10	159
Šprint 11	159
Retrospektíva	160
Zimný semester	161
Šprint 0	161
Šprint 1	161
Šprint 2	162
Šprint 3	162
Šprint 4	163
Šprint 5	163
Letný semester	163

Šprint 6	163
Šprint 7	164
Šprint 8	165
Šprint 9	165
Šprint 10	166
Šprint 11	166
Motivačný dokument	167
Metodiky	168
Metodika komunikácie pomocou Discord	169
Pravidlá jednotlivých komunikačných kanálov	169
Pravidlá jednotlivých dátových kanálov	170
Metodika správy úloh	172
Pravidlá vytvárania User Story	172
Definícia hotovo pre User Story	172
Stav User Story	172
Backlog	173
To Do	173
In Progress	173
Resolved	173
Closed	174
Pravidlá vytvárania Task	174
Definícia hotovo pre úlohu	174
Stav úloh	174
New	174
Active	175
Closed	175
Metodika správy kódu pomocou Github	176

Pravidlá práce s GitHub:	176
GitHub príkazy	176
Metodika písania kódu	180
Odsadzovanie kódu	180
Komentáre	180
Mená v kóde	182
Skripty	182
Prístupy	182
Export evidencie úloh	183
Šprint 0	183
Šprint 1	183
Šprint 2	184
Šprint 3	186
Šprint 4	193
Sprint 5	195
Šprint 6	198
Šprint 7	200
Šprint 8	202
Šprint 9	203
Šprint 10	205
Šprint 11	207
Webové sídlo projektu (a iné odkazy)	209
Pod'akovanie	209

Inžinierske dielo

Úvod

Tento dokument predstavuje dokumentáciu projektu AnimArch vytváraného v rámci predmetu Tímový projekt tímom ArchMages (č. 19) v akademickom roku 2019/2020 na FIIT STU. Dokumentuje proces tvorby produktu, postupné formovanie očakávanej funkcionality, ako aj výsledný produkt samotný. Na konci dokumentu je možné nájsť odkazy na úložiská zdrojového kódu.

Cieľom dokumentu je uviesť čitateľa do problematiky riešenej projektom a do štruktúry projektu, na vysokej aj na nižšej úrovni. Dokument neposkytuje podrobné vysvetlenia použitých existujúcich technológií. Objasňuje ich vlastnosti ovplyvňujúce projekt, avšak nie do hĺbky.

Globálne ciele projektu

Výstupom zimného semestra bola Unity aplikácia, ktorá dokázala jednoduchým spôsobom animovať diagram tried z predvoleného XMI súboru. Animácia bola tvorená interaktívne klikaním v diagrame a zapísaná jednoduchým jazykom bez špecifickej syntaxe. Nebolo možné do animácie pridať iné príkazy, ani ju nebolo možné zapojiť ako plugin do EA a animovať ľubovoľný diagram.

Ciele stanovené v zimnom semestri boli vtedy splnené nasledovne (v poradí podľa stanovenej priority):

- 1. Vytvorenie diagramu tried** - ako editor diagramu sme využívali EA, avšak aplikácia s ním nebola prepojená. Tento cieľ bol *splnený čiastočne*.
- 2. Vytvorenie animácie diagramu tried** - animáciu bolo možné vytvárať interaktívne klikaním priamo v diagrame. Tento cieľ bol *splnený*.
- 3. Vykonanie animácie** - aplikácia animovala diagram tried na základe zápisu animácie. Tento cieľ bol *splnený*.
- 4. Debug mód** - aplikácia tento cieľ nenaplnila. Bol vytvorený separatny CLI throwaway prototyp ako podklad pre ďalšiu prácu. Umožňoval vytváranie animácie a parsovanie príkazov v modifikovanom jazyku OAL. Tento cieľ bol *splnený čiastočne*.
- 5. Katalóg vzorov** - tento cieľ *nebol splnený*.
- 6. Generovanie kódu** - tento cieľ *nebol splnený*.

Úspech semestra bol zadefinovaný ako naplnenie prvých troch požiadaviek aspoň na minimálnej úrovni. Dve z týchto požiadaviek boli splnené a jedna len čiastočne. Zároveň sa podarilo čiastočne splniť štvrtú požiadavku. Konštatujeme preto, že zimný semester bol úspešný.

Globálne ciele letného semestra

Na základe poznatkov a artefaktov získaných v zimnom semestri boli určené ciele pre letný semester:

1. **Prepojenie aplikácie s EA** - aby aplikácia bola použiteľná v produkčnom prostredí, musí byť dostatočne robustná, čo sa týka rozmanitosti diagramov. Tiež musí pracovať ako plugin do EA, nie ako standalone aplikácia s jedným predvoleným diagramom.
2. **Vykonávanie animácie v jazyku OAL** - aby sa používateľ nemusel učiť náš vlastný jazyk pri vytváraní animácie, rozhodli sme sa využiť existujúci jazyk OAL. Tiež pomocou neho možno poskytnúť konštrukcie pre obohatenie animácie, ako sú podmienka, cyklus, paralelný blok, výpočet výrazov, manažment premenných a objektov. Je teda nutné navrhnúť podmnožinu jazyka OAL a obohatiť ju o vlastné konštrukcie, vytvoriť gramatiku a parser pre tento jazyk, implementovať vykonávanie tohto jazyka, jeho generovanie pri klikaní v diagrame a editor pre písanie animácie.
3. **Ovládanie animácie** - spúšťanie animácie možno obohatiť o pozastavenie, ukončenie, obnovenie prehrávania, krokovací režim, či spätný chod.
4. **Debug mód** - tento cieľ je previazaný s cieľom číslo 2. Počas vykonávania animácie by malo byť používateľovi umožnené dynamicky meniť hodnoty premenných a atribútov objektov.
5. **Katalóg vzorov** - systém by mal poskytnúť katalóg návrhových vzorov a umožniť ich vkladat do existujúcich diagramov a animácií.

Zhodnotenie letného semestra

Na základe výstupov letného semestra uvádzame krátke zhrnutie, ako boli jednotlivé ciele letného semestra naplnené:

- 1. Prepojenie aplikácie s EA** - aplikácia bola integrovaná do EA ako plugin. Umožňuje otvoriť v aplikácii ľubovoľný¹ diagram tried, vytvárať preň animáciu a spúšťať ju. Tento cieľ bol *splnený*.
- 2. Vykonávanie animácie v jazyku OAL** - animácia je zapísaná v modifikovanom jazyku OAL. Okrem animovania volaní poskytuje podmienku, cyklus, *foreach* cyklus, paralelný blok, objekty, premenné a výpočet výrazov. Zložené príkazy a výrazy možno vnárať do ľubovoľnej² hĺbky. Bola preň vytvorená gramatika a parser pomocou ANTLR. Bolo implementované vykonávanie všetkých konštrukcií jazyka. Editor poskytuje syntax highlighting. Obmedzenie je, že nie je vykonávaná kontrola syntaxe. Tento cieľ považujeme za *splnený*.
- 3. Ovládanie animácie** - funkcionality animovania bola rozšírená o pozastavenie, ukončenie, obnovenie prehrávania, krokovací režim, či spätný chod. Animácia samotná bola tiež upravená, aby pôsobila efektnejšie. Tento cieľ považujeme za *splnený*.
- 4. Debug mód** - interaktívne zásahy do vykonávania animácie neboli implementované. Tento cieľ *nebol splnený*.
- 5. Katalóg vzorov** - tento cieľ *nebol splnený*. Jeho splnenie bude predmetom bakalárskej práce v ďalšom akademickom roku.

¹ môžu sa uplatňovať obmedzenia

² obmedzené výpočtovým výkonom zariadenia

Celkový pohľad na systém

Aplikácia AnimArch poskytuje samostatnú metódu animácie softvérových architektúr a scenárov. Operuje nad diagramom tried, ktorý animuje na základe príkazov zapísaných v modifikovanom jazyku OAL. Jazyk umožňuje vytvárať jednoduché scenáre (postupnosť animačných krokov), alebo sofistikované scenáre pomocou podmienok, cyklov, paraleлизmu, objektov, premenných a výpočtov nad výrazmi. Aplikácia umožňuje pracovať s diagramami vytvorenými v EA priamo pomocou kliknutia v EA.

Poskytovaná funkcionalita

Používateľ si najskôr musí vytvoriť diagram tried. Túto funkcionalitu neposkytuje AnimArch. Využívame už existujúci editor v EA. Používateľ si teda vytvára nový diagram v EA, alebo využije svoj už existujúci diagram. Prostredníctvom GUI EA zvolí možnosť pre spustenie AnimArchu, ktorý následne zobrazí daný diagram v Unity okne, ktoré sa otvára ako súčasť GUI EA. Dáta o diagrame putujú do AnimArchu vo forme XMI exportu poskytovaného EA API.

Rozloženie tried v priestore vychádza z rozloženie v EA, avšak je možné využiť aj automatizované určené rozloženie. Na diagram je možné nazerat z ľubovoľnej perspektívy, otáčajúc ho v 3D priestore. Tiež je možné meniť rozloženie tried manuálne a ofarbovať individuálne triedy.

Vytváranie animácie v AnimArchu prebieha tak, že používateľ kliká na triedy, následkom čoho je generovaný kód v jazyku OAL. Kód je možné manuálne editovať, a doplniť ho tak o sofistikovanejšie správanie. Animáciu možno uložiť ako textový súbor a načítať inú. Pre jeden diagram je tak možné vytvoriť viacero rôznych animácií.

Prehrávanie animácie prebieha tak, že v diagrame sú postupne vysvecované triedy a hrany medzi nimi. Hrany sú vysvecované inkrementálne, vytvárajúc dojem pohybu. Animáciu je možné pozastaviť a opäť spustiť, úplne zastaviť, spúšťať krokovane alebo

využiť spätný chod. Tiež je možné dynamicky meniť farebnú schému prostredia a rýchlosť chodu animácie.

Cieľom je poskytnúť vývojárom softvéru jednoduchý spôsob, ako efektívne vysvetliť kolegom svoj návrh. Animácia by mala zjednodušiť pochopenie navrhnutého softvérového riešenia. Podobne je to s vysvetľovaním návrhu testerom, novým členom tímu pri onboardingu alebo pri demonštrovaní návrhových vzorov študentom.

Architektúra

AnimArch je navrhnutý ako desktopová aplikácia, takže tvorí monolit. Nevyužíva databázový systém ani cloudové riešenia. Závisí od EA, nakoľko funguje ako jeho plugin. Je možné ho využívať aj ako standalone aplikáciu, v tejto fáze však len s prednastaveným diagramom. Komunikácia medzi AnimArchom a EA prebieha pomocou EA API, pričom AnimArch sleduje a vyvoláva udalosti v EA. Ide najmä o sledovanie kliknutí pri voľbe možnosti spustenia AnimArchu a vyvolanie a zachytenie XMI exportu aktuálne otvoreného diagramu. AnimArch je implementovaný v jazyku C# využívajúc Unity engine. Jednotlivé moduly sú opísané v príslušnej časti.

Dátový model

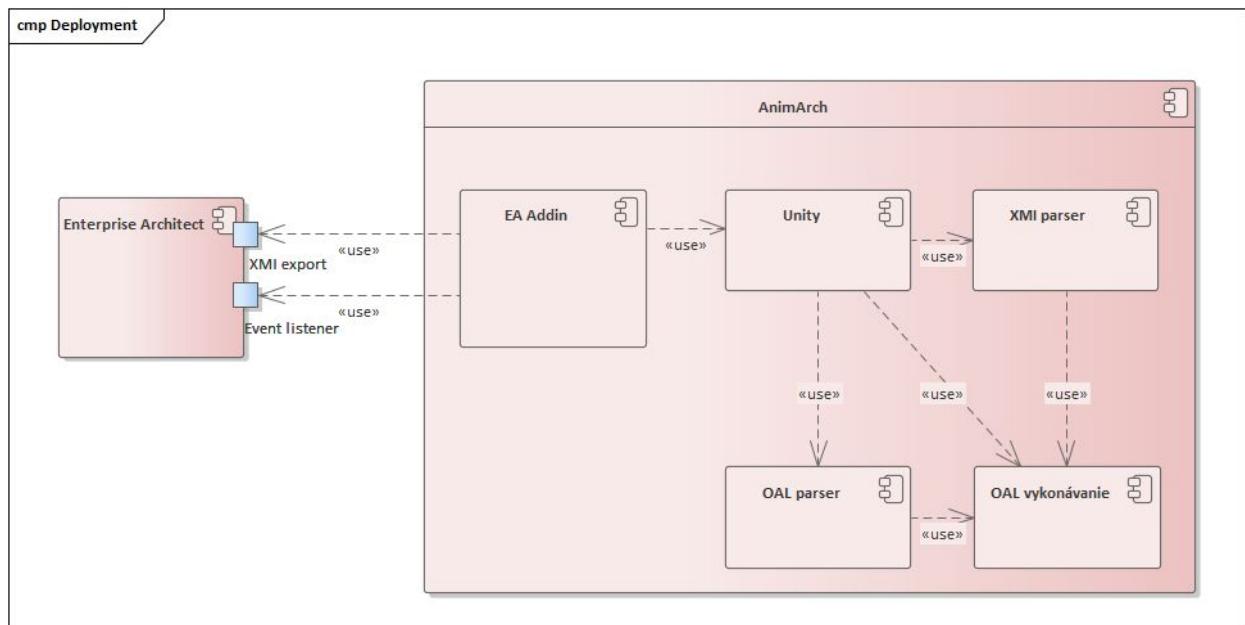
Projekt v súčasnej fáze implementácie nevyužíva databázový systém, preto nemožno hovoriť o databázovej schéme. Interná reprezentácia dát je uvedená pri jednotlivých moduloch vo forme diagramov tried.

Komunikácia medzi EA a AnimArchom prebieha vo forme XMI, čo je štandardný formát výmeny metaúdajov³. Animácia je zapísaná v syntaxi modifikovanej verzie jazyka OAL, ktorý je opísaný v časti [Príloha B - Dokumentácia modifikovaného jazyka OAL](#).

³ <https://www.omg.org/spec/XMI/About-XMI/>

Moduly

V rámci AnimArch možno identifikovať samostatné moduly *Unity*, *OAL vykonávanie*, *OAL parser*, *XMI parser*, *EA Addin*.



Obrázok č. 1: Moduly projektu na najvyššej úrovni

EA je externý modul, ktorý nie je vyvýjaný tímom ArchMages. Poskytuje funkciaľitu tvorby diagramov tried, vďaka čomu nie je potrebné implementovať vlastný editor diagramov. Nakoľko implementácia tohto modulu nie je riadená ani vykonávaná tímom ArchMages, nie je ďalej popisovaný.

EA Add-in je modul zodpovedný za komunikáciu medzi **EA** a **AnimArchom**. Zachytáva vyvoláva udalosti v EA prostredníctvom EA API.

Unity modul implementuje celú vizuálnu stránku projektu, a zároveň slúži ako riadenie celej aplikácie.

XMI parser parsuje údaje o diagrame tried z XMI súboru získaného ako export z **EA**. Inštanciuje triedy pre potreby vnútornnej reprezentácie diagraamu **Unity** modulom.

OAL parser je modul čiastočne vygenerovaný pomocou ANTLR. Získavanie údajov z parsovacieho stromu bolo implementované manuálne. Konvertuje OAL kód vo forme textu na triedy reprezentujúce OAL program.

OAL vykonávanie je modul obsahujúci triedy reprezentujúce diagram tried a OAL program, a umožňuje vykonávanie OAL programu.

Moduly systému

EA Addin

EA addiny umožňujú rozšíriť EA o vlastnú funkcia vrátane grafického rozhrania. V našom prípade je hlavnou úlohou addinu zabezpečiť komunikáciu medzi EA a Unity modulom. V aktuálnom stave projektu je jeho úlohou aj zobrazenie Unity modulu priamo v EA. Takéto zobrazenie nevyžaduje otvorenie externého okna, pretože využíva Add-Ins okno poskytované EA, čím v používateľovi navodzuje pocit, že stále pracuje v EA.

Analýza

Na komunikáciu medzi EA a Unity modulom je možné použiť viacero spôsobov. Prvým spôsobom je komunikácia pomocou súboru. To znamená, že addin vyexportuje XMI súbor aktuálne otvoreného class diagramu, ktorý bude slúžiť ako vstupný súbor pre Unity modul.

Druhým spôsobom by bolo použitie dátovodov (pipes), ktoré by umožňovali komunikáciu priamo cez operačnú pamäť. EA API však neposkytuje export XMI do pamäte, a preto sa vytvoreniu súboru nedá vyhnúť. Dali by sa však použiť tzv. queries priamo nad dátovým modelom EA, čím by sa dali získať potrebné údaje pre Unity modul priamo v pamäti bez exportovania XMI. Od tejto možnosti sa upustilo z dôvodu možnosti prípadnej zmeny dátového modelu EA, ktorá by úplne znefunkčnila nás addin.

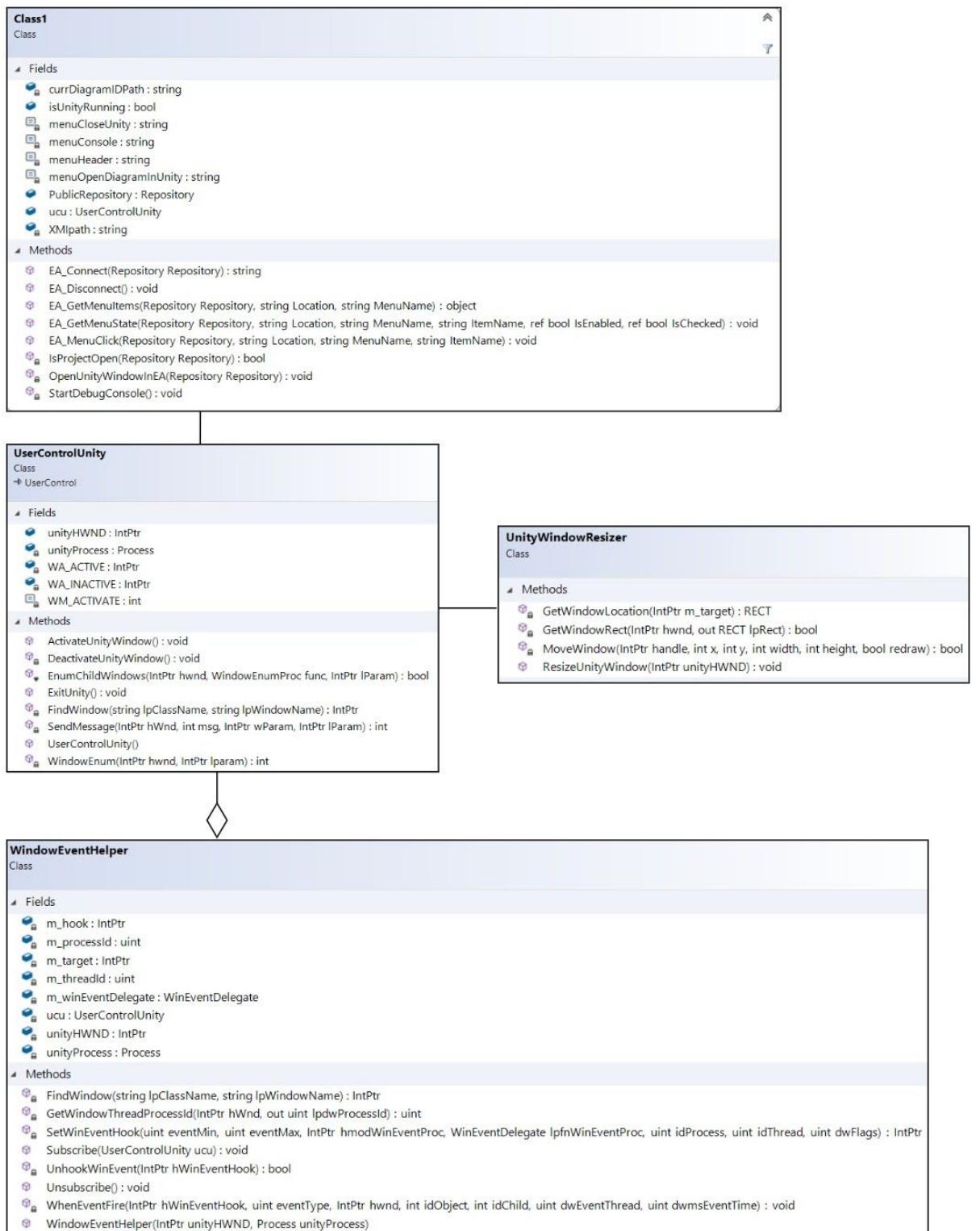
Zobrazenie externej aplikácie (v našom prípade Unity) nie je EA priamo podporované. Avšak vďaka možnosti pridania vlastného GUI do EA, je možné túto limitáciu

obísť. EA totiž dokáže zobraziť vytvorené GUI bez ohľadu na to, čo obsahuje. Preto stačí externú aplikáciu pridať do vyvíjaného GUI. Samotné Unity takéto zobrazenie oficiálne podporuje, ak je spustené ako dcérsky proces kompatibilného grafického elementu. Problémom takéhoto zobrazenia je práca s Add-Ins oknom, napr. resizovanie, maximalizácia, ukončovanie a iné. EA API takúto funkcionality nepodporuje, a preto, aby bolo možné poskytnúť pre používateľa základné operácie s oknom, je nevyhnutné použiť Windows API.

Návrh

EA komunikuje s EA addinom pomocou správ o rôznych udalostiach, ktoré nastali v používateľskom rozhraní. Vďaka týmto správam (napr. kliknutie na položku v menu), môže addin reagovať a vyvolať definované funkcie. V našom prípade sú potrebné iba dve položky v menu, a to možnosť zobrazenia diagramu tried v Unity (spustenie addinu) a možnosť ukončenia addinu. EA API umožňuje položku spustenia addinu sprístupniť len v prípade, ak je práve zobrazený diagram tried. Po kliknutí na túto položku v menu sa vygeneruje XMI súbor balíka, v ktorom sa nachádza aktuálny diagram tried, nakoľko vygenerovanie XMI iba samotného diagramu nie je možné. Keďže v balíku môže byť viac diagramov je potrebné si uložiť do súboru aj ID aktuálne otvoreného diagramu. Následne sa pomocou EA API pridá trieda GUI do EA okna Add-Ins.

V rámci konštruktora GUI triedy sa inicializuje a vytvorí Unity proces, ktorý sa pripne na GUI. Na ovládanie Add-Ins okna je použité Windows API, keďže EA API manipuláciu s Add-Ins oknom neumožňuje. Vďaka Windows API je možné odpočúvať window eventy Add-Ins okna a následne na ne príslušne reagovať.

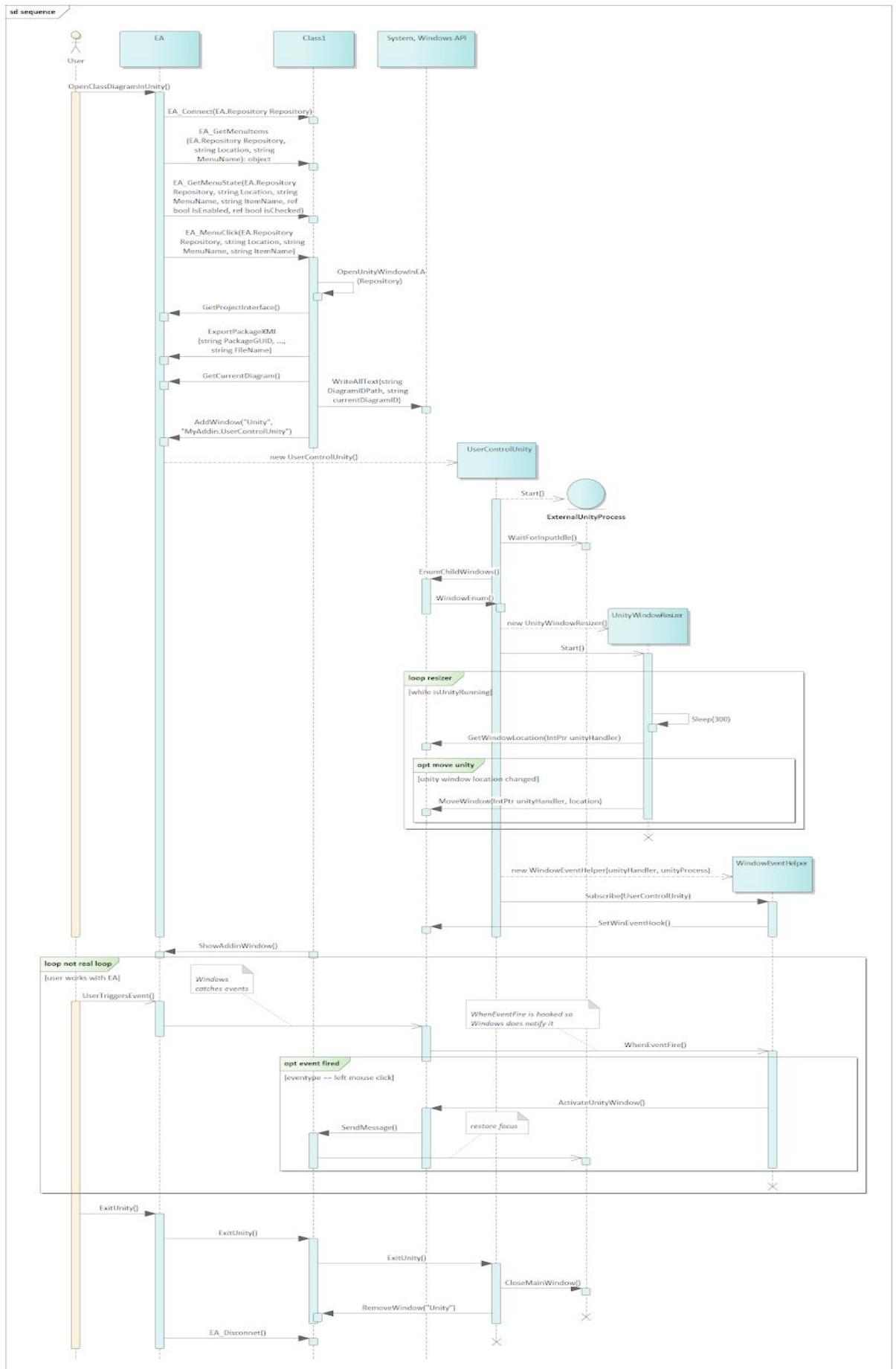


Obrázok č. 2: Modul EA addin - diagram tried

Implementácia

Addin je vyvíjaný v jazyku C# ako .dll knižnica pre EA. Samotná implementácia sa zhoduje s návrhom. EA API je poskytované knižnicou Interop.EA.dll a Windows API knižnicou user32.dll obsahujúcu funkcionality na prácu s rozhraním Windowsu.

Hlavná trieda addinu, *Class1*, obsahuje implementované správy EA, spomínané vyššie v návrhu. Trieda *UserControlUnity* implementuje GUI, inicializuje, spúšťa a zobrazuje Unity modul. Na zachytávanie a spracovávanie window udalostí slúži trieda *WindowEventHelper*. Konkrétnie sa zachytávajú eventy kliknutia ľavého tlačidla myši z dôvodu, že je potrebné vrátiť focus späť na addin, ak používateľ klikol mimo obrazovku addinu. Čo sa týka resizovania, maximalizovania alebo ukončenia okna, prichádzajúce eventy týchto akcií nie sú jednoznačné a v niektorých špeciálnych stavoch okna sa negenerovali vôbec, preto sú na zabezpečenie tejto funkcionality addinu nepoužiteľné. Taktiež množstvo zachytávaných eventov a ich spracovanie značne spomalovalo celý addin až do bodu kedy EA mohol nečakane spadnúť. Na vyriešenie tohto problému bola navrhnutá trieda *UnityWindowResizer*, ktorá periodicky každých 300 milisekúnd kontroluje pomocou Windows API pozíciu addin okna a v prípade ak sa zmenila tak Unity okno pomocou funkcie *MoveWindow()* resizne na novú pozíciu. Výhodou je, že tento spôsob zbytočne nepreťažuje EA a taktiež funguje vo všetkých špeciálnych stavoch okna, ktoré EA poskytuje. Používateľ si teda môže pripnúť addin Unity okno kde chce. Na lepšiu predstavu samotnej implementácie slúži nasledujúci sekvenčný diagram, ktorý zachytáva aktivitu addinu až po ukončenie EA.



Obrázok č. 3: Modul EA addin - diagram sekvencií základného toku vykonávania

Testovanie

Ked'že addin je vyvíjaný ako .dll knižnica, jeho spustenie je možné iba ak je súčasne spustené EA. Preto všetky testy, ktorými bola overovaná správnosť fungovania addinu, boli vykonalé manuálne. Testovanie tohto modulu spočíva v testovaní Unity modulu cez EA, preto tento modul nemá samostatné testovanie.

Unity modul

Používateľ má v Unity module možnosť zobrazenia diagramu spôsobom, ktorý zodpovedá tomu v EA. Následne má pomocou GUI možnosť realizovať nasledovné akcie:

1. Priblížiť a vzdialiť pohľad na diagram.
2. Pohybovať prvkami diagramu a kamerou v 2D a 3D priestore.
3. Možnosť vytvorenia, úpravy a spustenia animácie.
4. Spustenie animácie je možné súvislou a krokovou animáciou.
5. Možnosť meniť vlastnosti animácie ako rýchlosť a použité farby.

Analýza

Počas analýzy sme skúmali možnosti existujúcich knižníc a ich aplikáciu pre potreby nášho zadania. Hlavnú knižnicu, na základe ktorej môžeme realizovať zobrazenie diagramu, nám sprostredkoval J. Vincúr. Je ňou **UMASGL**. Táto knižnica nám poskytuje predpripravené Prefaby(herný objekt vyskladaný z množstva komponentov rôzneho typu - C# script, Rect transform, canvas renderer..) typu Graph, Class a Line. Pre nás platí, že nepotrebujeme poznáť konkrétnu implementáciu súčasti daných prefabov. Stačí nam si vytvoriť v Scéne objekt typu Graph, ktorý je akýmsi kontajnerom pre objekty typu Class a Line, tie budú predstavovať triedy a vzťahy medzi triedami. Pre špecifickejšie implementácie vzťahov nám J. Vincúr poskytol obmeny Prefabu Line predstavujúce asociáciu, generalizáciu a závislosť, aby sme mohli zobraziť aj tieto vzťahy.

Medzi všeobecnejšie knižnice, ktoré by sme mohli využiť, patrí:

- **TextMeshPro** - na prácu s textovými objektami v Unity.
- **SimpleFileBrowser** - open source implementácia pre prácu so súbormi.
- **UnityUiExtensions** - množstvo doplnkových UI elementov.

Návrh

Unity aplikácia sa bude skladať z troch hlavných častí:

- **Zobrazenie diagramu** - pomocou externej knižnice UMASGL chceme zobrazit diagram používateľovi tak, aby zobrazenie čo najlepšie zodpovedalo tomu v EA. Integrujeme funkciuálnitu XMI parseru a skriptu pre EA plugin.
- **GUI** - pomocou ktorého bude mať používateľ možnosť interaktívne pracovať s diagramom a inými funkciuálnitami ponúkanými naším EA pluginom.
- **Animáciou** - táto časť sa bude starat o manažovanie animačných skriptov od ich vytvorenia, úpravy, až po prezentáciu vhodným spôsobom. V tejto časti budeme integrovať moduly ANTLR parseru a spracovania OAL skriptu.

Zobrazenie diagramu - pri spustení Unity pluginu nám EA poskytuje všetky potrebné dátá o diagrame vo forme XMI súboru. Teda pri štarte Unity pluginu zavoláme modul XMI parseru a dátu, ktoré nám poskytne parser, si uložíme do vlastných štruktúr takým spôsobom, aby sme vedeli efektívne vyhľadávať jednotlivé triedy, metódy a vzťahy medzi nimi. Následne si vytvoríme inštanciu Graph objektu a pre každú entitu uloženú v našich dátových štruktúrach vytvoríme herný objekt, ktorý bude v hierarchii objektov dieťaťom Graph objektu a bude obsahovať všetky potrebné dátá pre jeho správne zobrazenie.

GUI bude pozostávať z panelu, ktorý bude staticky umiestnený v pravej časti obrazovky a budú v ňom možnosti pre pre tvorbu, zmenu a otvorenie animácie. Ako doplnkové možnosti môžu byť tlačidlá na automatické zoradenie diagramu. Panel nástrojov bude v strednej hornej časti obrazovky. Panel bude obsahovať tlačidlá s ikonami nástrojov na:

- Pohyb jednotlivých tried po osi X a Y
- Pohyb kamery po osi X a Y
- Priblíženie a oddialenie kamery

Pri stlačení jedného z tlačidiel sa ako aktívny nástroj nastaví nástroj zodpovedajúci danému tlačidlu a tlačidlo sa bude zobrazovať ako stlačené - neaktívne. Tlačidlá pre prácu s animáciami budú otvárať panely pre:

- **Vytvorenie a úprava zvolenej animácie** - v tomto paneli chceme umožniť, aby si používateľ vedel interaktívnym spôsobom vytvoriť alebo upraviť animáciu klikaním na triedy a metódy v priamo v zobrazenom diagrame. Vytvárame štvorce volajúca trieda, volaná metóda, cielená trieda, cielená metóda. Ďalej poskytujeme možnosť vidieť a upravovať vygenerovaný animačný skript v textovom poli a uložiť vytvorený skript v textovej forme na zvolené miesto na disku.
- **Načítanie uloženého animačného skriptu** v textovej podobe vhodným spôsobom, idealne takým, aký poskytuje operačný systém Windows.
- **Prehrávanie animácie** - chceme vedieť prehrať, zastaviť alebo pauznúť zvolenú animáciu plynulým spôsobom. Používateľ by mal mať možnosť zmeniť rýchlosť a farby použité na animovanie tried, metód a hrán v diagrame. Tiež poskytujeme možnosť prepínať sa medzi módom plynulého prehrávania a krokovej animácie.

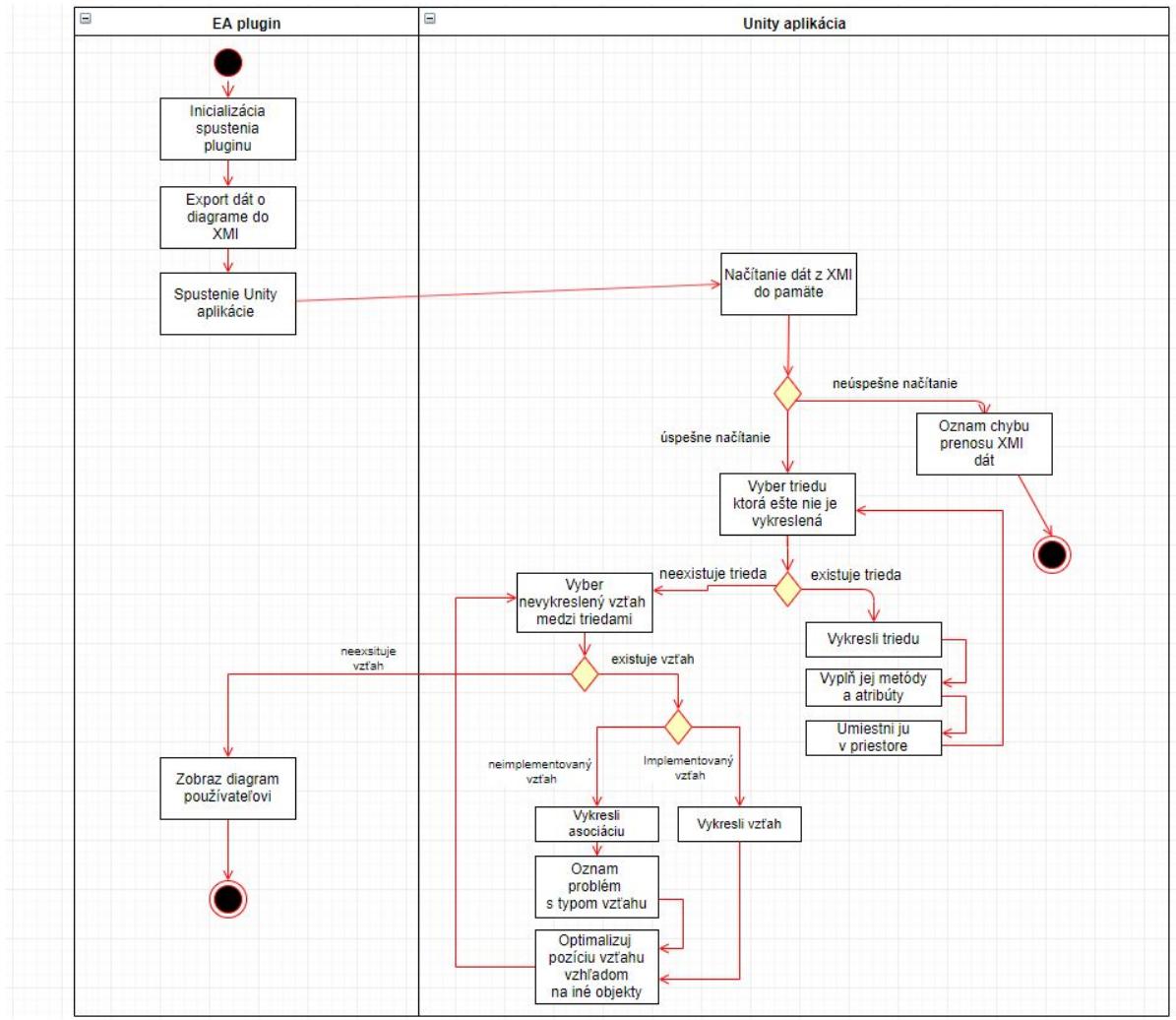
Animácia - v tejto časti si definujeme dva procesy, proces tvorby animácie a proces prehrávania animácie. Tvoríť animáciu chceme interaktívnym spôsobom, preto musia zobrazené časti diagramu vedieť reagovať na udalosti kliknutia myši. Pri inicializácii procesu tvorby animácie sa aktivuje možnosť používateľa si zvolať triedu, odkiaľ pôjde volanie. Trieda je následne vysvetlená a v príslušnom paneli bude mať používateľ možnosť si zvolať vybranú metódu, ktorú trieda obsahuje. Následne si používateľ zvolí cielovú triedu a cielovú metódu podobným spôsobom. Takto je jeden krok interaktívneho vytvárania skončený a príslušné volanie sa zapíše do textového poľa pre animačný skript. Používateľ má možnosť si sám doplniť skript o funkcie ponúkané OAL. Tento proces môže opakovať ľubovoľný počet krát. Výslednú animáciu si uloží do súboru. Okrem súboru sa animácia uloží aj do pamäte a nastaví sa ako zvolená animácia. V druhom procese má používateľ možnosť si prehrať zvolenú animáciu. Pri tejto voľbe sa animačný skript vykoná pomocou OAL spracovania a do poľa atomických príkazov sa uložia príkazy v poradí ako ich chceme zobraziť v animácii. Celá animácia má prebiehať tak, že sa postupne zafarbijú triedy, metódy a hrany tak, aby bolo jasné, ako prebieha interakcia medzi triedami v diagrame. Pre zafarbovanie hrany

spravíme plynulý tok(plynulé zafarbovanie hrany) smerom od volanej triedy do cieľovej triedy. Kroková animácia nám umožňuje ovládať tento proces tak, že jeden krok znamená jednu atomickú inštrukciu vyfarbenia.

Implementácia

Každú z troch častí z návrhu sme implementovali zvlášť tak, že pre každú časť existuje hlavná trieda, ktorá sa stará o riadenie danej časti. Riadiaca trieda je vždy typu Singleton, aby sme zjednodušili komunikáciu týchto tried v rámci projektu tak, že máme vždy priamy prístup k inštancii tejto triedy a vieme volať jej metódy. Pretože Unity aplikácia pracuje vo viacerých vláknach, neexistuje centrálné vlákno, ktoré by spravovalo všetky hlavné triedy, preto je dôležitá komunikácia týchto tried v rámci Unity Update metódy(to je metóda, ktorá sa volá zvlášť pre každú triedu, ktorá dedí od Unity triedy MonoBehaviour a vykonáva sa každý frame aplikácie) a prostredníctvom Unity event systému, ktorý vyvoláva metódy vzhľadom na vstup od používateľa. Okrem hlavných tried implementujeme množstvo tried, ktoré sú slúžia pri práci Unity komponentov. Tými sú napríklad Animator, Image renderer, Rect transform a ďalšie. Tieto triedy sú súčasťou vytvorených UI komponentov alebo prázdnych GameObjectov a svoju funkciu vykonávajú vzhľadom na príslušné udalosti od Unity, preto neuvažujeme interakciu a zobrazenie týchto tried v diagrame tried, ako to býva zvykom pri iných projektoch.

V časti **zobrazenie diagramu** hra hlavnú úlohu trieda *ClassDiagram*. Táto metóda dedí od triedy Graph zo spomínamej knižnice UMASGL. Jej úlohou je pomocou metódy Start pri spustení aplikácie prečítať XMI súbor pomocou XMI parseru a dátu si uloží do svojich štruktúr. Štruktúry na uloženie dát sú vo forme Listu alebo Dictionary a pre potreby vyhľadávania v dátach implementuje public metódy na vyhľadávanie podľa rôznych atribútov, napríklad podľa názvu triedy. Po načítaní dát sa spustí proces generovania GameObjectov z šablón uložených v Prefaboch, ktoré sme dostali k dispozícii s knižnicou **UMASGL**. Najprv sa vytvoria objekty typu *Class* a naplnia dátami. Keď sú všetky vytvorené, vytvoria sa objekty typu *Line* medzi vytvorenými objektami *Class*. Na záver sa objekty typu *Class* rozmiestnia tak, že im priradíme X,Y polohu v priestore, ktorú poznáme zo spracovaného XMI. Proces vieme zobraziť Activity diagramom:



Obrázok č. 4: Modul Unity - GUI a interakcia - diagram aktivít pri spustení modulu

GUI je implementované pomocou funkcionality, ktorú ponúka knižnica Unity Engine.UI. Hlavnou triedou, ktorá sa stará o interakciu UI komponentov, je trieda **UIManager**. Táto trieda implementuje množstvo public metód, ktoré sú vyvolávané prijatím eventov od UI komponentov. *Canvas* komponent sa postará o to, aby sa veľkosť UI vedela dynamicky prispôsobiť veľkosti okna. V *Canvas* sme vytvorili panely, do ktorých sme umiestnili jednotlivé tlačidlá a textové polia. O pozícii týchto prvkov v rámci UI sa stará komponent *RectTransform*, ktorý nám umožňuje zakotviť jednotlivé UI elementy na pravej strane obrazovky. Jednotlivé tlačidlá vyvolávajú eventy *OnClick()*, *OnPointerDown()*, *OnPointerUP()*, ktoré následne aktivujú metódy triedy *ToolManager* a triedy *FlyCamera*. *ToolManager* si ukladá aktívny nástroj a prostredníctvom *FlyCamera* triedy realizuje zmeny

pozície kamery v priestore, tak umožňuje interakcie s diagramom. O zmenu pozície prvkov diagramu sa starajú triedy z externej knižnice *UmaSGL*. Počas aktívneho nástroja pre pohyb prvku diagramu sa mení pozícia zvolenej triedy na základe pozície kurzoru a *UmaSGL* sa postará o prepočítanie pozície väzieb danej triedy s inými triedami v rámci zobrazeného diagramu.

V poslednej časti, **Animácia**, hrá hlavnú úlohu trieda *Animation*. Trieda implementuje atomické metódy pre vysvetlenie jednotlivých tried, metód a hrán. Parametre metód sú názvy metód/tried a bool premenná rozhodujúca, či majú byť vysvetlené alebo nie. Ďalej poskytuje metódy typu *Coroutine*, ktoré v Unity ponúkajú možnosť prerušiť vykonávanie metódy v nejakom bode jej vykonávania, počkať po stanovený čas a pokračovať vo vykonávaní. Na tomto princípe je vytvorená aj hlavná metóda *ResolveCallFunct*, ktorá sa stará o postupné vysvetlenie entít na základe parametra *OALCall*, ktorý obsahuje všetky potrebné informácie o volaných a cieľových triedach a metódach. V prípade vykonávania krokovnej animácie sa po každej metóde vysvetenia nečaká po zvolený čas, ale čaká sa na stlačenie tlačidla *next*, ktoré indikuje vykonanie ďalšieho vysvetenia. Trieda si pamäta predchádzajúce kroky v rámci jedného OALCallu a vie sa pomocou kroku späť vrátiť vo vykonávaní o krok dozadu. Trieda ponúka možnosť paralelného vykonávania prostredníctvom viacnásobného volania metódy *ResolveCallFunct*. Taktiež poskytuje metódu *UnhighlightAll* na odfarbenie všetkých entít a zastavenie všetkých aktívnych korutín(*Coroutine*) v rámci tejto triedy.

Testovanie

Testovanie prebieha pravidelne po každej väčšej aktualizácii (potrebu indikuje hlavný vývojár). Vzhľadom na povahu modulu testovanie prebieha manuálne, teda testerom, ktorý simuluje používateľa. Testovacie scenáre sú opísané v prílohe [Príloha A - Testovacie scenáre](#).
V tejto podobe boli dodané tímu JV pre vykonanie externého testovania.

XMI parser

Analýza

Ked' chceme zobraziť diagram tried v Unity (a následne animovať), ktoré sme vytvorili pomocou EA, máme viac možnosti. Jedna je taká, že z XMI súboru, ktorý sme vygenerovali z EA, načítame do Unity modulu. Bohužiaľ, sa to nedá priamo načítať do Unity modulu, preto je potrebné spraviť „most“ medzi EA a Unity, ktorý dodá potrebné dátá (napr. objekty, atribúty, metódy, smer šípky atď.) v požadovanom formáte.

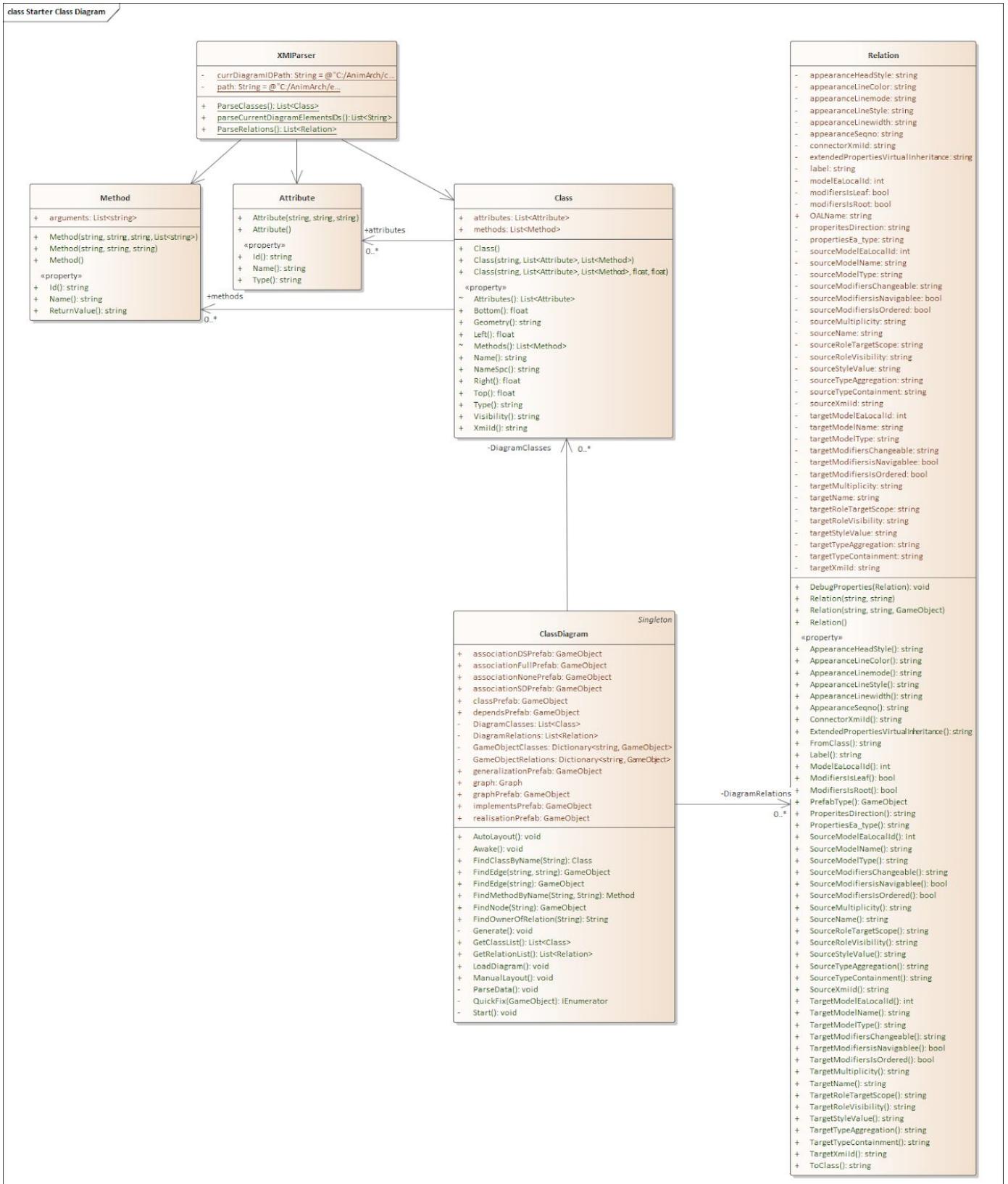
Alternatívou bolo vykonávať dopyt priamo nad databázou predstavujúcou EA projekt. Táto možnosť bola zamietnutá, nakoľko EA môže zmeniť databázovú schému v ďalších vydaniach, kdežto exporty ostávajú v konzistentnom XMI formáte.

Návrh

Po preskúmaní možností parsovania XML (a tým pádom aj XMI) súborov, sme pristúpili k využitiu vstavaných knižníc C#, nakoľko nepotrebujeme formulovať zložité dopyty. Pre naše účely je vhodnejšie prejst' celý súbor a uložiť si všetky nájdené údaje vo vhodnom formáte. Medzi zvažované alternatívy patrili napríklad XPath či XQuery.

Implementácia

Parser bol vyvíjaný v jazyku C#. Trieda *XMIParser* je zodpovedná za logiku parsovania dát, teda parsuje XML do C# dátovej štruktúry. Vie parsovať dátá o triedach, atribútoch, metódach a vzťahy medzi triedami. *XMIParser* dáva údaje do tried v balíku *ClassDiagram*, pričom informácie o triedach, atribútoch atď. sú uložené v triedach *Class*, *Attribute*, *Method*, *Relation*. Pseudokód pre túto časť možno nájsť v sekcií [XMI Parser](#).



Obrázok č. 5: XMI Parser - diagram tried

Testovanie

Parser bol overený vizuálne, po integrácii do Unity. Porovnali sme zdrojový diagram z EA s tým, čo sme dostali v Unity potom, ako sme vykonali parsovanie.

Okrem toho parsovanie podstúpilo záťažové testy, teda testovanie s veľkým počtom tried v diagrame, diagramov v projekte a podobne.

OAL parser

Analýza

ANTLR (Another Tool for Language Recognition) je generátor syntaktického analyzátora, ktorý sa dá využiť pri čítaní, spracovávaní, vykonávaní alebo prekladaní štruktúrovaného textu zo súboru. Takisto sa dá aj použiť pri vytváraní rôznych jazykov, nástrojov, frameworkov, čítačiek konfiguračných súborov, konvertorov kódu alebo aj analyzátorov rôznych formátov. Za prvé, ANTLR za pomocou gramatiky generuje parser/lexer, ktorý vie zostaviť štruktúrovaný strom a následne ho prehľadávať. Za druhé, ANTLR generuje runtime, ktorý je potrebný pre vygenerovaný parser/lexer.

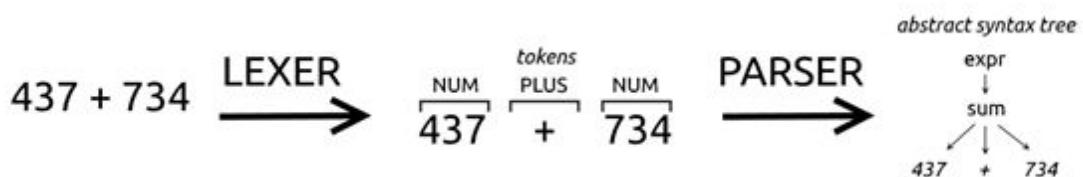
Prvý krok pri používaní ANTLR je nainštalovať a nastaviť potrebné veci k spusteniu ANTLR.

- Nainštalovať JAVA 1.6 alebo vyššiu
- Stiahnuť knižnicu ANTLR ([antlr-x.x.x-complete.jar](#))
- Pridať knižnicu ANTLR do systémového prostredia po CLASSPATH
- Vytvoriť krátke príkazy pre knižnicu ANTLR
 - doskey antlr4=java org.antlr.v4.Tool \$*
 - doskey grun =java org.antlr.v4.gui.TestRig \$*

Druhý krok je vytvoriť si gramatiku v súbore s príponou .g4, ktorá sa podobá syntaxou jazyku C. Gramatika obsahuje pravidlá, ktoré sa delia na dve časti (parser a lexer).

- Parser sa skladá zo súboru pravidiel syntaktického analyzátora, ktoré po spustení vygeneruje metódy na prehľadávanie stromu
- Lexer sa skladá zo súboru lexerových pravidiel. Slúži na tokenizáciu vstupu, ktorá sa následne využíva pri hľadaní zhôd v pravidlách parseru. Reprezentuje základné stavebné bloky vytváraného jazyka

Pri vytváraní parsovaného stromu zo vstupného textu sa najskôr vstupný text rozloží pomocou lexerových pravidiel na tokeny, ktoré sa následne využívajú pri hľadaní prislúchajúcich pravidiel v parseri.

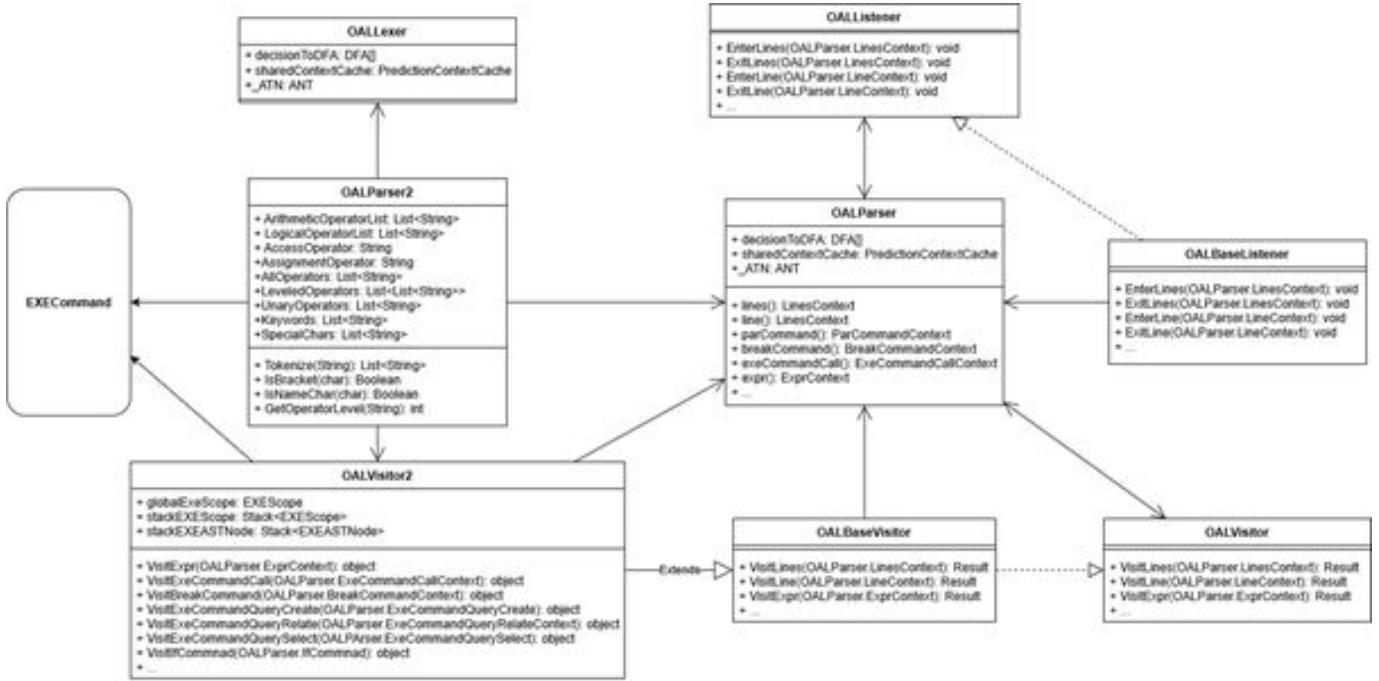


Obrázok č. 6: Postupnosť akcií spracovania textu pomocou ANTLR

Návrh

Bolo potrebné sa rozhodnúť ako budeme parsovať vstupný text. Ponúkali sa nám dve možnosti, jedna bola vytvoriť si vlastný parser a druhá použiť už existujúci. Nakol'ko tvorba vlastného parsera je komplikovaná a zdĺhavá, rozhodlo sa použiť existujúci parser ANTLR, ktorý poskytuje širokú funkcionality.

Dôvodom bola najmä udržateľnosť, nakoľko naša verzia jazyka OAL nie je definitívna a vyvíja sa.



Obrázok č. 7: Modul OAL parsovanie - diagram tried

Implementácia

Ako prvé bolo potrebné vytvoriť gramatiku našej verzia jazyka OAL, aby sme mohli pokryť všetky príkazy a výrazy, ktoré sa budú používať v našich animáciach. Gramatika je uvedená v časti [Príloha D - Gramatika našej verzie OAL](#).

Uvádzame ukážku OAL kódu:

relate subject to observer across R15;

select any young_dog from instances of Dog where selected.age < 5 OR 6;

create object instance myUserAccount of UserAccount;

unrelate subject from observer across R15;

delete object instance observer3;

y = 4;

*x = 15.0 * y;*

if (x == x AND x != y)

x = x;

end if;

```
while (x != x)
```

```
    x = x;
```

```
end while;
```

Po vygenerovaní tried je nutné ich vložiť do nášho projektu a prekonáť a implementovať metódy triedy *OALVisitor.cs*. Tie je potrebné upraviť tak, aby boli schopné zo syntaktického stromu získať údaje, ktoré sú následne využité pre vytvorenie vnútornej vykonateľnej reprezentácie OAL programu, vid' časť [OAL vykonávanie](#).

Upravovanie ANTLR metód funguje nasledovne:

- vytvoríme podtyp triedy *OALBaseVisitor*
 - `public class OALVisitor2:OALBaseVisitor<object>`
- prekonáme potrebné metódy
 - `public override object VisitExpr([NotNull] OALParser.ExprContext context)`
- implementujeme telá prekonaných metód. Získajú tak funkciu na vyskladanie syntaktického stromu jazyka OAL
 - `EXEASTNodeLeaf ast = new EXEASTNodeLeaf(context.GetChild(0).GetText());`
 - `stackEXEASTNode.Push(ast);`

Testovanie

Testovanie sme robili pomocou jednotkových testov, v ktorých sme sa snažili obsiahnuť všetky existujúce možnosti vstupov nášho programu. Súhranne sme vytvorili viac ako 100 jednotkových testov, v ktorých sa testujú možné vstupy. Postupovali sme systematicky a každé pravidlo sme otestovali s niekoľkými obmenami až po fázu, keď sme ich začali navzájom kombinovať. Vďaka nim sme odhalili niekoľko nedostatkov v našej gramatike, ktoré sme následne opravili.

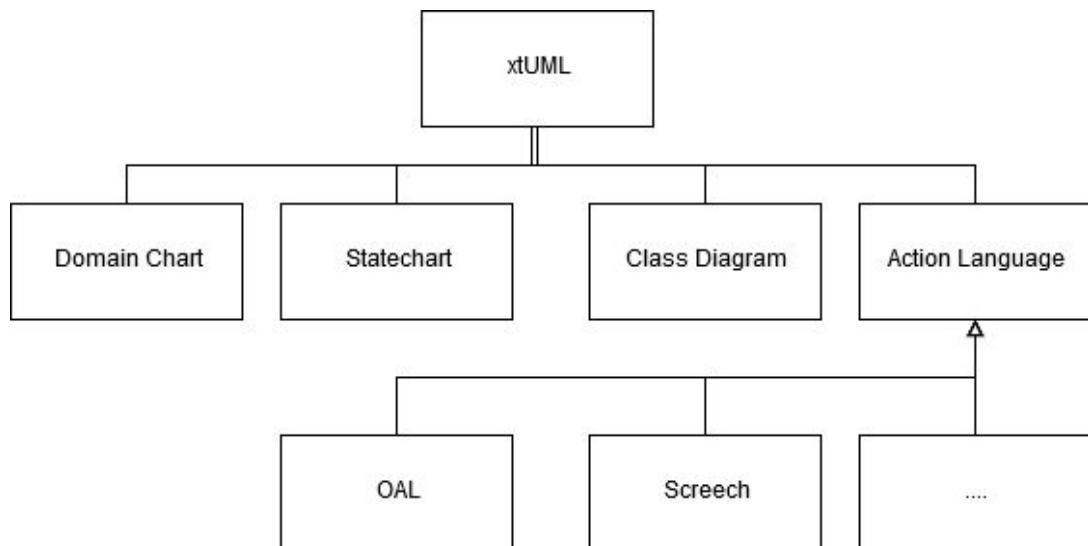
Naše Unit testovanie funguje na spôsobe, že vstupný text musí byť možné vysklaďať na základe vnútornej reprezentácie programu vytvorennej zo syntaktického stromu vytvoreného ANTLR.

OAL vykonávanie

Základnou úlohou tohto modulu je **vykonávanie animácie** (zmeny stavu programu, interakcia s GUI pri zvýrazňovaní volaní). Obsahuje tak komplexnú logiku, ktorej úlohou je zabezpečiť korektné vykonanie OAL programu.

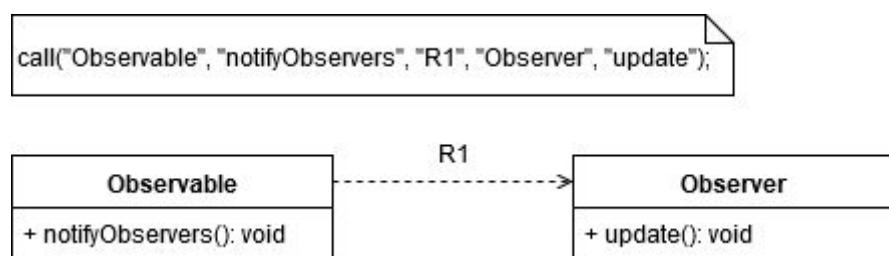
Analýza

Vytváranú animáciu je potrebné zapísat v konzistentnom formáte. Aby tento formát bol zrozumiteľný a prenositeľný, bol pre zápis animácie zvolený jazyk OAL, ktorý je súčasťou xtUML, resp. je špecializáciou jedného z jeho komponentov.



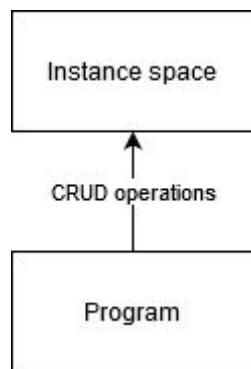
Obrázok č. 8: Komponenty xtUML

Animácia je postupnosť usporiadaných päťic tvorených názvom volajúcej triedy, názvom volajúcej metódy, pomenovaním vzťahu(hrany) medzi triedami, názvom volanej triedy a názvom volanej metódy.



Obrázok č. 9: Ukážka pomenovaných účastníkov jedného kroku

Pre využitie OAL príkazov je vhodné vytvoriť prostredia vykonávania tak, aby čo najviac rešpektovalo jazyk OAL podľa jeho manuálu⁴. V bežných programovacích jazykoch objekty vznikajú a zanikajú v scopoch. V jazyku OAL vytvorené objekty nezanikajú, dokým nie sú explicitne vymazané príkazom *delete*. Zanikajú len premenné na ne ukazujúce, objekt nezaniká ani keď naň neexistuje žiadna referencia. Referenciu naň možno znova získať dopytom nad priestorom inštancií tried.

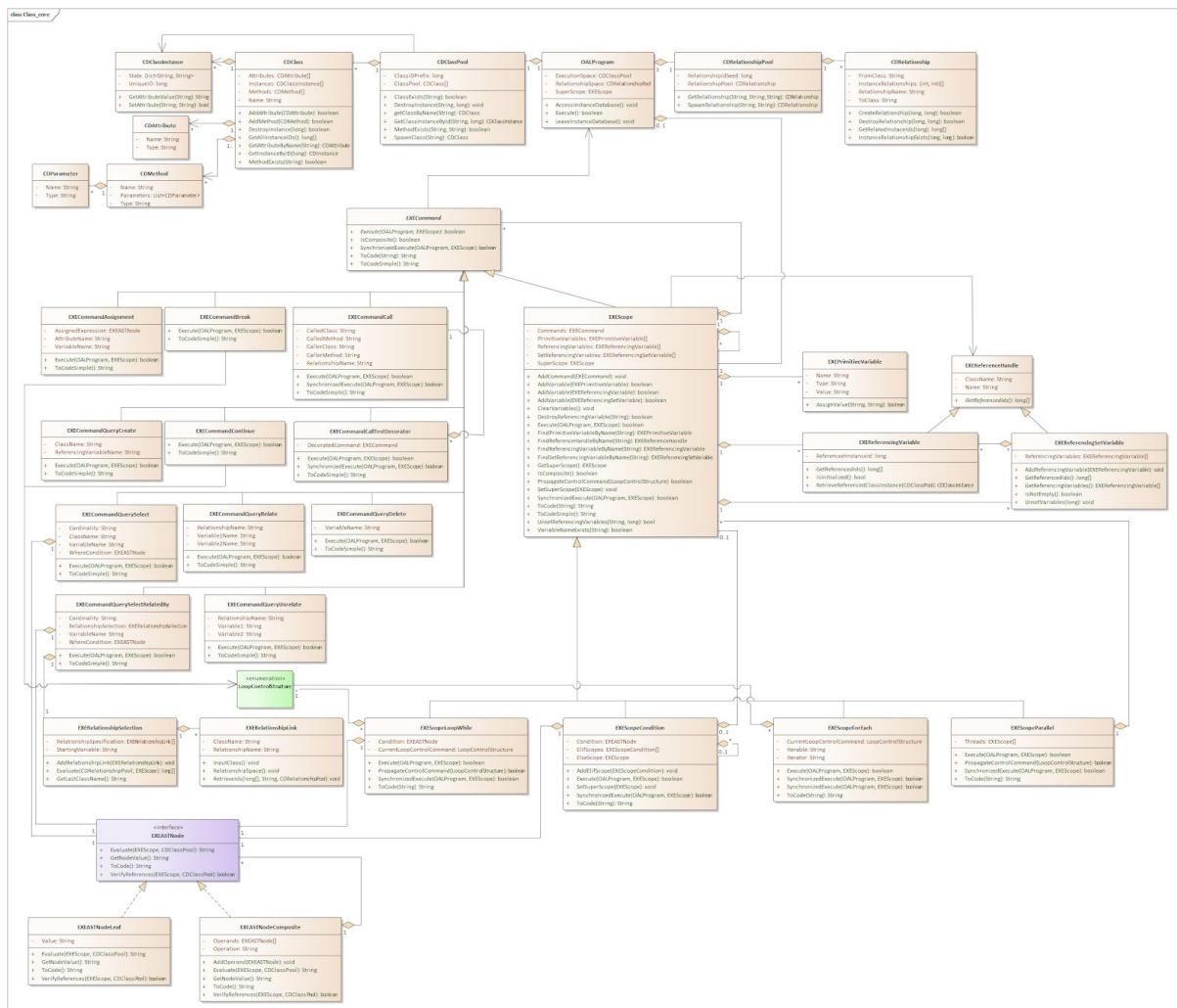


Obrázok č. 10: Závislosť programu na priestore inštancií

Návrh

Program sa skladá zo scopov tvoriačich strom. Koreňom je superscope, teda procedurálny obal programu. Scopy sa skladajú z príkazov, pričom aj scope môže byť príkaz. Vzniká tak strom, ktorý je reprezentovaný pomocou vzoru interpreter, kde dynamickými metódami sú *execute*, *synchronized_execute* a *preeexecute*. Podobne je riešená reprezentácia výrazov. Zložený výraz sa skladá z operátora a operandov. Operand môže byť zložený výraz alebo jednoduchý výraz.

⁴ <http://www.oaool.com/docs/OAL08.pdf>



Obrázok č. 11: OAL vykonávanie - diagram tried

Implementácia

Statický aj dynamický stav vykonávania programu je na **najvyššej úrovni** zachytený triedou *OALProgram* v atribútoch *ExecutionSpace* a *RelationshipSpace*. *ExecutionSpace* je typu *CDClassPool*, čo je trieda, ktorá v sebe uchováva zoznam všetkých tried (v zmysle používateľom modelovaných tried). Vytváranie nových tried v priestore by malo byť realizované na začiatku animácie, vždy volaním metódy *SpawnClass()* triedy *CDClassPool*.

Trieda je reprezentovaná triedou *CDClass*. Každá trieda si uchováva zoznam svojich inštancií (*CDClassInstance*), aby nad nimi mohli byť vykonávané dopyty. Tiež má zoznam atribútov (*CDAttribute*) a metód (*CDMethod*).

Trieda *CDClassInstance* obsahuje svoje unikátne ID (unikátnie naprieč všetkými triedami a inštanciami). Hodnoty atribútov sú uchovávané ako reťazce znakov v slovníku, kde kľúčom je názov atribútu.

Na nižšej úrovni vystupujú triedy *EXECommand* a *EXEScope*, ktoré implementujú spoločné rozhranie *EXECommandInterface* a tvoria tak vzor *Composite*, resp. *Interpreter*. Trieda *EXECommand* predstavuje elementárny príkaz, ako je volanie metódy, priradenie hodnoty do premennej, dopyt nad priestorom tried a podobne. Trieda *EXEScope* predstavuje blok kódu, pri ktorom vzniká *scope*⁵. Ním môže byť telo animácie (“superscope”), telo metódy alebo vetviaci príkaz ako *if/elif/else*, *while* a *foreach*. *EXEScope* je tvorený stavom svojich lokálnych premenných (*EXEPrimitiveVariable*, *EXEReferenceHandle*), rodičovským *scopom*(*EXEScope*) a zoznamom príkazov vo svojom tele (*EXECommandInterface*). Tiež obsahuje určenie svojho typu a atribúty charakterizujúce jednotlivé typy *scopu*. Možné typy *scopu* a ich charakteristické atribúty popisuje nasledujúca tabuľka.

Tabuľka č. 1: Typy *scopu*

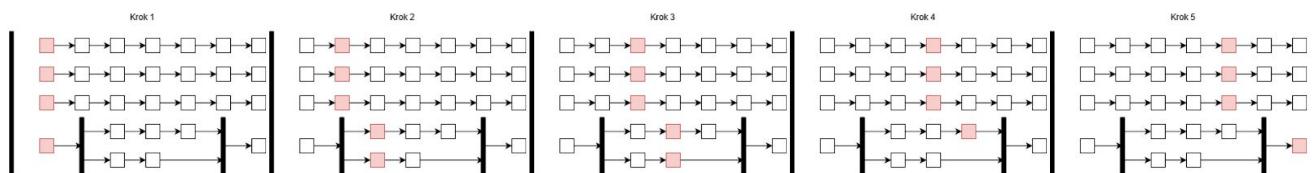
Typ <i>scopu</i>	Charakteristické atribúty
superscope	N/A
if	podmienka, pole elif scopov, else scope
while	podmienka
foreach	iterátor, iterovateľná kolekcia
parallel	pole vlákien

Na najnižšej úrovni sú triedy predstavujúce AST reprezentáciu výrazov. Triedy *ASTNodeComposite* a *ASTNodeLeaf* implementujú rozhranie *ASTNode*, tvoriac vzor *Interpreter*. Trieda *ASTNodeComposite* obsahuje operátor (+, -, ., *and*, *cardinality*,) a jeho operandy, reprezentované rozhraním *ASTNode*. Každý *ASTNodeComposite* vytvára vlastný strom, pričom operátor je jeho koreňom. Operandy sú typicky dva pre binárne operátory alebo jeden pre unárny operátor, avšak trieda počet operandov neobmedzuje. Trieda

⁵ žiadaj slovenský preklad nebol uznaný za adekvátny

ASTNodeLeaf je ďalej nevetvená, ďalej nepočítaná hodnota. Môže to byť číslo, reťazec, názov premennej, atribútu, metódy, triedy a podobne. Predstavuje list AST stromu výrazu.

Kvôli vykonaniu paralelného bloku je využívaný multithreading. Prístup k inštanciam tried a premenným je preto synchronizovaný mutexom. Kvôli animácii sú vlákna synchronizované. Vlákno sa vykonáva dokým v ňom nie je príkaz pre vysvetenie hrany. Vtedy vlákno počká na všetky ostatné vlákna. Synchronizačná konštrukcia je nami navrhnutá znovupoužiteľná bariéra s dynamicky sa meniacou veľkosťou. Vnorené paralelné bloky nemajú samostatnú synchronizáciu, synchronizujú sa spoločne s rodičovským paralelným blokom. Paralelný blok na začiatku registruje svoje vlákna v bariére, po skončení sú jednotlivé vlákna odregistrované.



Obrázok č. 12: Demonštrácia synchronizácie vlákin

V časti [Vykonávanie OAL kódu](#) možno nájsť pseudokód vykonávania vybraných metód tohto modulu.

Testovanie

Vzhľadom na povahu tohto modulu, najvhodnejším spôsobom testovania sú jednotkové testy. Testy sú sústredené na vykonávanie jednotlivých príkazov. Nakol'ko ostatné triedy poskytujú služby triedam predstavujúcim príkazy, týmto sú testované aj ostatné triedy. Celkovo je tento modul pokrytý takmer 600 testami o dĺžke 25 000 riadkov⁶. Testy boli tvorené v iteráciách podľa postupu implementácie. Každá trieda predstavujúca príkaz je testovaná na korektné správanie v bežných aj výnimkových stavoch. Masívne je testované aj počítanie výrazov, ktorého testy tvoria približne 25% všetkých testov.

⁶ merané pomocou Microsoft Visual Studio 19

Príručky

Prvá príručka obsahuje návod na inštaláciu pomocou inštalátora, ktorý vykoná inštaláciu automaticky. Druhá príručka je určená pre prípady, že by automatický inštalátor zlyhal, alebo pre používateľov, ktorí chcú addin nielen používať, ale aj modifikovať.

Oba inštalátory pracujú s adresárom *C:/AnimArch*. V prvom prípade sa doňho nainštaluje celý projekt, v druhom prípade slúži na výmenu XMI a ID medzi EA Addinom a Unity modulom. Je to nevyhnutná podmienka pre spustenie projektu. Tento adresár je možné zmeniť iba zásahom do zdrojového kódu Unity a taktiež EA Addinu.

Požiadavky:

- Windows 10
- Enterprise Architect 15 (môže byť aj trial verzia)
- Administrátorské práva k počítaču
- Disk C:/ s aspoň 100MB voľného miesta

Inštalačná príručka pomocou inštalátora

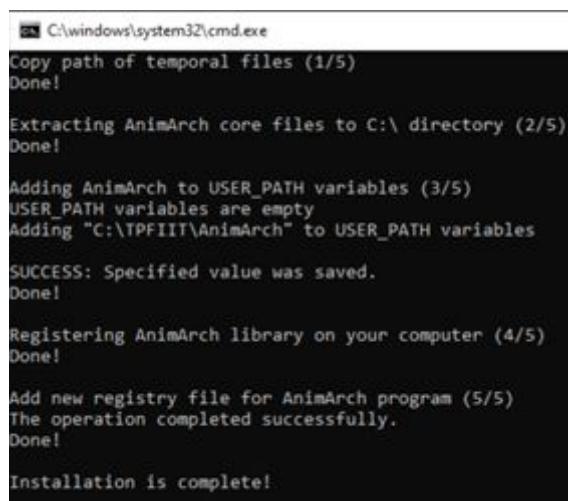
1. Spustite inštalátor ako správca

Inštalátor extrahuje zložku *AnimArch* do adresára *C:*.

Pridá novú premennú do *PATH* medzi používateľské premenné.

Zaregistruje náš EA Addin do vášho počítača pomocou *RegAsm.exe*.

Pridá nové súbory do registrov.



```
C:\windows\system32\cmd.exe
Copy path of temporal files (1/5)
Done!

Extracting AnimArch core files to C:\ directory (2/5)
Done!

Adding AnimArch to USER_PATH variables (3/5)
USER_PATH variables are empty
Adding "C:\TPFIIT\AnimArch" to USER_PATH variables
SUCCESS: Specified value was saved.
Done!

Registering AnimArch library on your computer (4/5)
Done!

Add new registry file for AnimArch program (5/5)
The operation completed successfully.
Done!

Installation is complete!
```

Obrázok č. 13: Úspešné ukončenie inštalácie

Inštalačná príručka pomocou Microsoft VS

V súčasnom stave projektu je inštalácia add-inu realizovaná cez vývojové prostredie Microsoft Visual Studio 2019 (VS). Je to z dôvodu, že addin je neustále vo vývoji, a pre jeho spustenie je potrebné množstvo nastavení, ktoré rieši VS automaticky. Pri vydaní finálnej verzie však bude vytvorený plnohodnotný inštalátor, ktorý pomocou skriptu vykoná všetky potrebné kroky na inštaláciu addinu bez VS. Momentálne by však vytvorenie takého skriptu bolo časovo náročné. Preto sa táto príručka zaoberá inštaláciou pomocou VS, ktorá má stále vysoký význam pre používateľov, ktorí chcú addin nielen používať, ale aj modifikovať.

1. Pred použitím príručky je potrebné vlastniť:

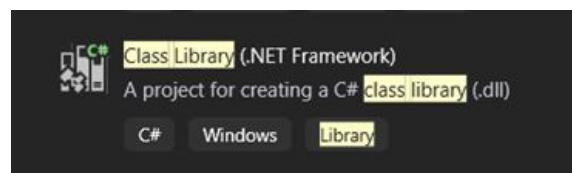
1. Zdrojový kód add-inu
2. Microsoft Visual Studio 2019 a .NET Framework verzia 4 a vyššia
3. Enterprise Architect 15
4. Administrátorské práva k počítaču
5. Operačný systém Windows 10
6. Build Unity modulu nazvaný *AnimeArch.exe* (obr.č.1), v akomkoľvek adresári, ktorý ale musí byť pridaný do *PATH-u*

AnimeArch_Data	30. 4. 2020 20:19	Priečinok súborov
MonoBleedingEdge	30. 4. 2020 20:16	Priečinok súborov
AnimeArch.exe	5. 3. 2020 14:47	Aplikácia 636 kB
UnityCrashHandler64.exe	5. 3. 2020 14:48	Aplikácia 1 069 kB
UnityPlayer.dll	5. 3. 2020 14:48	Rozšírenie aplikácie 25 163 kB

Obrázok č. 14: Build Unity modulu

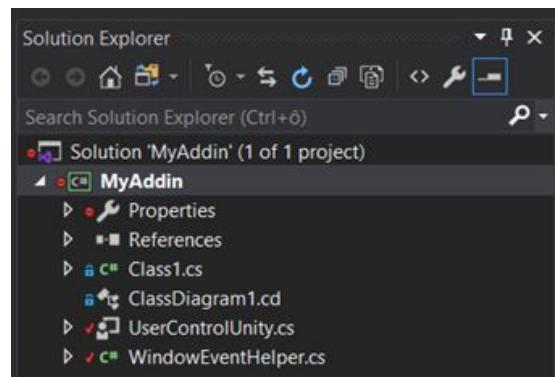
2. Vytvorenie projektu vo Visual Studiu 2019 a pridanie zdrojového kódu

Vzhľadom nato, že addin je vyvíjaný ako .dll knižnica a je potrebné ju registrovať do systému, aby bola pre EA viditeľná aj s poskytovanými funkciami, nie je možné kvôli unikátnym identifikátorom vo VS súboroch addin projektu celý projekt jednoducho importovať. Je potrebné založiť nový projekt ako C# .dll knižnica .NET frameworku:



Obrázok č. 15: Tvorba projektu vo VS

Projekt musí byť nazvaný **MyAddin** a vytváraná prvá trieda sa musí volať **Class1**. Po poslednom potvrdení na vytvorenie projektu by sa mala zobraziť prázdna trieda nazvaná *Class1* a namespace by mal byť *MyAddin*. Teraz je potrebné do tejto práznej triedy skopírovať zdrojový kód triedy *Class1.cs* nachádzajúcej sa v zložke MyAddin získanej z Github-u. Ďalej je potrebné označiť v tejto zložke súbory: *UserControlUnity.cs*, *UserControlUnity.Designer.cs*, *UserControlUnity.resx*, *DebugConsole.cs*, *WindowEventHelper.cs*, *UnityWindowResizer.cs* a pretiahnuť ich do VS napravo do časti Solution Explorer:



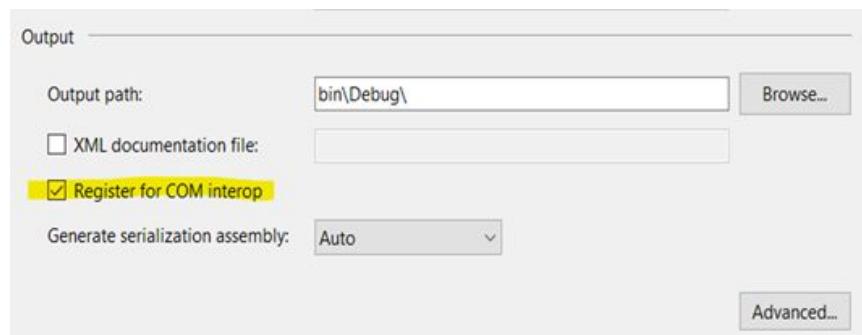
Obrázok č. 16: Pridanie tried do Solution Explorer

Ak bol tento krok vykonaný správne, VS by malo automaticky pridať tieto súbory k projektu.

3. Nastavenie projektu

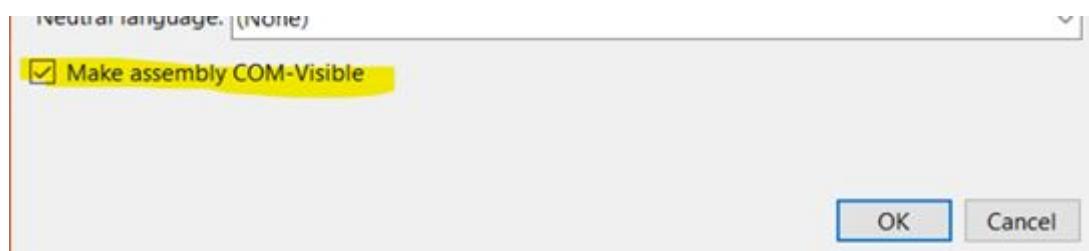
Pred spustením vloženého kódu do projektu je potrebné projekt najskôr nastaviť. Nasledujúce kroky je možné vykonať v ľubovoľnom poradí:

1. Project → MyAddin Properties... → Build → vyscrollovať dole a kliknúť na *Register for COM interop*



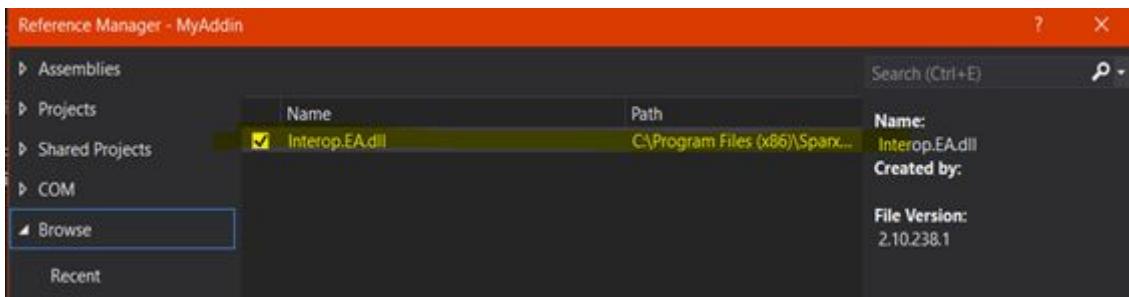
Obrázok č. 17: Registrovanie projektu na COM interop

2. Project → MyAddin Properties... → Application → Assembly Information... → kliknúť na *Make assembly COM-visible* a potvrdiť OK



Obrázok č. 18: Zviditeľnenie projektu na COM

3. Project → Add Reference... → Browse → a treba nájsť v počítači EA API knižnicu *Interop.EA.dll* zvyčajne umiestnenú v *Program Files(x86)\Sparx Systems\EA* a pridať ju do projektu.



Obrázok č. 19: Pridanie .dll knižnice potrebnej pre komunikáciu s EA

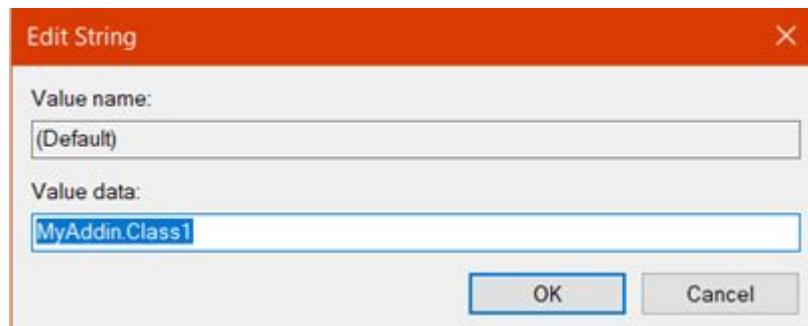
4. Project → Add Reference... → Assemblies → v pravom hornom rohu vyhľadať knižnicu Windows.Forms.dll a pridať ju do projektu

Toto by mali byť všetky nastavenia, ktoré je potrebné vykonať vo VS. Otestovať to je možné kliknutím na horné tlačidlo Štart, ktoré spraví build projektu a pokúsi sa ho spustiť. Ak bol build úspešný, na modrej spodnej lište vľavo je možné vidieť napísané *Build Successful*. S najväčšou pravdepodobnosťou úspešný nebude, je však dôležité, aby neboli vypísané žiadne chyby v zdrojom kóde, pretože tie by znamenali, že budť zdrojové súbory neboli správne pridané k projektu, alebo neboli správne vybrané referenčné knižnice. V každom prípade je teraz potrebné VS zavrieť.

4. Pridanie Addinu do registrov OS

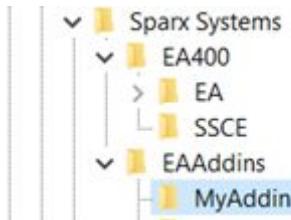
Enterprise Architect pri štarte kontroluje dostupnosť addinu pomocou registrov, a preto je potrebné do nich referenciu na addin vložiť.

1. Windows tlačidlo na klávesnici + r, napísat *regedit* a stlačiť enter
2. Nájsť cestu *HKEY_CURRENT_USER\Software\Sparx Systems\EAAddins*
3. Kliknúť pravým na zložku *EAAddins* → *New* → *Key*
4. Vytvorí sa nová zložka v *EAAddins*, ktorú treba nazvať *MyAddin*
5. Vyplniť defaultný kľúč ako *Namespace.NázovClassy*, v tomto prípade *MyAddin.Class1*



Obrázok č. 20: Vytvorenie kľúča v registri

V niektorých prípadoch sa môže stať, že zložka *EAAAddins* v registroch neexistuje. V tomto prípade ju stačí jednoducho najsíkôr vytvoriť a pokračovať krokmi 3, 4 a 5. Nakoniec by mal register vyzerat nasledovne:



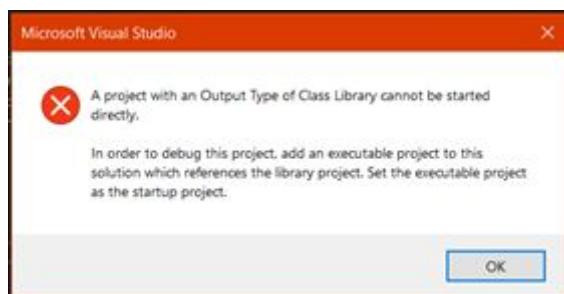
Obrázok č. 21: Vytvorenie priečinka v registroch

Hodnota kľúča v zložke *MyAddin* je na predchádzajúcim obrázku. Ak vyzerá byť všetko v poriadku, register je možné zavrieť.

5. Build addinu vo VS

Na posledný krok je potrebné spustiť Visual Studio ako správca. Spúšťať VS ako správca je potrebné vždy, ak sa plánujú vykonať zmeny v kóde, inak build projektu nezbehne správne. V samotnom projekte už nie je potrebné nič nastavovať ani meniť (ak bol správne vykonaný krok 3), stačí stlačiť zelené tlačidlo Start, prípadne Build v hornej lište VS. Ak build zbehol bez chýb a varovaní, je s najväčšou pravdepodobnosťou všetko tak, ako má byť. Chybová hláška (obr.č.9) oznamujúca, že projekt nie je možné samostatne spustiť, je v poriadku a netreba si ju všímať, keďže sa jedná o knižnicu, ktorá funguje iba v rámci EA. Ak svieti v modrej spodnej lište naľavo *Build successful*, je možné VS zavrieť, keďže Addin by mal byť

v tomto momente úspešne nainštalovaný. Ak build nezbehol alebo zbehol s varovaniami, je potrebné tento problém vyriešiť. Najčastejšia príčina je nespustenie VS ako správca. Druhou najčastejšou príčinou je spúšťanie buildu addinu, ak je súčasne spustený aj EA. EA je vždy potrebné pred spustením buildu zavrieť a ideálne niekoľko sekúnd počkať, kým sa úspešne ukončí. V oboch prípadoch chyby a varovania VS vyzerajú rovnako a hovoria o tom, že nemohli pristúpiť k existujúcemu .dll addinu, prípadne podobné problémy s prístupom.



Obrázok č. 22: VS chybová hláška

6. Overenie správnej inštalácie addinu

Nainštalovaný addin je možné overiť spustením EA, otvorením akéhokoľvek projektu s diagramom a na danom diagrame kliknúť pravým tlačidlom myši. Následne je potrebné vybrať možnosť *Specialize* a medzi možnosťami musí byť na výber *Unity Addin*. Ak sa *UnityAddin* medzi možnosťami nezobrazuje, neboli pravdepodobne nainštalovaný správne a je potrebné kroky inštalácie zopakovať a odhaliť chybu. Niekedy môže pomôcť reštart EA, ďalší build vo VS, prípadne reštart počítača.

Používateľská príručka

Používateľská príručka obsahuje návod pre spustenie a popis všetkých obrazoviek a funkcionality, ktoré sú v aktuálnom stave projektu pre používateľa prístupné. Predpokladom je správna inštalácia Addinu podľa inštalačnej príručky.

1. Nastavenie Unity modulu

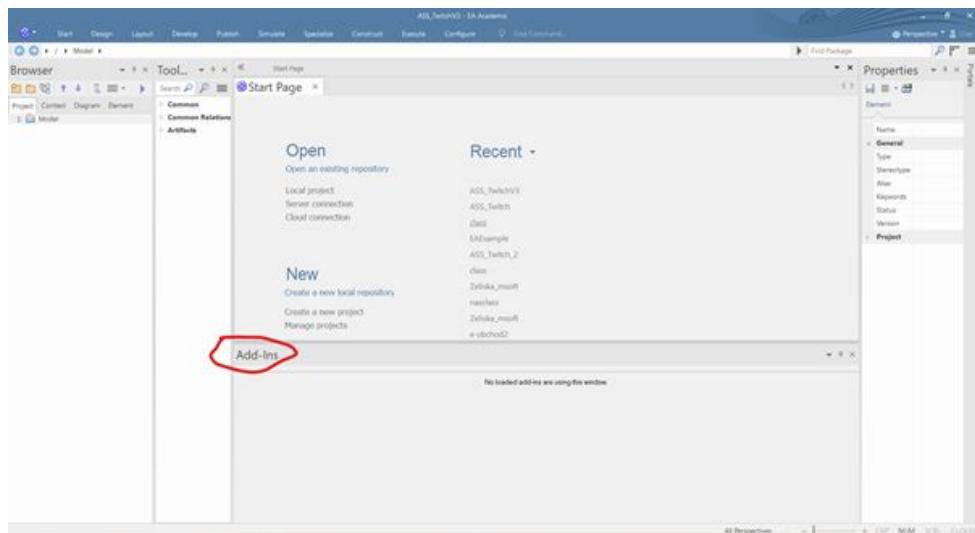
Poskytnutá funkcialita zobrazovania diagramov tried v Unity je implementovaná ako modul. Tento modul je vyexportovaný z vývojového prostredia Unity a do určitej miery samostatne spustiteľný aj bez EA. Samotná inštalácia Unity enginu na používateľských počítačoch nie je nutná, avšak dôležité je umiestnenie tohto modulu. Aby nenastávali problémy s umiestnením, stačí pridať cestu k spustiteľnému modulu *AnimeArch.exe* do PATH-u systému. Na operačnom systéme Windows 10 je postup nasledovný:

1. Štart → vyhľadávanie a kliknutie na “Edit the system environment variables”
2. Kliknúť na “Environment variables...”
3. Medzi “User variables for -yourUserName-” kliknúť na Path → Edit...
4. Kliknúť na New → vložiť cestu k *AnimeArch.exe* → kliknúť OK

Odporúča sa použitie inštalátora, ktorý vykoná všetky kroky automaticky.

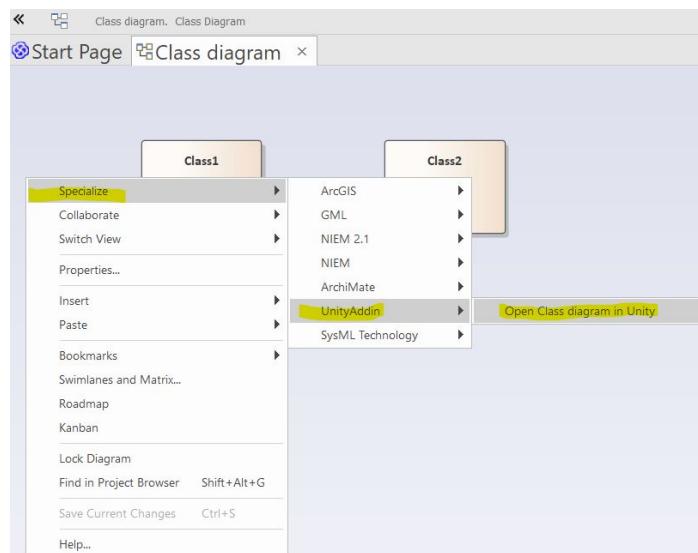
2. Spustenie addinu

Pre spustenie addinu je najskôr potrebné spustiť EA. Add-Ins okno možno zobraziť kliknutím na *Specialize* v hornej lište a následne na *Windows*. Spustením addinu sa však okno zobrazí automaticky aj v prípade, že je skryté.



Obrázok č. 23: Add-ins okno v EA

Ďalej je potrebné otvoriť akýkoľvek diagram tried. Add-in menu bude prístupné aj pri iných typoch diagramov, ale nebude klikateľné. Menu addinu sa dá zobrazíť pravým kliknutím kdekoľvek v otvorenom diagrame a vybrať možnosť *Specialize → UnityAddin*.

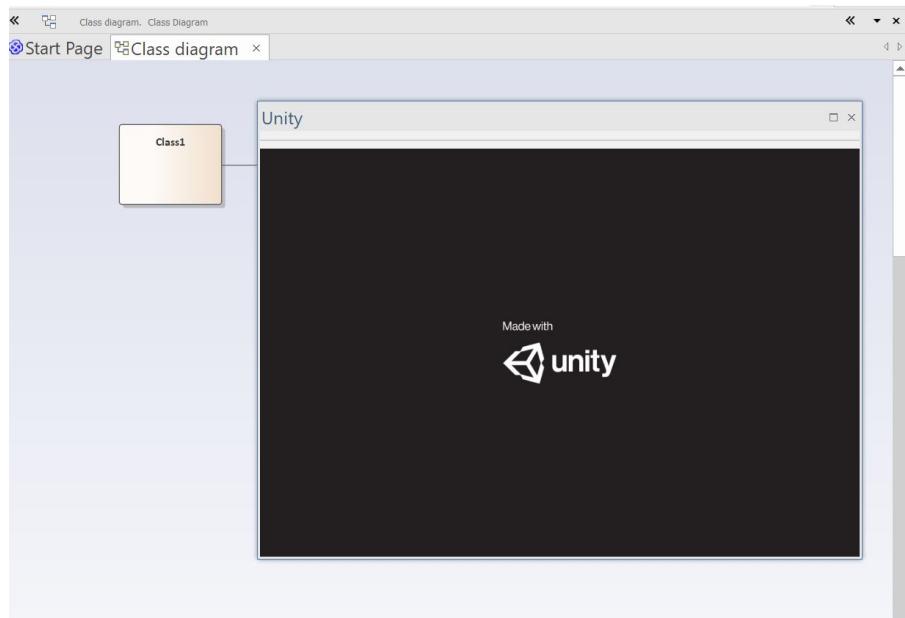


Obrázok č. 24: Spustenie add-inu cez EA

3. Zobrazenie Unity v Add-Ins okne

Pre zobrazenie diagramu tried v Unity je potrebné mať otvorený diagram tried a kliknúť pravým tlačidlom kdekoľvek v otvorenom diagrame a vybrať možnosť *Specialize →*

UnityAddin → Open class diagram in Unity. Malo by sa začať načítavať Unity ako na obrázku nižšie.

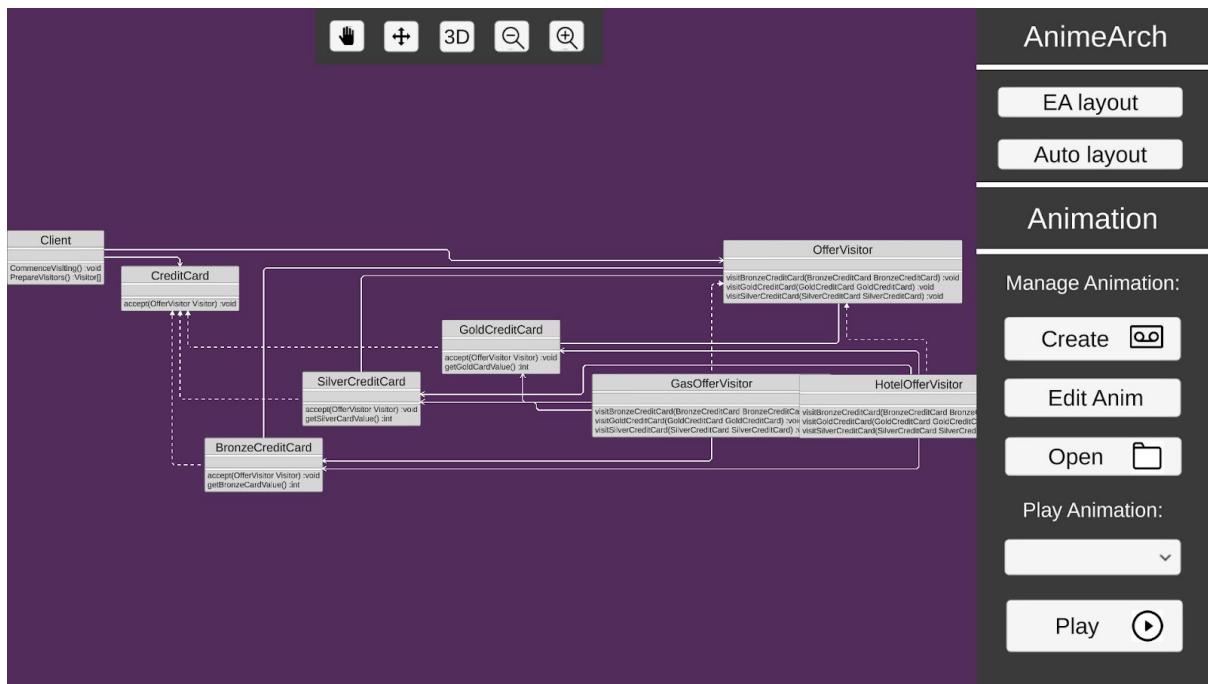


Obrázok č. 25: Unity v okne EA

Veľkosť zobrazeného Unity okna je možné dynamicky meniť potiahnutím jeho hrán. Maximalizácia je možná kliknutím na ikonu štvorca v jeho pravom hornom rohu. Po opäťovnom kliknutí na tento štvorec sa okno vráti do pôvodnej veľkosti. Krížik vedľa štvorca má pôvodné správanie EA, čiže okno sa iba skryje a neukončí sa! Add-Ins okno je možné znova zobraziť kliknutím na *Specialize* v hornej lište a následne na *Windows*. Je povolené mať zobrazené iba jedno Unity okno súčasne. Samotné Unity okno sa správa ako akékoľvek iné okno v EA, čiže je možné ho kdekoľvek pripínať. Pre správne ukončenie addinu je potrebné kliknúť pravým tlačidlom myši kdekoľvek v diagrame a vybrať *Specialize → UnityAddin → ExitUnity*.

4. Diagram tried v Unity

Po načítaní Unity v Add-Ins okne sa zobrazí nasledovná obrazovka, v ktorej je zobrazená verná kópia otvoreného diagramu z EA.



Obrázok č. 26: Diagram tried v Unity

So zobrazeným diagramom tried je možné manipulovať pomocou ovládacieho panelu naľavo.



- umožňuje uchytiať a posunúť triedu v priestore



- umožňuje uchytiať a posunúť celý diagram v priestore



- prepnutie zobrazenia do 3D režimu



- oddialenie celého diagramu



- priblíženie celého diagramu



- umožňuje manuálne prefarbovanie tried zvolenou farbou

EA layout - usporiadanie tried a konektorov podľa EA

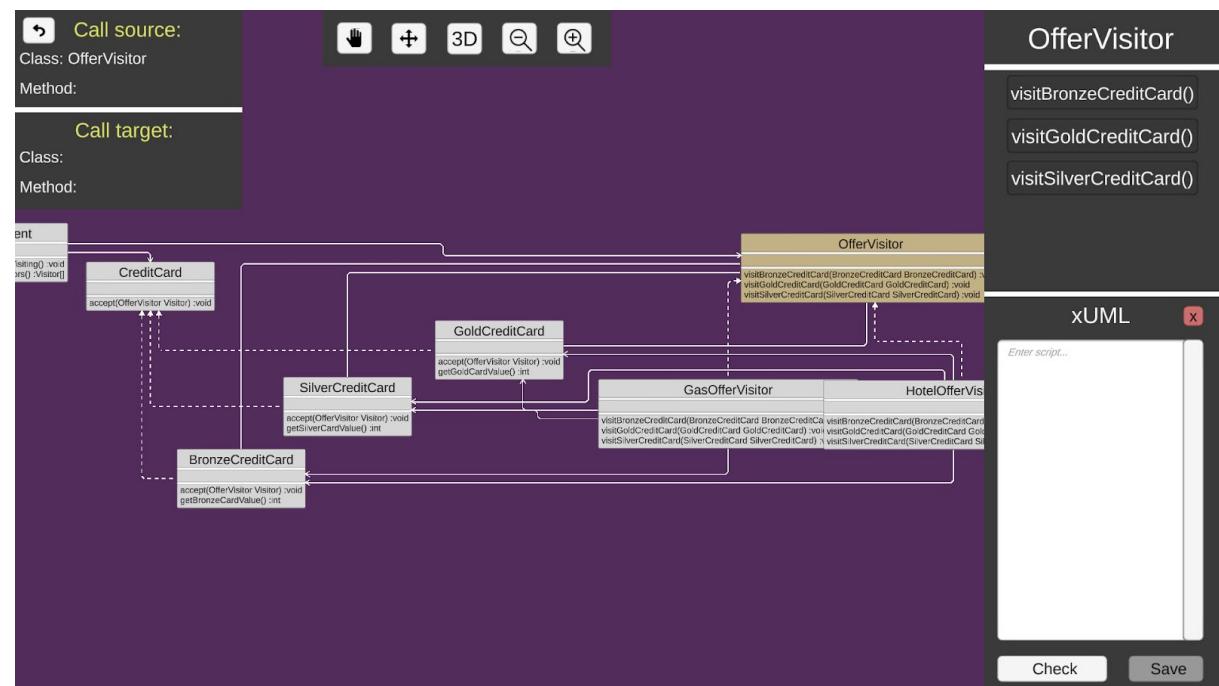
Auto layout - automatické usporiadanie tried a konektorov

Create - umožňuje vytvorenie animácie, zobrazí sa panel pre vytváranie animácie

Animácia sa vytvára klikaním na triedy. Najprv je potrebné označiť triedu v ktorej bude krok



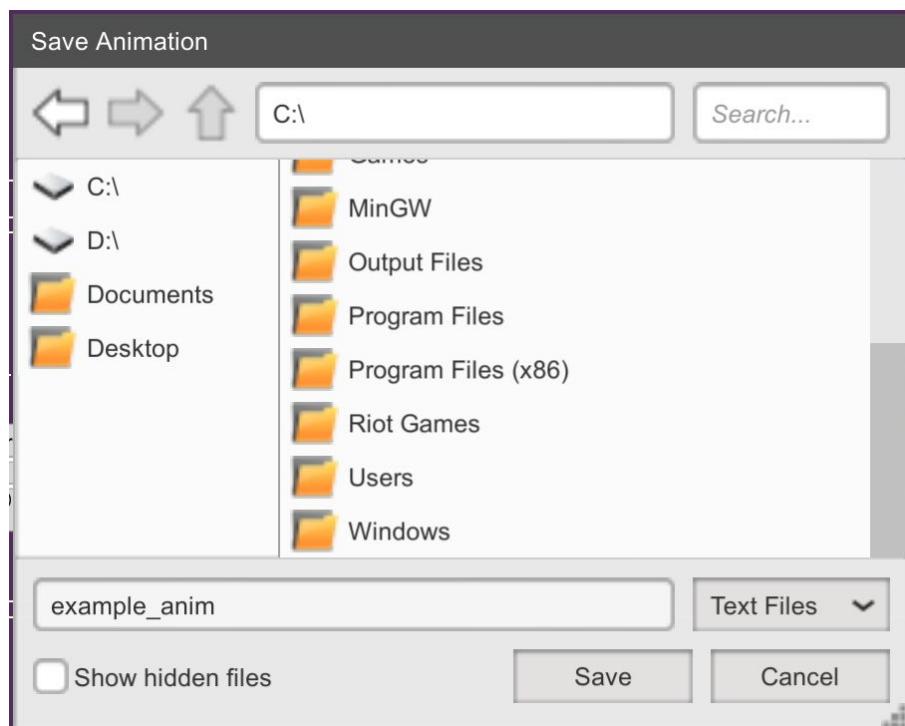
Následne sa vpravo hore zobrazia metódy triedy, z ktorých je potrebné si vybrať, ktorú chceme použiť. Potom je potrebné rovnakým spôsobom zvoliť aj cieľovú triedu kroku.



Obrázok č. 27: Ukážka obrazovky pre vytváranie animácie

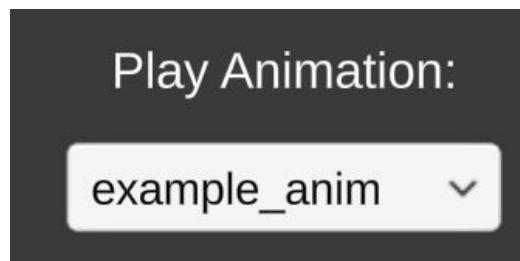
Check - kontrola zadaného kódu, či sa dá vykonat

Save - umožní uloženie animácie a výber miesta kam má byť uložená



Obrázok č. 28: Ukážka ukladania animácie

Play animation - umožní zvolenie uloženej animácie:

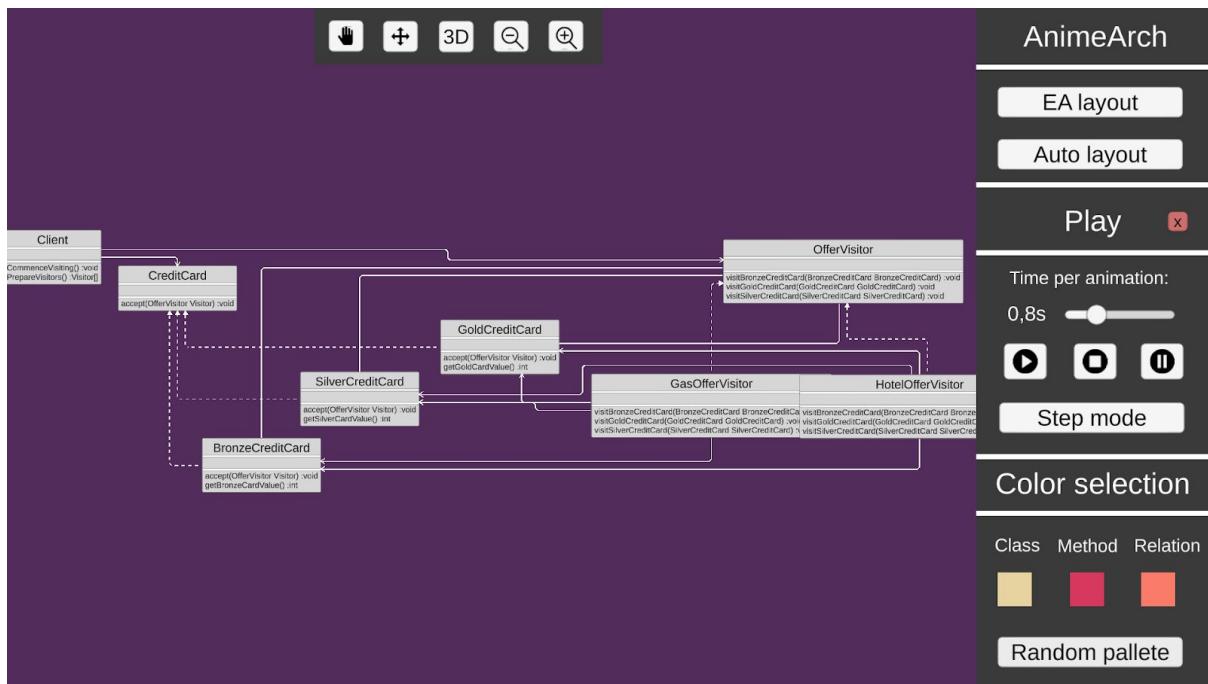


Obrázok č. 29: Výber uloženej animácie

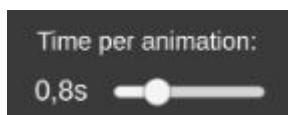
Edit Anim - umožňuje zmeniť kód vybranej animácie

Open - otvorí uloženú animáciu do formátu na obr.č.24

Play - umožní prehranie a ovládanie zvolenej animácie, otvorí panel na prehrávanie



Obrázok č. 30: Obrazovka s panelom pre ovládanie animácie



- posuvník umožňuje nastavenie rýchlosťi jedného kroku v animácii



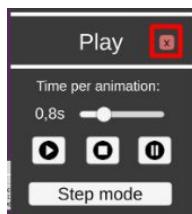
- spustenie animácie



- pauza v prehrávaní animácie

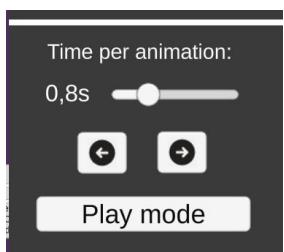


- zrušenie prehrávania animácie



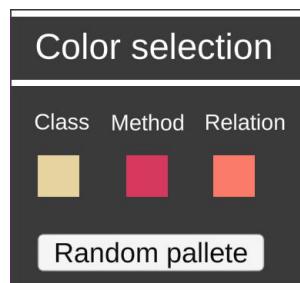
- červeným krížikom vypneme mód prehrávani a vrátíme sa do pôvodného panelu

Step mode - umožní manuálne krokovanie animácie, zmení sa panel pre prehrávanie animácie



Animácia sa krokuje šípkami, šípkou doprava sa posunieme o krok vpred, šípkou doľava o krok vzad.

Tlačidlom Play mode sa vrátim do pôvodného módu prehrávanie animácie



Color selection umožňuje prefarbenie tried, metód a vzťahov

Random pallete prefarbí náhodne triedy, metódy a vzťahy

Technická dokumentácia

Za súčasť technickej dokumentácie považujeme už uvedené diagramy tried v časti Diagramy tried a diagramy aktivít a sekvencií v kapitolách venovaných jednotlivým modulom v časti Moduly systému.

Ďalej uvádzame snippety kódu na vysokej úrovni logiky, aby sme demonštrovali vykonávanie zaujímavých častí aplikácie.

EA Add-in

Hlavná trieda add-inu vyexportovanie XMI a pridanie Unity okna do EA:

```
// generate XMI of the current package of the currently opened diagram
Repository.GetProjectInterface().ExportPackageXMI(Repository.GetPackageByID(Repository.GetCurrentDiagram().PackageID).PackageGUID,
    EnumXMIType.xmiEA21, 1, -1, 1, 0, XMIPath);

// EA calls constructor of UserControl Unity like this
ucu = Repository.AddWindow("Unity", "MyAddin.UserControlUnity");
// show Unity window in EA
Repository.ShowAddinWindow("MyAddin.UserControlUnity");
```

Obrázok č. 31: Ukážka kódu - EA Add-in 1

Spustenie Unity modulu ako dcérskeho procesu v konštruktore UserControlUnity triedy:

```
// prepare and start Unity process
unityProcess = new Process();
unityProcess.StartInfo.FileName = "TpFiiit2019.exe";
unityProcess.StartInfo.Arguments = "-parentHWNd " + this.Handle.ToInt32() + " " + Environment.CommandLine;
unityProcess.StartInfo.UseShellExecute = true;
unityProcess.StartInfo.CreateNoWindow = true;
unityProcess.StartInfo.WindowStyle = ProcessWindowStyle.Maximized;
unityProcess.Start();
```

Obrázok č. 32: Ukážka kódu - EA Add-in 2

Registrovanie sa na odchytávanie window eventov v triede WindowEventHelper:

```
public void Subscribe(Process unityProcess, IntPtr unityHWNd)
{
    m_hook = SetWinEventHook(10, 99999, m_target, m_winEventDelegate, m_processId, m_threadId, 0);
    this.unityProcess = unityProcess;
    this.unityHWNd = unityHWNd;
}
```

Obrázok č. 33: Ukážka kódu - EA Add-in 3

Prispôsobenie Unity okna ak je vyvolaný vhodný event :

```
var location = GetWindowLocation();
MoveWindow(unityHWND, 0, 0, location.Right - location.Left, location.Bottom - 50 - location.Top, true);
```

Obrázok č. 34: Ukážka kódu - EA Add-in 4

Vykonávanie OAL kódu

V module sa nachádza mnoho tried predstavujúcich konkrétné príkazy. Zaujímavú logiku majú zložené príkazy, akými sú podmienka, cyklus či paralelný blok. Uvádzame preto pseudokód pre zložené príkazy, pričom s paralelným blokom súvisí aj znovupoužiteľná bariéra s dynamicky sa meniacou veľkosťou, ktorá je tiež demonštrovaná pseudokódom.

EXEScope

Trieda *EXEScope* predstavuje jednoduchý scope, teda priestor vykonávania s postupnosťou príkazov. Uvádzame pseudokód zjednodušenej verzie tejto triedy, len s podstatnými atribútmi a metódami

```
CLASS EXEScope EXTENDS EXECommand

    EXEScope SuperScope = NULL

    EXECommand[] Commands = EMPTY_ARRAY

    EXEPrimitiveVariable[] PrimitiveVariables = EMPTY_ARRAY

    EXEReferencingVariable[] ReferencingVariables = EMPTY_ARRAY

    EXEReferencingSetVariable[] SetReferencingVariables = EMPTY_ARRAY

    FUNCTION Execute(OALProgram OALProgram, EXEScope Scope)

        BOOL success = TRUE

        FOREACH Command in Commands

            success = Command.Execute(OALProgram, this)

            IF success == FALSE

                BREAK

            END IF

        END FOR

        RETURN success

    END FUNCTION

END CLASS
```

EXEScopeCondition

Trieda *EXEScopeCondition* predstavuje príkaz if. Obsahuje aj prípadné elify a else. Zobrazujeme pseudokód tejto triedy so základnými atribútmi a hlavnou metódou.

```
CLASS EXEScopeCondition EXTENDS EXEScope

    EXEASTNode Condition = NULL

    EXEScopeCondition[] ElifScopes = EMPTY_ARRAY

    EXEScope ElseScope = NULL

    FUNCTION Execute(OALProgram OALProgram, EXEScope Scope)

        result = TRUE

        IF Condition == NULL

            RETURN FALSE

        END IF

        evaluate Condition

        IF evaluation successful

            IF Condition is TRUE

                FOREACH Command in Commands

                    Command.Execute(OALProgram, THIS)

                    IF execution failed

                        RETURN FALSE

                    END IF

                END FOR

            END IF

            ELSE

                FOREACH ElifScope in ElifScopes

                    IF ElifScope.Condition is TRUE

                        FOREACH Command in ElifScope.Commands
```

```

        Command.Execute(OALProgram, THIS)

        IF execution failed

            RETURN FALSE

        END IF

        END FOR

        BREAK

        END IF

        END FOR

        IF no scope was executed

            ElseScope.Execute()

        END IF

        END IF

        END IF

    END FUNCTION

END CLASS

```

EXEScopeLoopWhile

EXEScopeLoopWhile predstavuje while cyklus. Má určený maximálny počet iterácií.

```

CLASS EXEScopeLoopWhile EXTENDS EXEScope

    EXEASTNode Condition = NULL

    LoopControlStructure CurrentLoopControlCommand = NONE

    FUNCTION Execute(OALProgram OALProgram, EXEScope Scope)

        success = TRUE

        iterationCounter = 0

        WHILE Condition IS TRUE AND iterationCounter < iterationCap

            FOREACH Command in Commands

```

```

success = Command.Execute(OALProgram, Scope)

IF success == FALSE

    RETURN FALSE

END IF

END FOR

iterationCounter++

IF CurrentLoopControlCommand IS BREAK

    BREAK

END IF

IF CurrentLoopControlCommand IS CONTINUE

    CONTINUE

END IF

END WHILE

END FUNCTION

END CLASS

```

EXEScopeForEach

EXEScopeForEach predstavuje foreach cyklus.

```

CLASS EXEScopeForEach EXTENDS EXEScope

STRING IteratorName = NULL

STRING IterableName = NULL

LoopControlStructure CurrentLoopControlCommand = NONE

FUNCTION Execute(OALProgram OALProgram, EXEScope Scope)

    iterator = create iterator

    iterable = retrieve iterable

    IF init IS successful

```

```

FOREACH item IN iterable

    assign item to iterator

    FOREACH Command IN Commands

        success = Command.Execute(OALProgram, Scope)

        IF success == FALSE

            RETURN FALSE

        END IF

        IF CurrentLoopControlCommand IS NOT NONE

            BREAK

        END IF

    END FOR

    IF CurrentLoopControlCommand IS BREAK

        BREAK

    END IF

    IF CurrentLoopControlCommand IS CONTINUE

        CONTINUE

    END IF

    END FOR

END IF

END FUNCTION

END CLASS

```

EXEScopeParallel

EXEScopeParallel predstavuje paralelný blok, ktorý obsahuje niekoľko vlákien. Pre synchronizáciu vlákien využíva nižšie opísanú triedu [EXEThreadSynchronizer](#). Pozorovať ju možno ako atribút *ThreadSyncer* triedy *OALProgram*.

CLASS EXEScopeParallel

```

EXEScope[] Threads = EMPTY_ARRAY

MONITOR ThreadEndSyncer = NEW MONITOR

INTEGER ActiveThreadCount = 0

FUNCTION Execute(OALProgram OALProgram, EXEScope Scope)

    OALProgram.ThreadSyncer.RegisterThread(LENGTH OF Threads)

    success = TRUE

    SYNCHRONIZE ON ThreadEndSyncer

        ActiveThreadCount = LENGTH OF Threads

    END SYNCHRONIZE

    OALProgram.ThreadSyncer.UnregisterThread()

    FOREACH Thread IN Threads

        START THREAD

            thread_success = Thread.Execute(OALProgram, THIS)

            OALProgram.ThreadSyncer.UnregisterThread()

            SYNCHRONIZE ON ThreadEndSyncer

                success = success AND thread_success

                ActiveThreadCount--

                IF ActiveThreadCount == 0

                    NOTIFY ALL ON ThreadEndSyncer

                END IF

            END SYNCHRONIZE

        END THREAD

    END FOR

    SYNCHRONIZE ON ThreadEndSyncer

    WHILE ActiveThreadCount > 0

        WAIT ON ThreadEndSyncer

    END WHILE

END SYNCHRONIZE

```

```

OALProgram.ThreadSyncer.RegisterThread(1)

RETURN success

END FUNCTION

END CLASS

```

EXEThreadSychronizer

Predstavuje znovupoužiteľnú bariéru s dynamicky sa meniacou veľkosťou. Uvádzame jej pseudokód, použiteľný pre ľubovoľný problém.

```

CLASS ReusableResizableBarrier

MONITOR syncer = NEW MONITOR

INTEGER barrierSize = 0

INTEGER freeBarrierSlotCount = 0

INTEGER inBarrierCount = 0

BOOL blocked = FALSE

INTEGER pendingSizeChange = 0

FUNCTION registerThread(UNSIGNED INTEGER count)

    SYNCHRONIZE ON syncer

        pendingSizeChange += count

    END SYNCHRONIZE

END FUNCTION

FUNCTION unregisterThread()

    SYNCHRONIZE ON syncer

        pendingSizeChange--

        NOTIFY ALL ON syncer

    END SYNCHRONIZE

END FUNCTION

FUNCTION requestStep()

    SYNCHRONIZE ON syncer

```

```

WHILE blocked == TRUE

    WAIT ON syncer

    END WHILE

    CALL performThreadCountChange

    inBarrierCount++

    freeBarrierSlotCount--

    WHILE freeBarrierSlotCount > 0

        CALL performThreadCountChange

        WAIT ON syncer

        END WHILE

        IF freeBarrierSlotCount > 0

            blocked = TRUE

            NOTIFY ALL ON syncer

        END IF

        InBarrierCount--

        IF inBarrierCount == 0

            blocked = FALSE

            freeBarrierSlotCount = threadCount

            NOTIFY ALL ON syncer

        END IF

    END SYNCHRONIZE

END FUNCTION

FUNCTION performThreadCountChange()

    IF pendingSizeChange != 0

        threadCount += pendingSizeChange

        freeBarrierSlotCount += pendingSizeChange

        pendingSizeChange = 0

    END IF

```

```

IF threadCount < 0 OR freeBarrierSlotCount < 0

    RAISE ERROR

END IF

END FUNCTION

END CLASS

```

XMI Parser

XMIParser získava údaje o diagrame tried z XMI exportu získaného z EA. Nasledujúca ukážka demonštruje získavanie údajov o vzťahoch medzi triedami.

```

FUNCTION ParseRelations()
    Relation[] connectorClassesList = EMPTY_ARRAY
    XmlDocument xmlDoc = NEW XmlDocument
    STRING[] currDiagramElements = parseCurrentDiagramElementsIDs()
    XmlNodeList connectorClass = xmlDoc.GetElementsByTagName("connectors")

    FOREACH connector IN connectorClass
        IF connector HAS child nodes
            childNodeList = connector.ChildNodes;
            FOREACH XmlNode childNode IN childNodeList
                xmiConnectorClass = NEW Relation
                xmiConnectorClass.ConnectorXmiId = EXTRACT value FROM childNode WHERE
attributes IS xmi:idref
                IF (childNode has child nodes)
                    XmlNodeList childNodeNextList = childNode.ChildNodes
                    FOREACH nodeNext IN childNodeNextList
                        STRING name = nodeNext.Name
                        SWITCH name DO
                            CASE source
                                SET attributes for source
                            CASE target
                                SET attributes for target
                            CASE model
                                SET attributes for model
                            CASE properties:
                                SET attributes for properties
                            CASE modifiers
                                SET attributes for modifiers
                            CASE appearance

```

```

        SET attributes for appearance
CASE labels
        SET attributes for labels
CASE extendedProperties
        SET attributes for extendedProperties
    END FOR
END IF
IF (SourceModelType EQUALS TO Class IN xmiConnecorClass OR
SourceModelType EQUALS TO Interface IN xmiConnecorClass)
    AND
    (TargetModelType EQUALS TO Class IN xmiConnecorClass OR
TargetModelType EQUALS TO Interface IN xmiConnecorClass)

    IF (currDiagramElements contains xmiConnecorClass.ConnectorXmId)
        ADD xmiConnecorClass TO connecorClassesList
    END IF
    END IF
    END FOR
    END IF
END FOR

RETURN connecorClassesList
END FUNCTION

```

Nasledujúca ukážka demonštruje získavanie údajov o triedach.

```

FUNCTION ParseClasses()
    Class[] XMIClassList = EMPTY_ARRAY
    XmlDocument xmlDoc = NEW XmlDocument
    STRING xml = System.IO.File.ReadAllText(path)
    xmlDoc.LoadXml(xml)
    STRING currDiagramID = System.IO.File.ReadAllText(currDiagramIDPath)
    STRING[] currDiagramElements = parseCurrentDiagramElementsIDs();

    XmlNodeList classNodeList = xmlDoc.GetElementsByTagName("UML:Class")
    XmlNodeList classIndices = xmlDoc.GetElementsByTagName("UML:DiagramElement")
    XmlNodeList elementClass = xmlDoc.GetElementsByTagName("elements")

    XmlNodeList elementsClass = elementClass[0].ChildNodes
    XmlNodeList geometryElements = NULL

```

```

FOR i = 1 TO number of elements in elementsClass - 1
    XmlNode parentDiagram = elementClass[i].ParentNode
    XmlNodeList parentDiagramNodes = parentDiagram.ChildNodes
    FOREACH XmlNode node IN parentDiagramNodes
        IF node name EQUAL to model
            IF value of extracted attributes IS localID IN node EQUAL TO currDiagramID
                geometryElements = elementClass[i].ChildNodes
                BREAK
            END IF
        END IF
    END FOR
END FOR

FOR i = 1 TO number of elements in elementsClass - 1
    Class XMIClass = NEW Class
    XMIClass.Name = EXTRACT value FROM elementsClass WHERE attributes IS name
    XMIClass.Name = EXTRACT value FROM elementsClass WHERE attributes IS xmi:type
    XMIClass.Name = EXTRACT value FROM elementsClass WHERE attributes IS xmi:idref

    IF      type of XMIClass IS NOT uml:Interface
        OR      type of XMIClass IS NOT uml:Class
        CONTINUE;
    END IF

    IF elementsClass[i] has child nodes
        XmlNodeList test = elementsClass[i].ChildNodes
        FOREACH node in test
            IF node name EQUALS attributes
                XmlNodeList attributes = node.ChildNodes
                XMIClass.Attributes = NEW Attribute
                FOREACH attribute IN attributes
                    ADD attribute TO XMIClass attributes
                END FOR
            ELSE IF node name EQUALS operations
                XmlNodeList operations = node.ChildNodes
                XMIClass.Methods = NEW Method
                FOREACH (XmlNode operation in operations)
                    ADD method TO XMIClass methodes
                END FOR
            END IF
        END FOR
    END IF

    IF currDiagramElements CONTAINS XMIClass.XmiId
        XMIClassList.Add(XMIClass)
    END IF

```

END FOR

```
FOR i = 1 TO number of elements in geometryElements - 1
    STRING subject = geometryElements[i].Attributes["subject"].Value
    FOREACH item XMIClassList
        IF item.XmiId EQUALS subject
            SET geometry elements
        END IF
    END FOR
END FOR

RETURN XMIClassList;
```

END FUNCTION

Prílohy

Príloha A - Testovacie scenáre

1. Požiadavky na používanie Unity Addinu pre animovanie architektúr

- a. Windows 10
- b. Mať nainštalovaný **Enterprise Architect** (stačí aj trial verzia)
- c. Mať **Disk C**, kam bude addin nainštalovaný.
- d. 200 MB voľného miesta na disku

2. Inštalácia

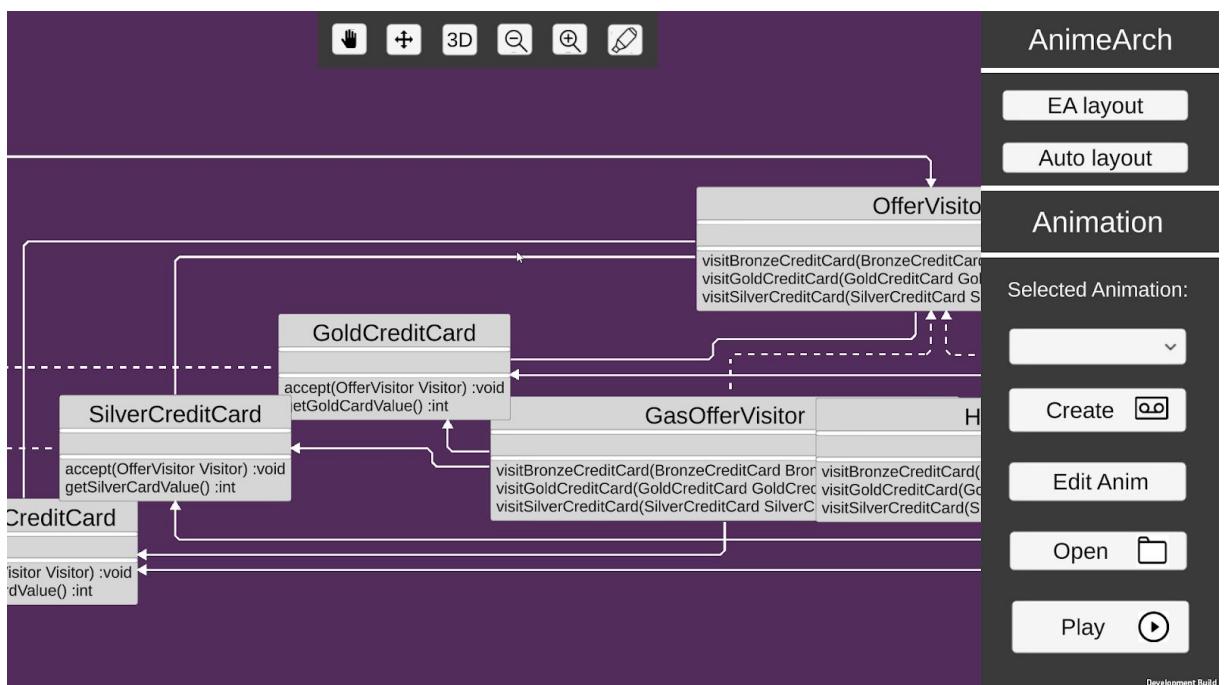
- a. Spustíme AnimArch.exe ako správca. Aplikácia rozbalí potrebné súbory, pridá cestu "C:\AnimArch\AnimeArch" do USER_PATH premennej a add-in do registrov OS.

 AnimArch	4/30/2020 10:20 PM	Application	24,168 KB
---	--------------------	-------------	-----------

- b. (Poznámka: odporúčaná metóda je ísť cez Enterprise Architect + plugin (vid' 3. Spustenie pluginu v Enterprise Architect), zatiaľ nemáme stand-alone verziu. Táto aplikácia slúži pre prípad zlyhania EA a preto má svoje obmedzenia, ako napríklad otvára len predvolený pribalený diagram.)
Nájdeme adresár C:\AnimArch\AnimeArch a spustíme AnimeArch.exe.

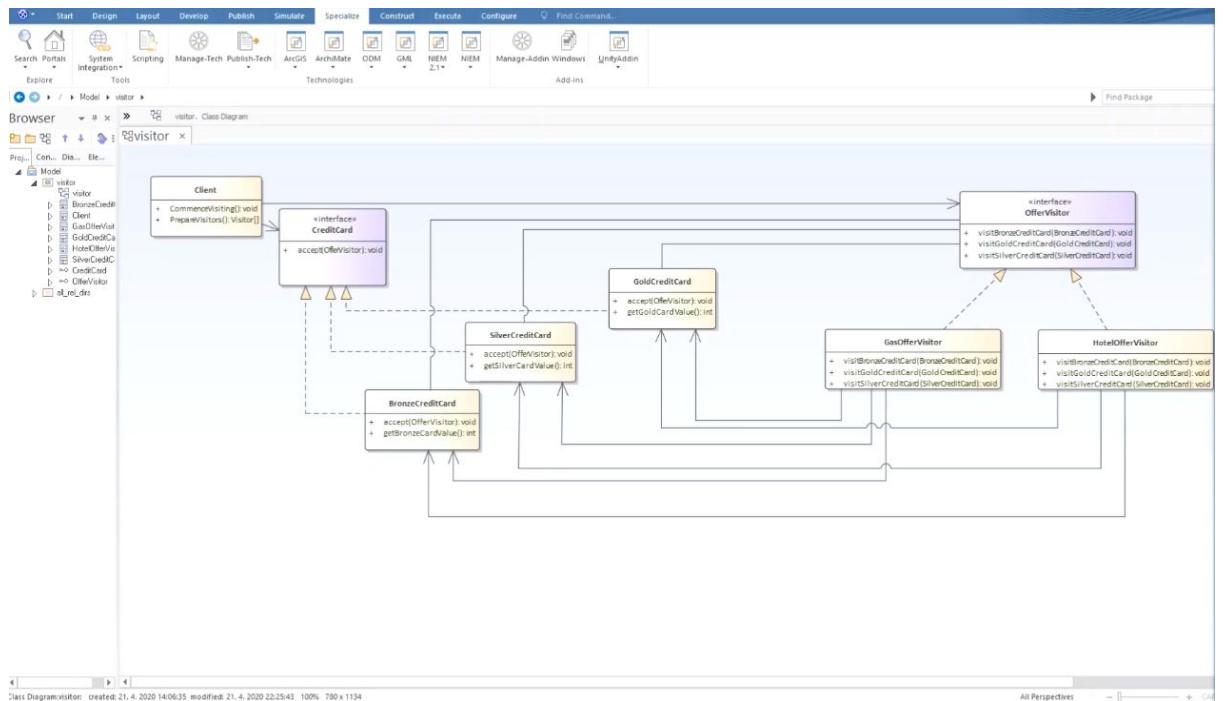
 AnimeArch_Data	4/30/2020 10:16 PM	File folder	
 MonoBleedingEdge	4/30/2020 8:16 PM	File folder	
 AnimeArch	3/5/2020 3:47 PM	Application	636 KB
 UnityCrashHandler64	3/5/2020 3:48 PM	Application	1,069 KB
 UnityPlayer.dll	3/5/2020 3:48 PM	Application exten...	25,163 KB

c. Ak všetko prebehlo úspešne zobrazí sa táto obrazovka:

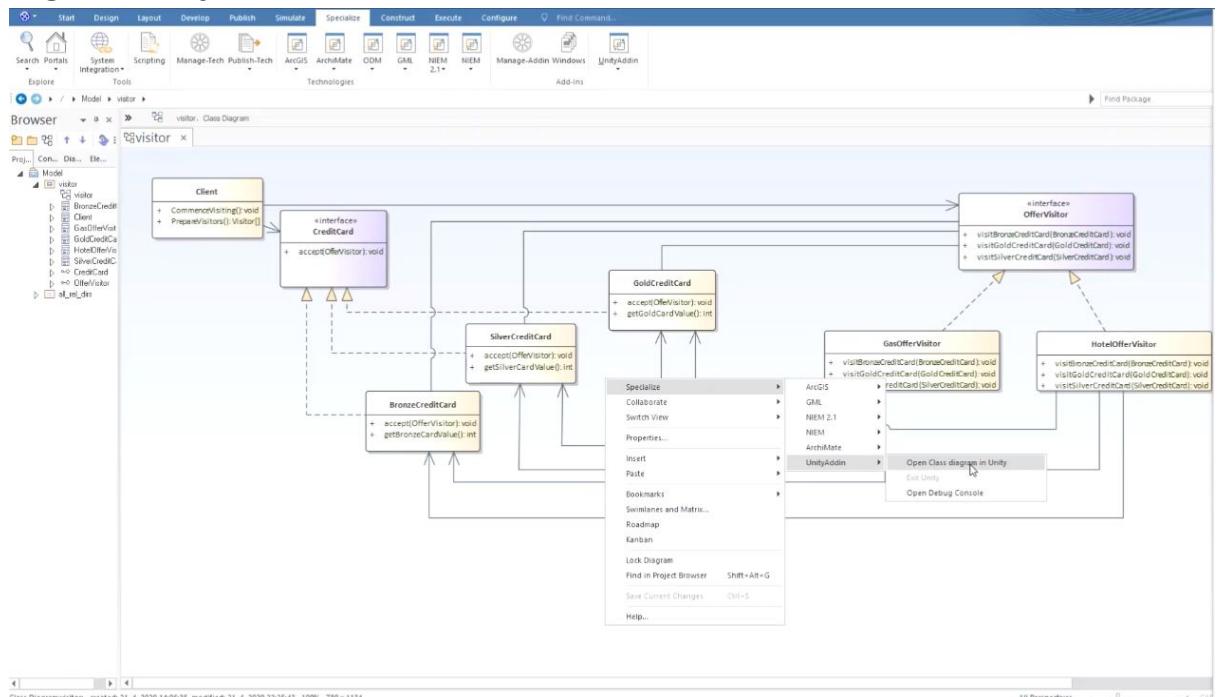


3. Spustenie pluginu v Enterprise Architect

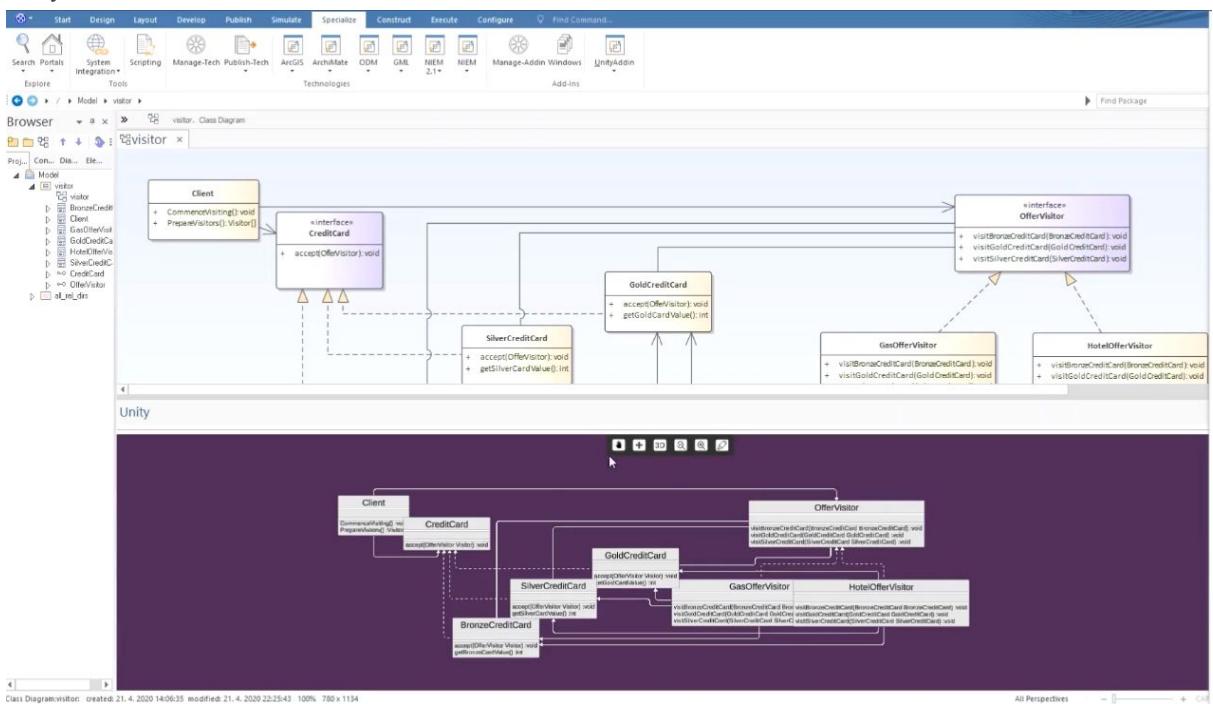
- a. Spustíme EA a vytvoríme/importujeme diagram tried



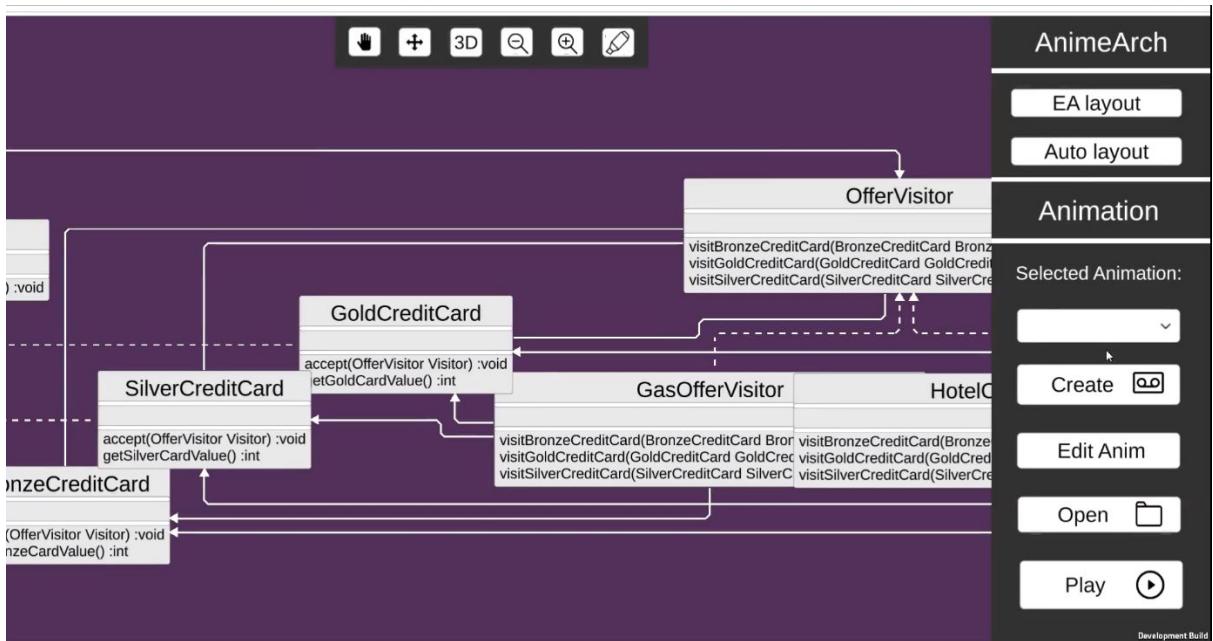
- b. Klikneme pravým tlačidlom na diagram, vyberieme možnosť **Specialize**, v ďalšom paneli vyberieme možnosť **UnityAddin** a v ďalšom paneli klikneme na možnosť **Open class diagram in Unity**.



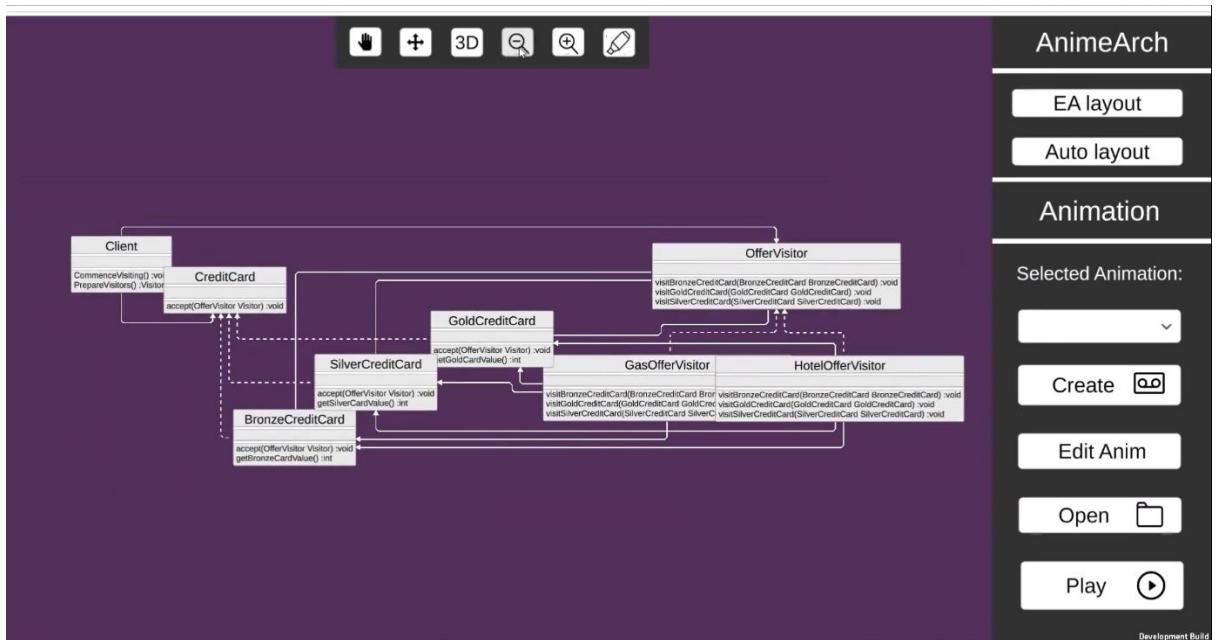
- c. Na spodnej časti obrazovky sa otvorí Unity okno a po načítaní sa zobrazí diagram tried v Unity.



- d. Kliknutím na tlačidlo Maximalizovať v pravom hornom rohu Unity okna si zobrazíme Unity diagram na celú obrazovku.

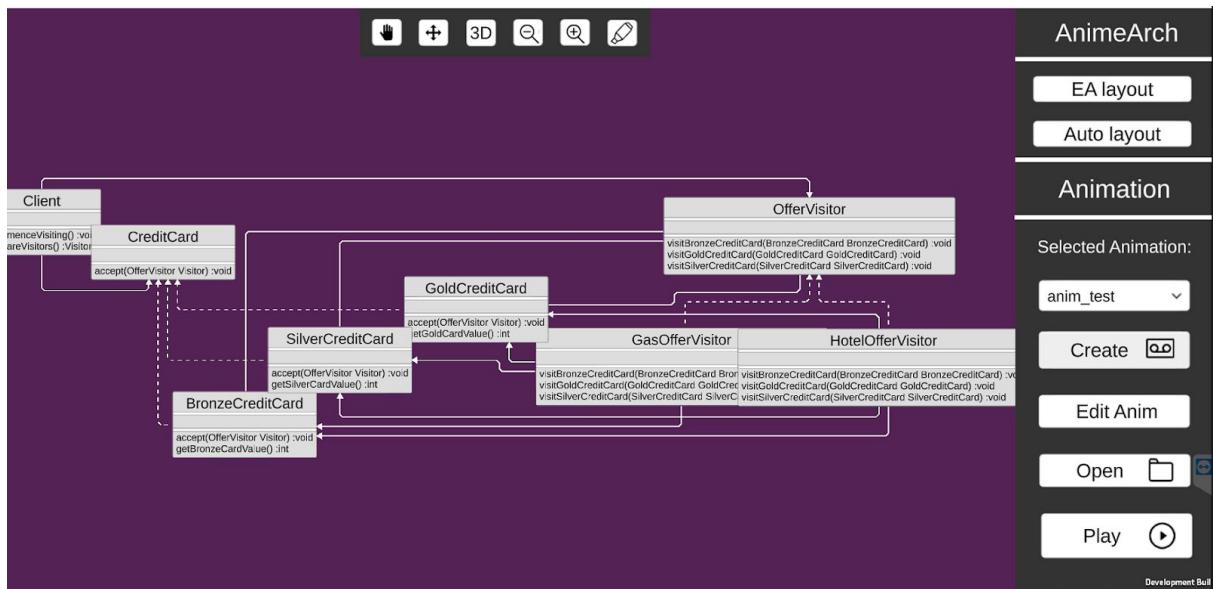


- e. Tlačidlami v na hornej lište si prispôsobíme zobrazenie diagramu podľa potreby (priblížiť, oddialiť)

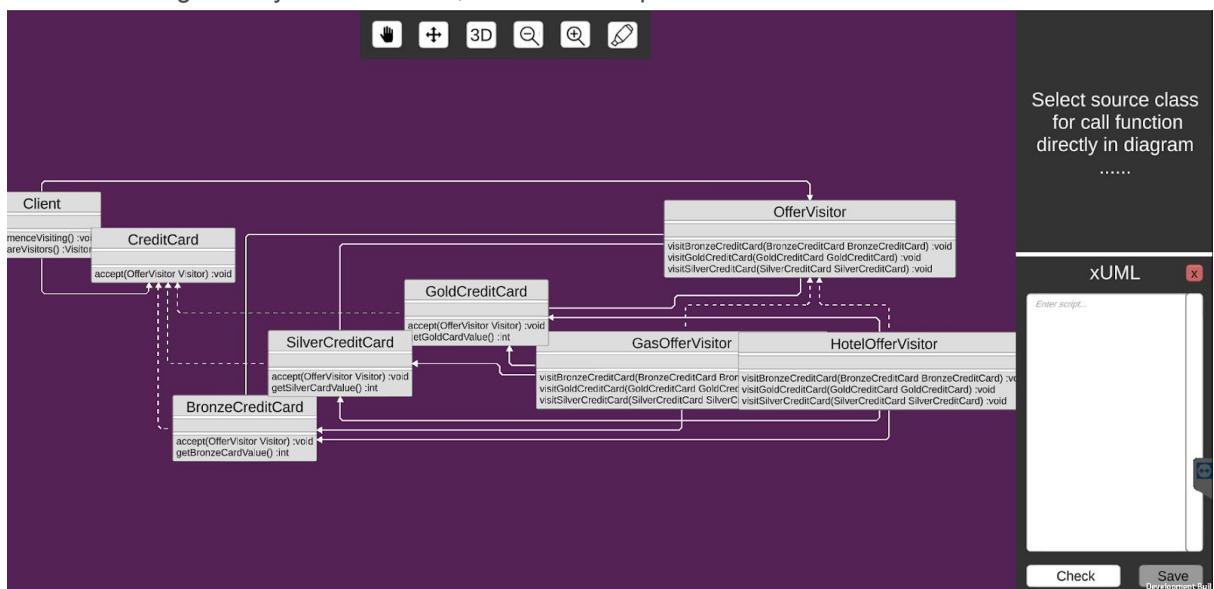


4. Vytvorenie jednoduchej animácie

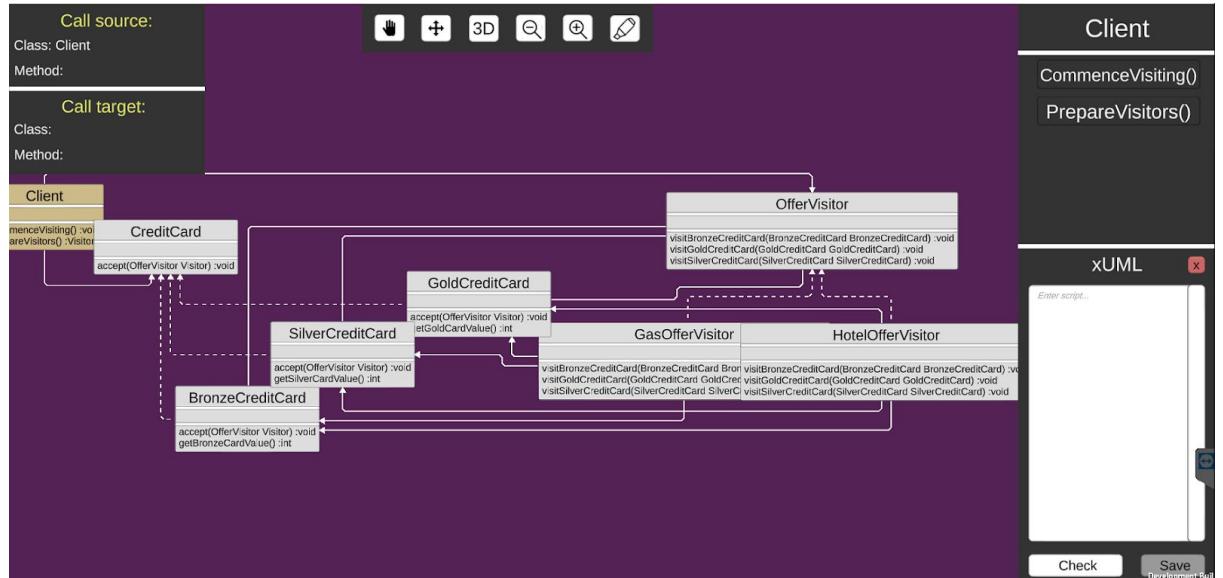
- a. Klikneme na tlačidlo Create.



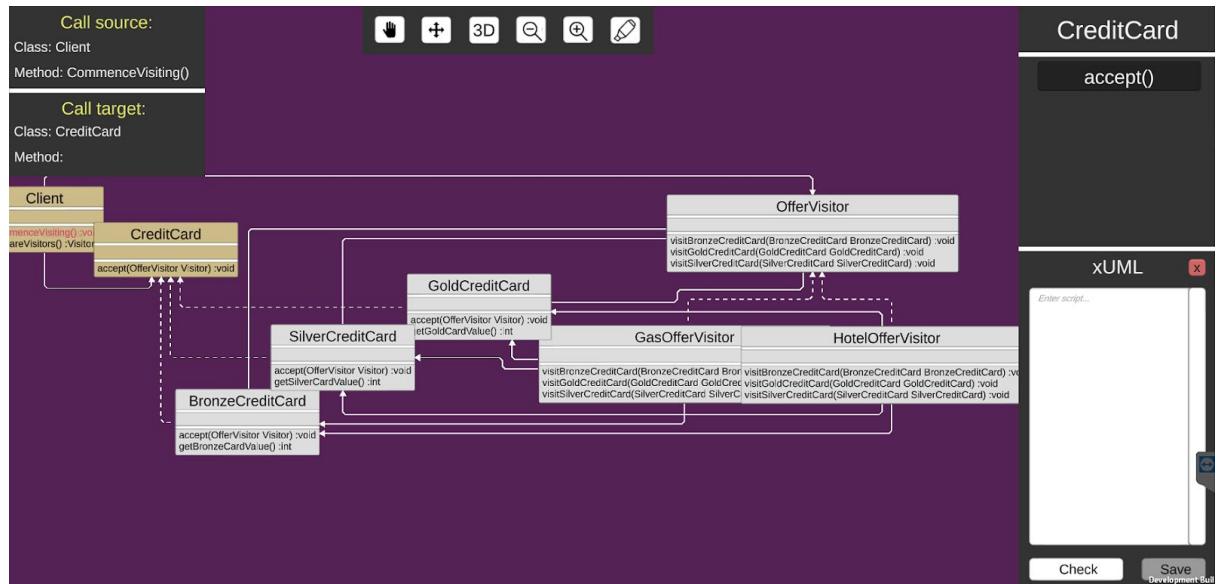
- b. Kliknutím v diagrame vyberieme triedu, ktorú chceme pridať do animácie.



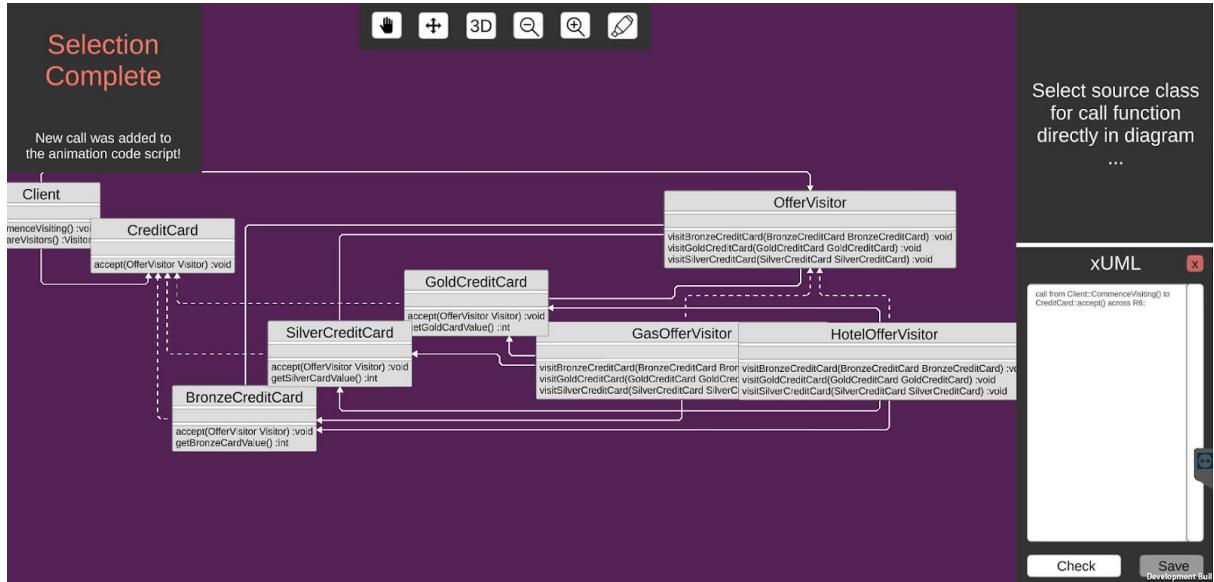
c. Vyberieme požadovanú metódu na paneli vpravo hore.



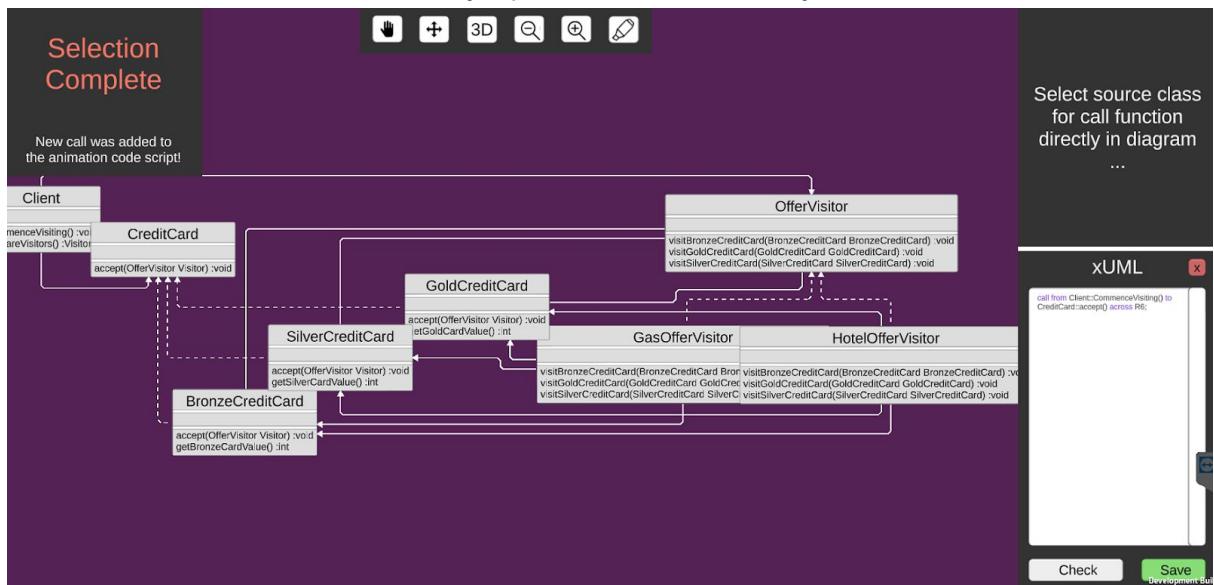
d. Kliknutím vyberieme druhú triedu a následne jej metódu.



- e. Volanie bolo pridané do animácie. Do xUML bol pridaný kód animácie.

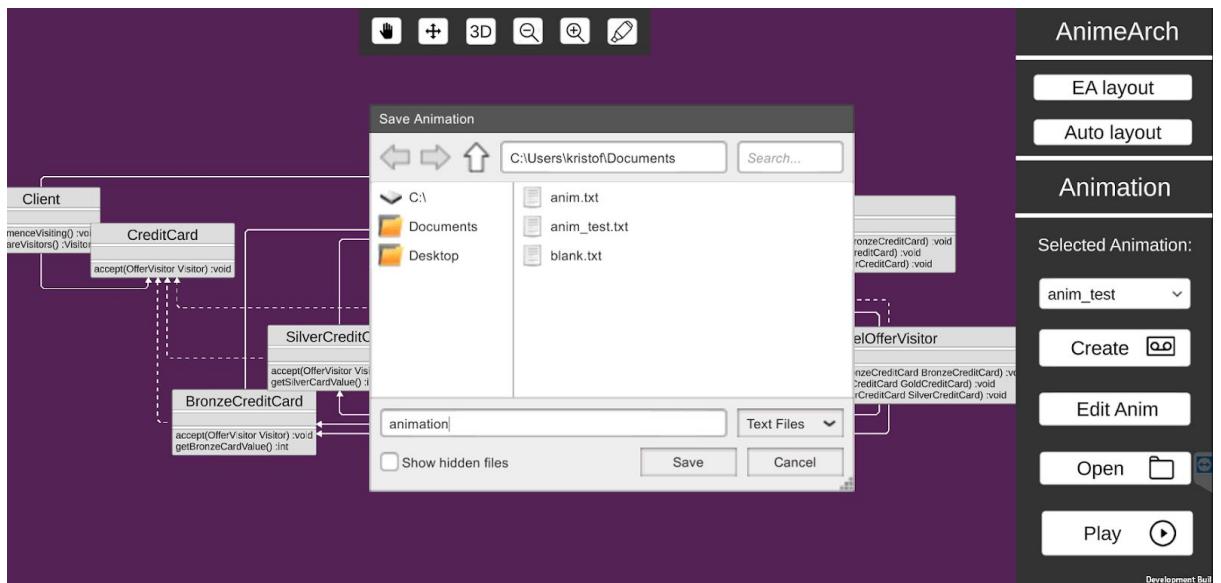


- f. Kliknutím na tlačidlo check sa skontroluje správnosť kódu a odblokuje sa zlačidlo Save.

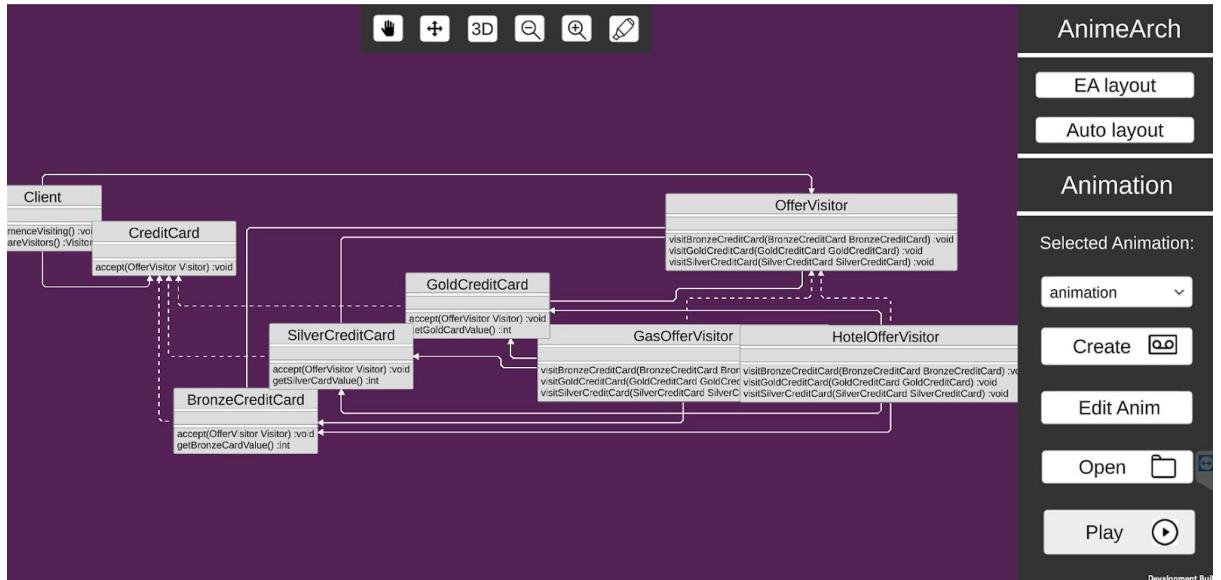


- g. Kód animácie je potrebné uložiť kliknutím na tlačidlo Save. Po kliknutí na tlačidlo Save sa zobrazí dialógové okno v ktorom vyberieme miesto na disku, kam chceme uložiť animáciu. Tá

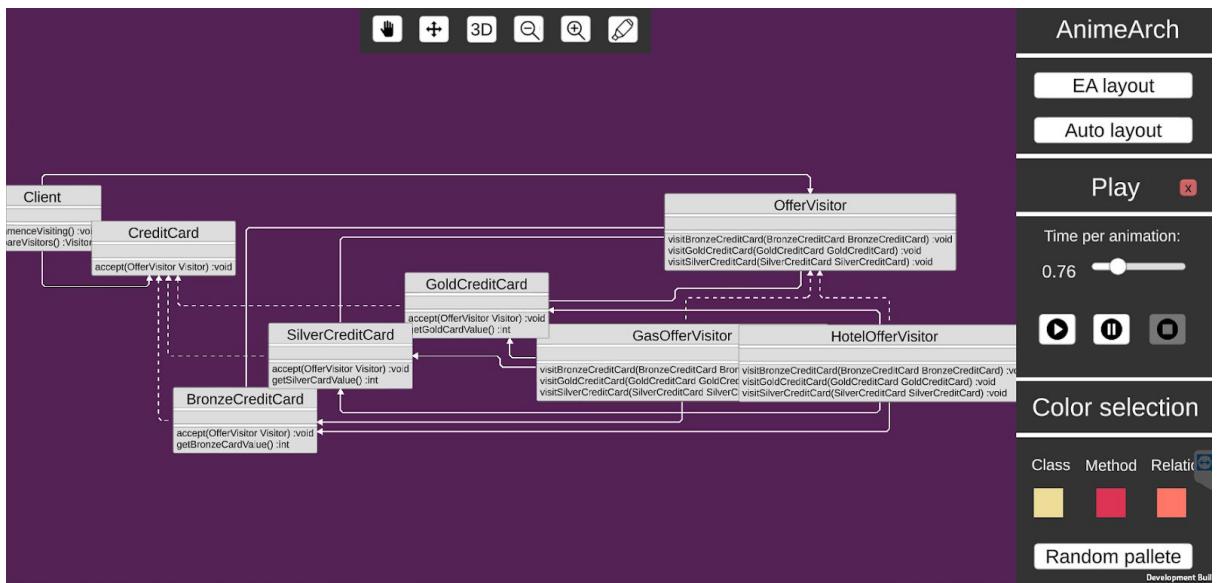
sa uloží vo formáte .txt.



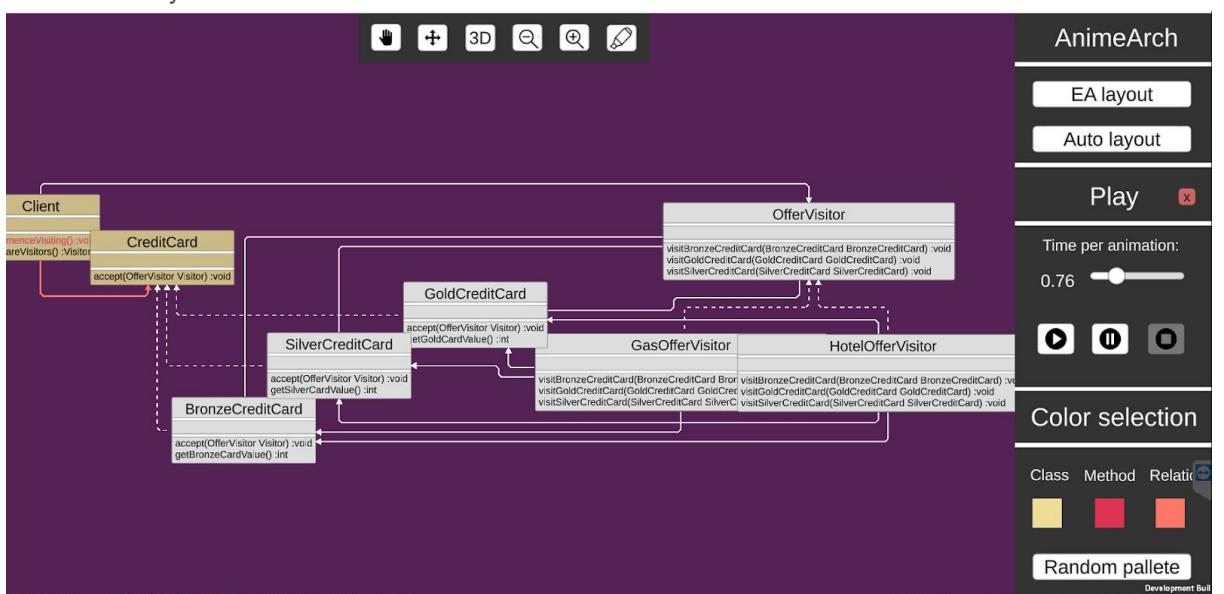
h. Po uložení animácie ju spustíme kliknutím na tlačidlo Play.



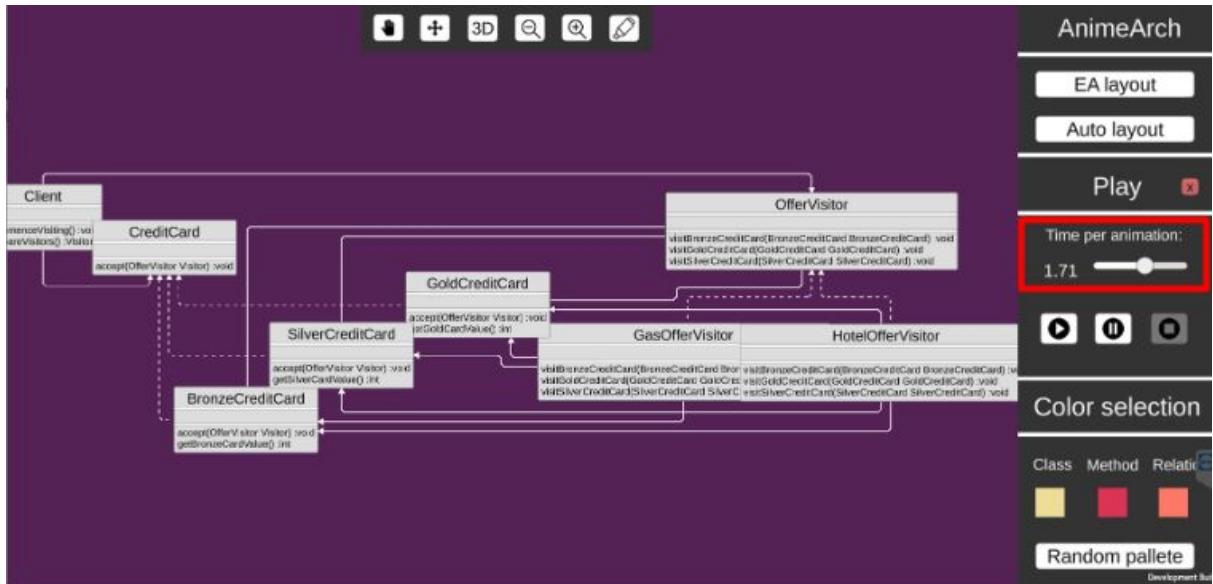
- i. Na pravom paneli sa zobrazí menu prehrávania animácie. Kliknutím na tlačidlo s ikonou Play sa spustí prehrávanie animácie.



- j. Animácia sa vykonáva.

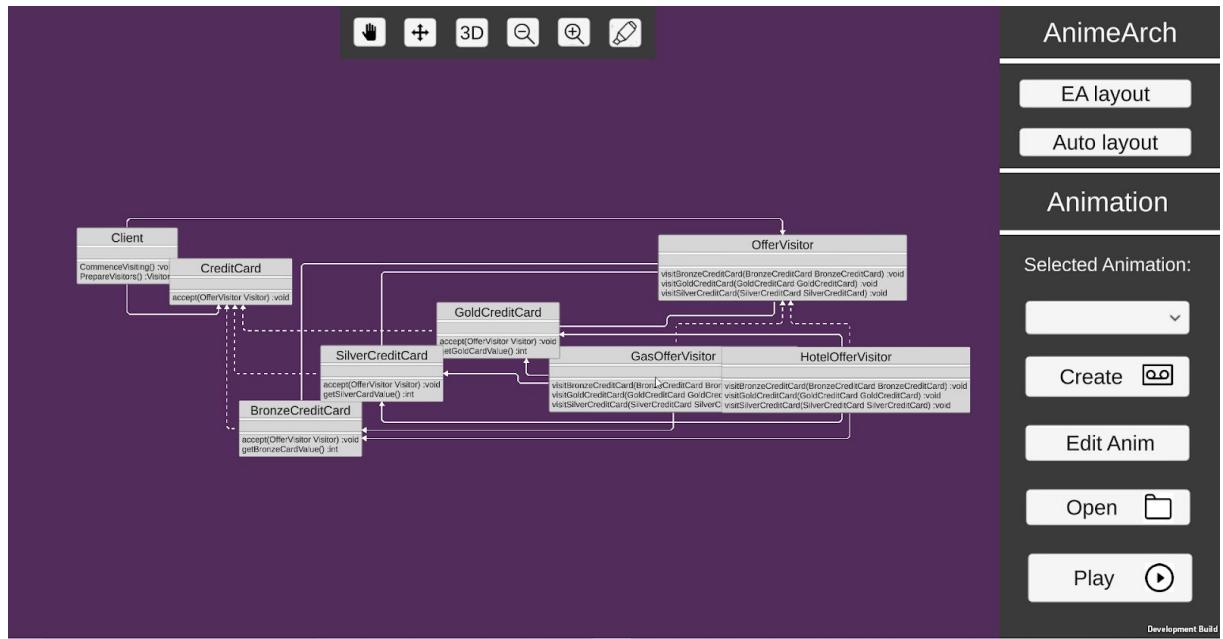


k. Na paneli vpravo môžeme nastaviť rýchlosť vykonávania animácie posuvníkom.

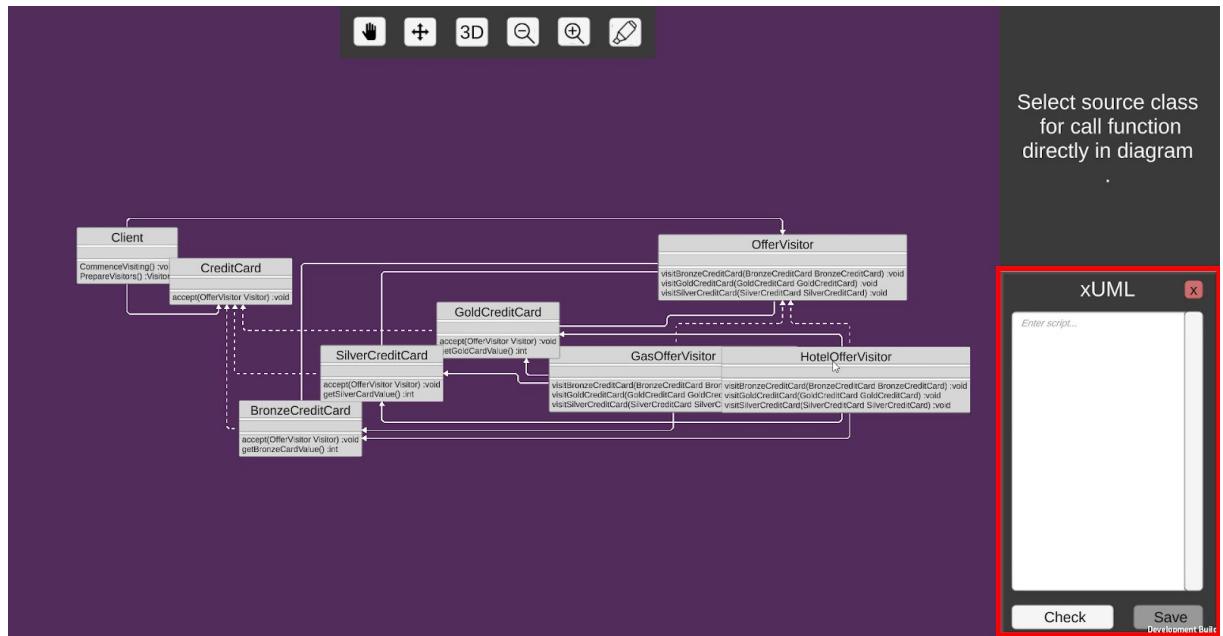


5. Testovanie podmienku “IF”

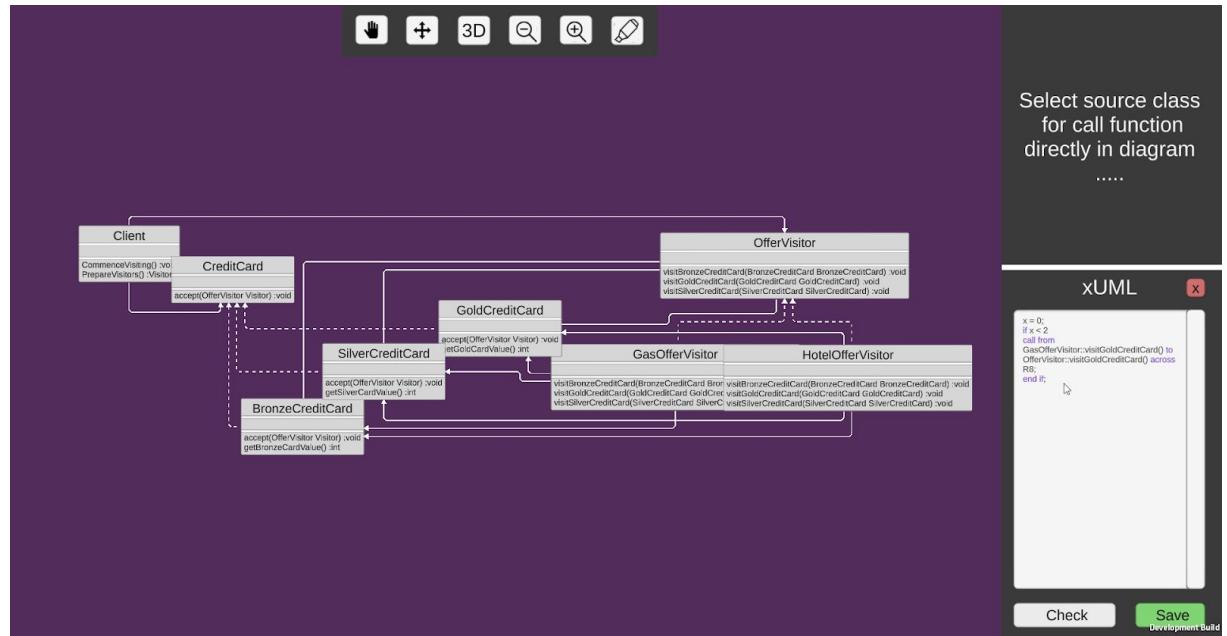
- Klikneme na tlačidlo Create.



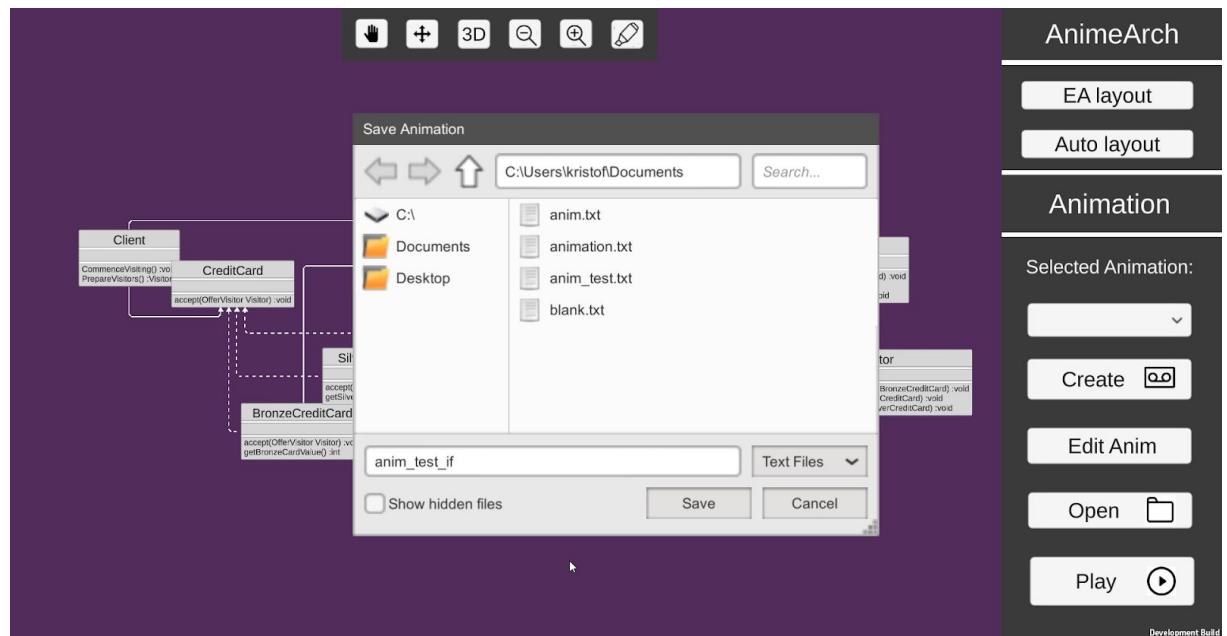
- Do xUML panelu napíšeme kód s podmienkou “IF”



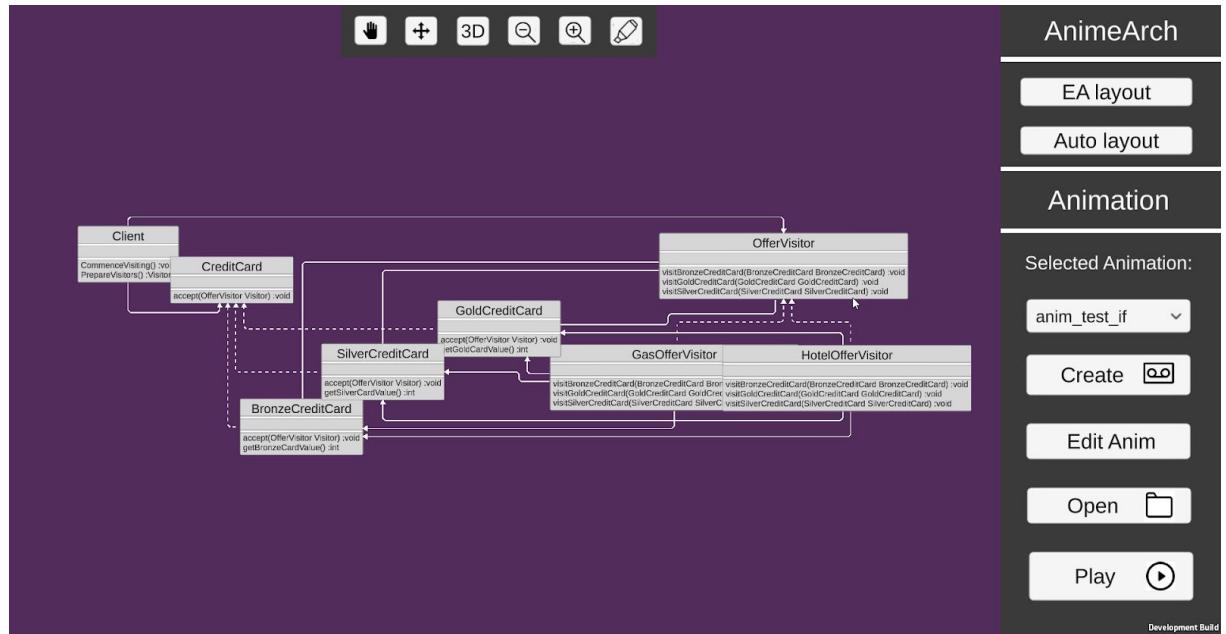
- c. Kliknutím na tlačidlo check sa skontroluje správnosť kódu a odblokuje sa tlačidlo Save.



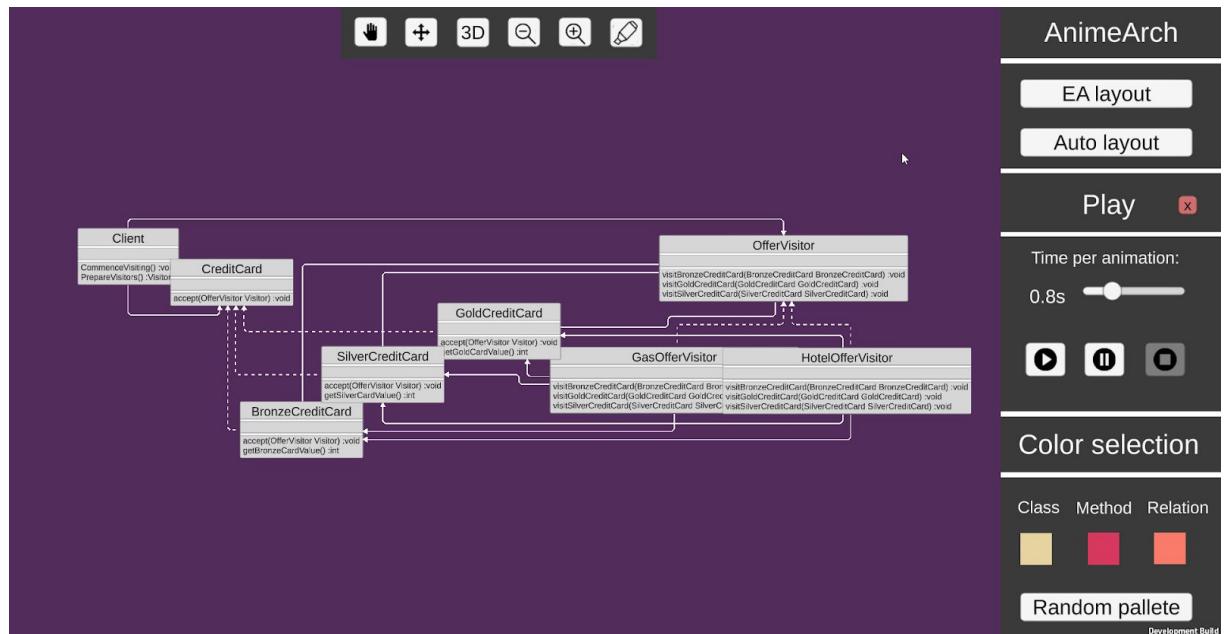
- d. Kód animácie je potrebné uložiť kliknutím na tlačidlo Save. Po kliknutí na tlačidlo Save sa zobrazí dialógové okno v ktorom vyberieme miesto na disku, kam chceme uložiť animáciu. Tá sa uloží vo formáte .txt.



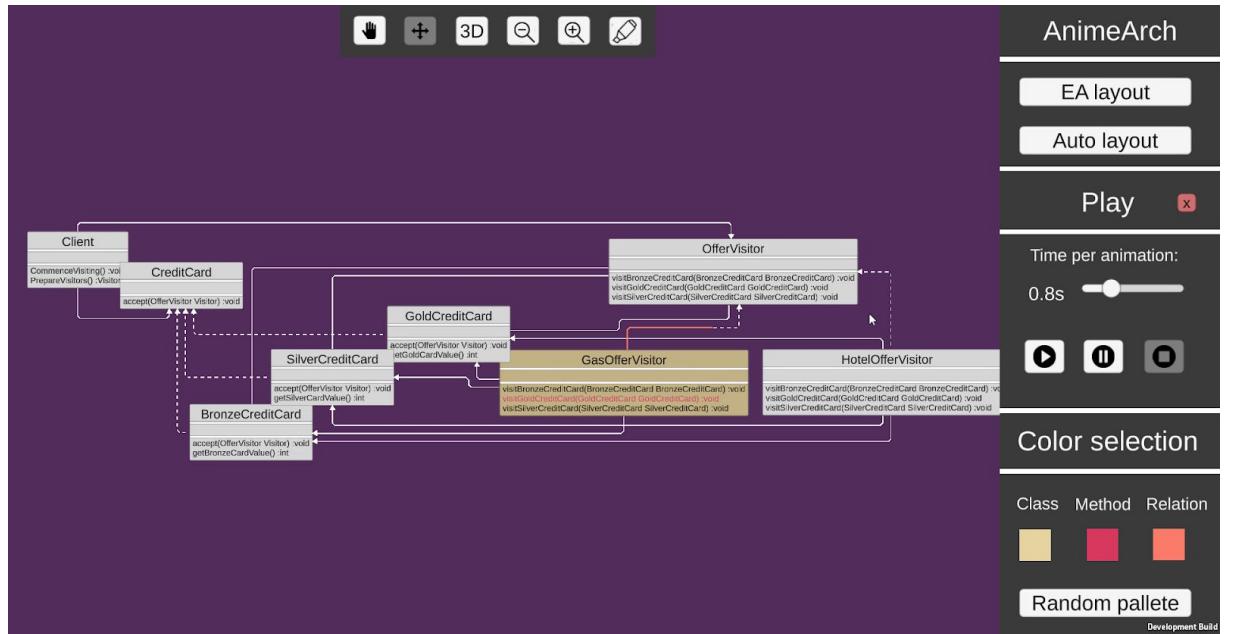
e. Po uložení animácie ju spustíme kliknutím na tlačidlo Play.



f. Na pravom paneli sa zobrazí menu prehrávania animácie. Kliknutím na tlačidlo s ikonou Play sa spustí prehrávanie animácie.



g. Animácia sa vykonáva s podmienkou "IF", ak je splnená podmienka.



Použitý kód počas testovania:

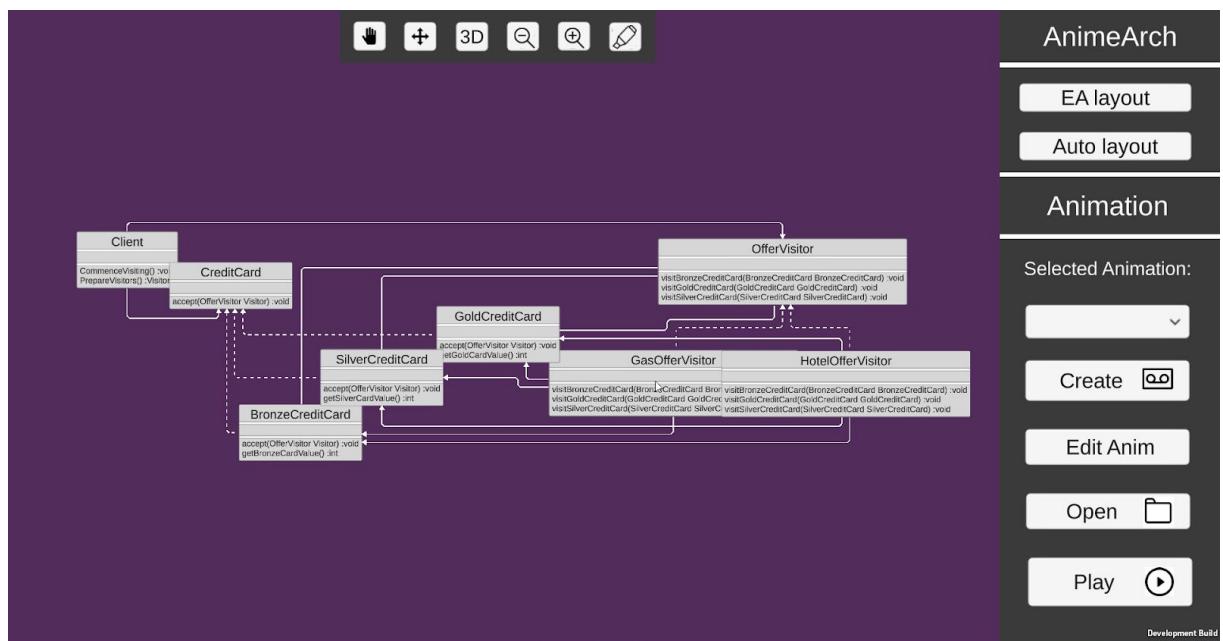
```

x = 0;
if x < 2
call from GasOfferVisitor::visitGoldCreditCard() to OfferVisitor::visitGoldCreditCard() across R8;
end if;

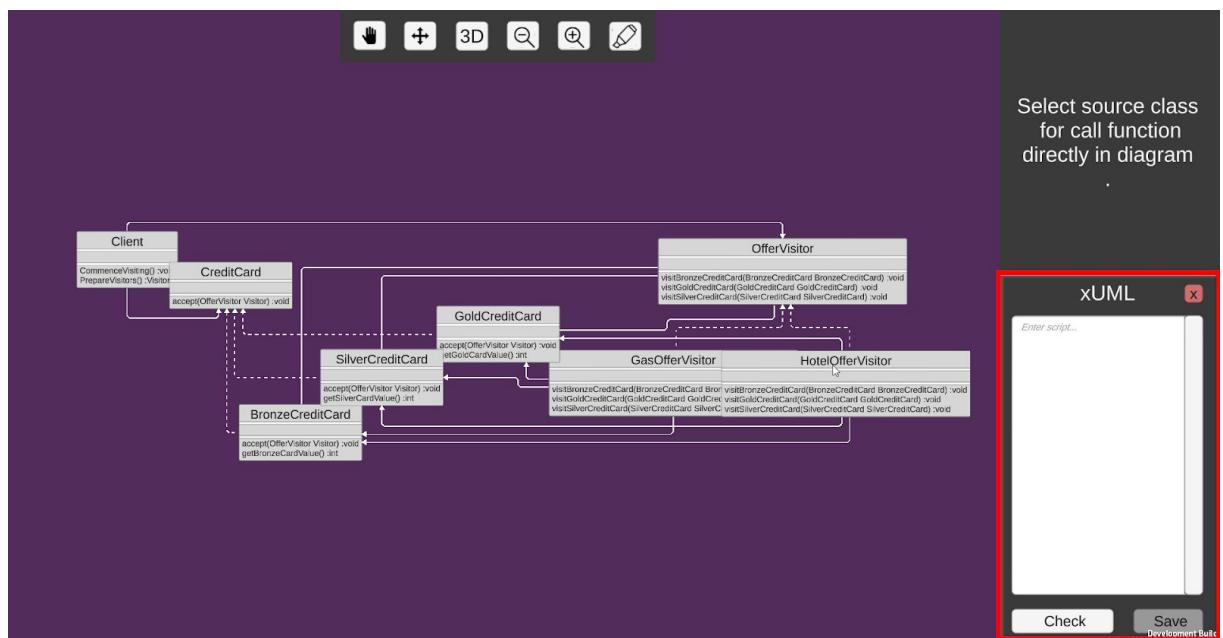
```

6. Testovanie cyklus “WHILE”

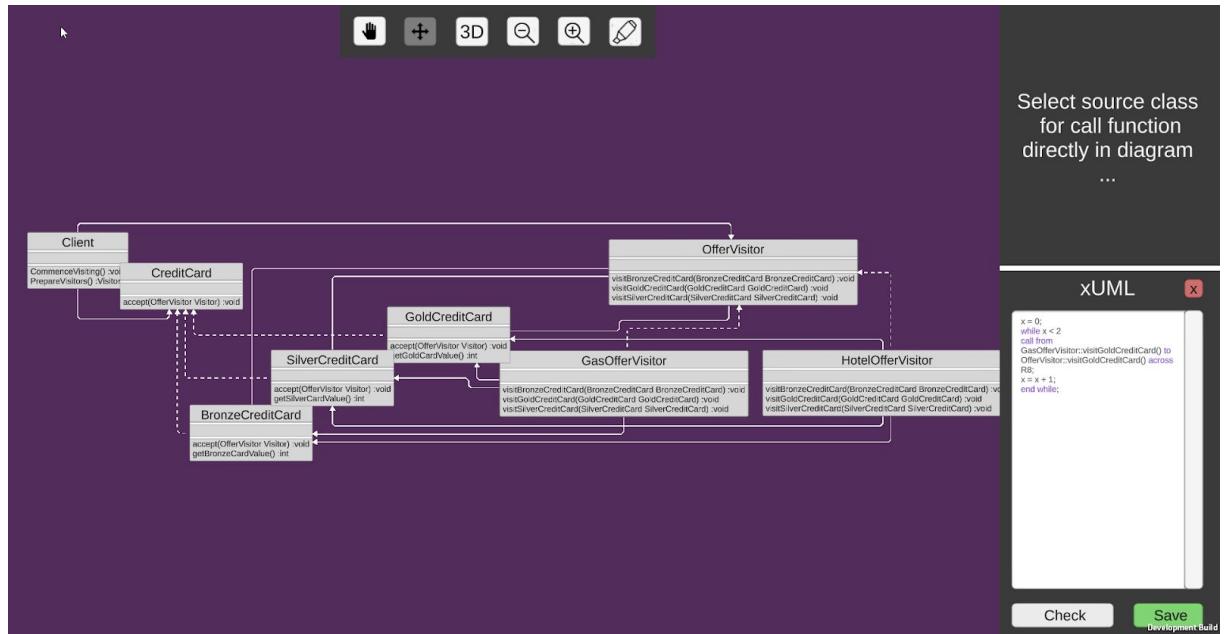
- a. Klikneme na tlačidlo Create.



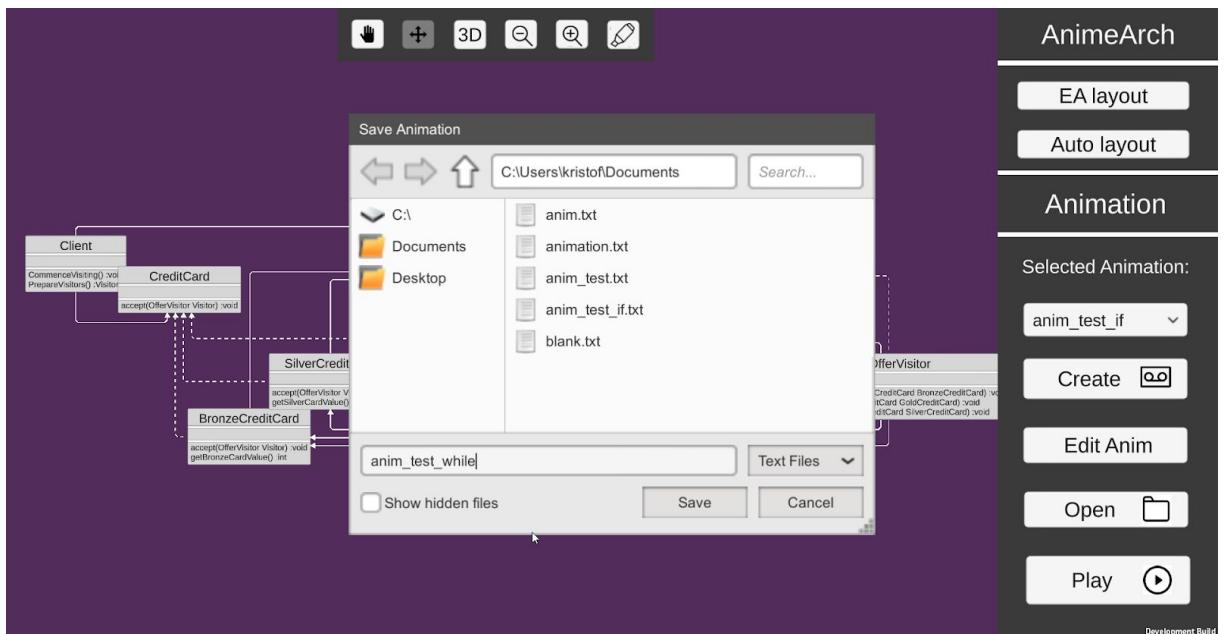
- b. Do xUML panelu napíšeme kód, ktorý obsahuje cyklus “WHILE”



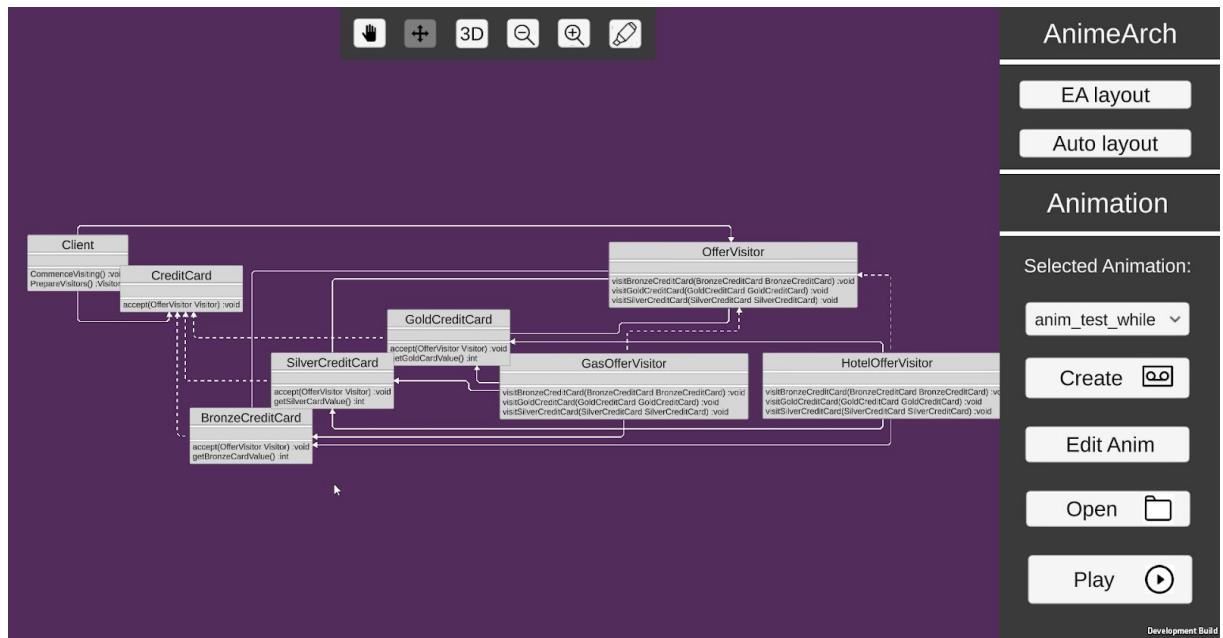
- c. Kliknutím na tlačidlo check sa skontroluje správnosť kódu a odblokuje sa tlačidlo Save.



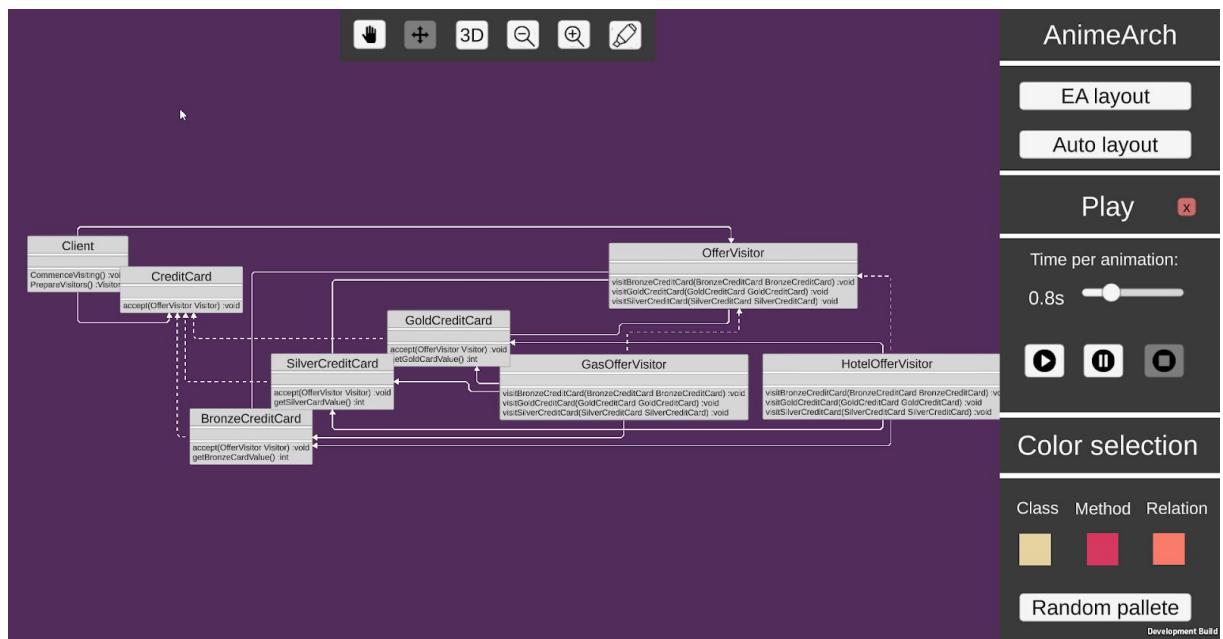
- d. Kód animácie je potrebné uložiť kliknutím na tlačidlo Save. Po kliknutí na tlačidlo Save sa zobrazí dialógové okno v ktorom vyberieme miesto na disku, kam chceme uložiť animáciu. Tá sa uloží vo formáte .txt.



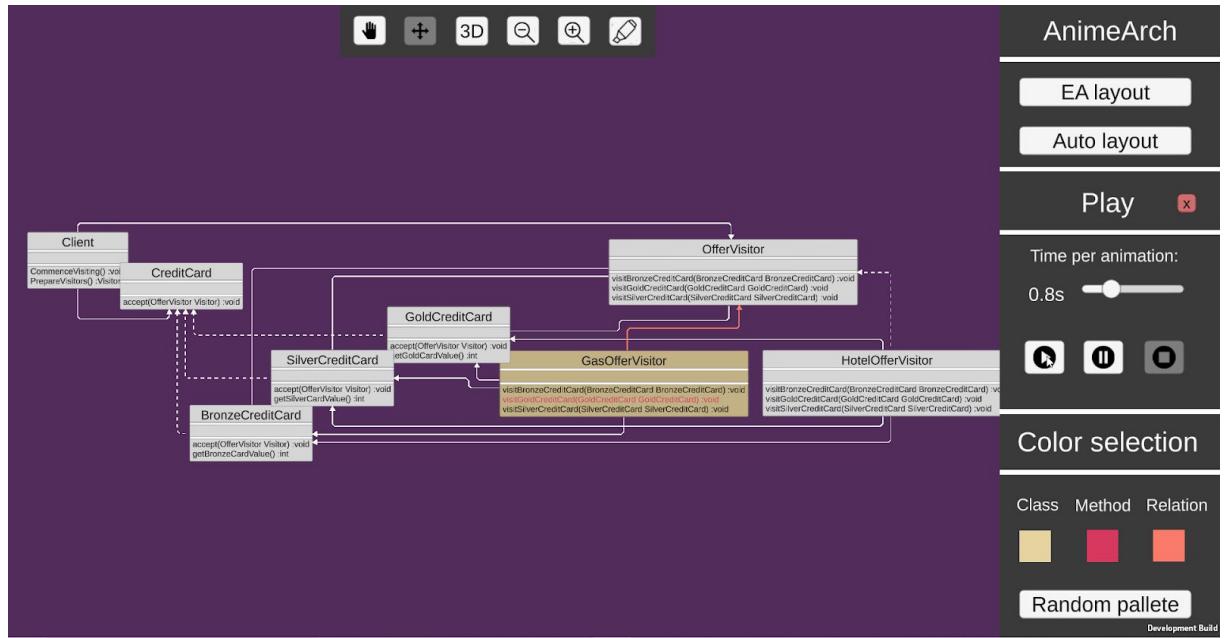
e. Po uložení animácie ju spustíme kliknutím na tlačidlo Play.



f. Na pravom paneli sa zobrazí menu prehrávania animácie. Kliknutím na tlačidlo s ikonou Play sa spustí prehrávanie animácie.



g. Animácia sa vykonáva s cyklom "WHILE" a opakuje sa dovtedy, kým je splnená podmienka.



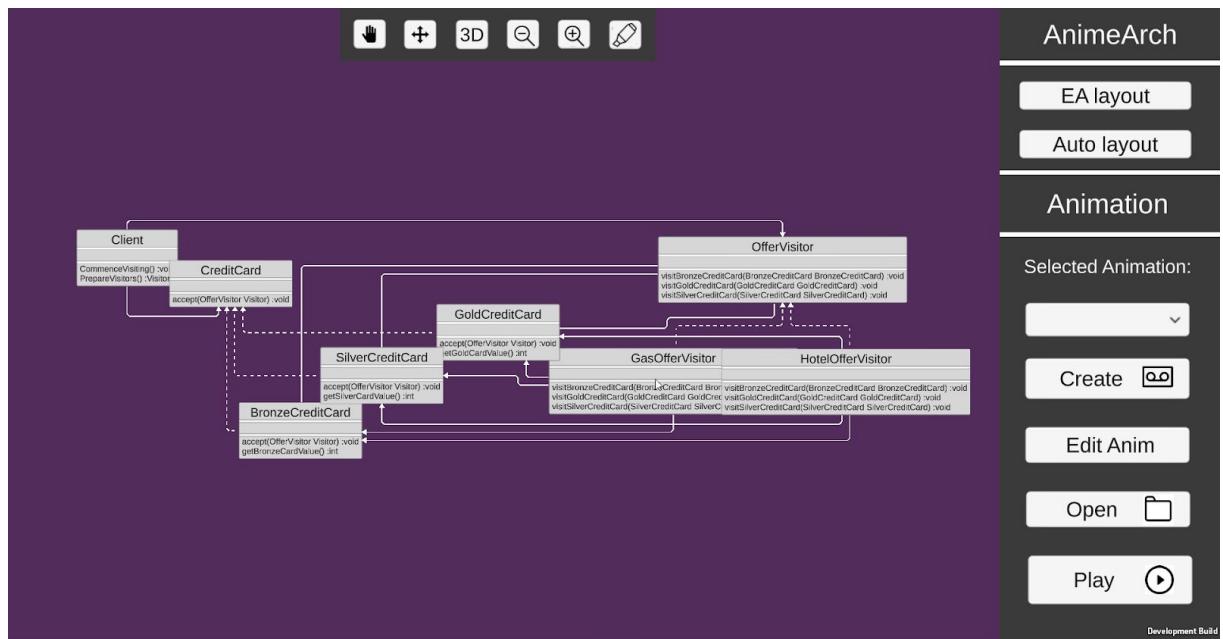
Použitý kód počas testovania:

```

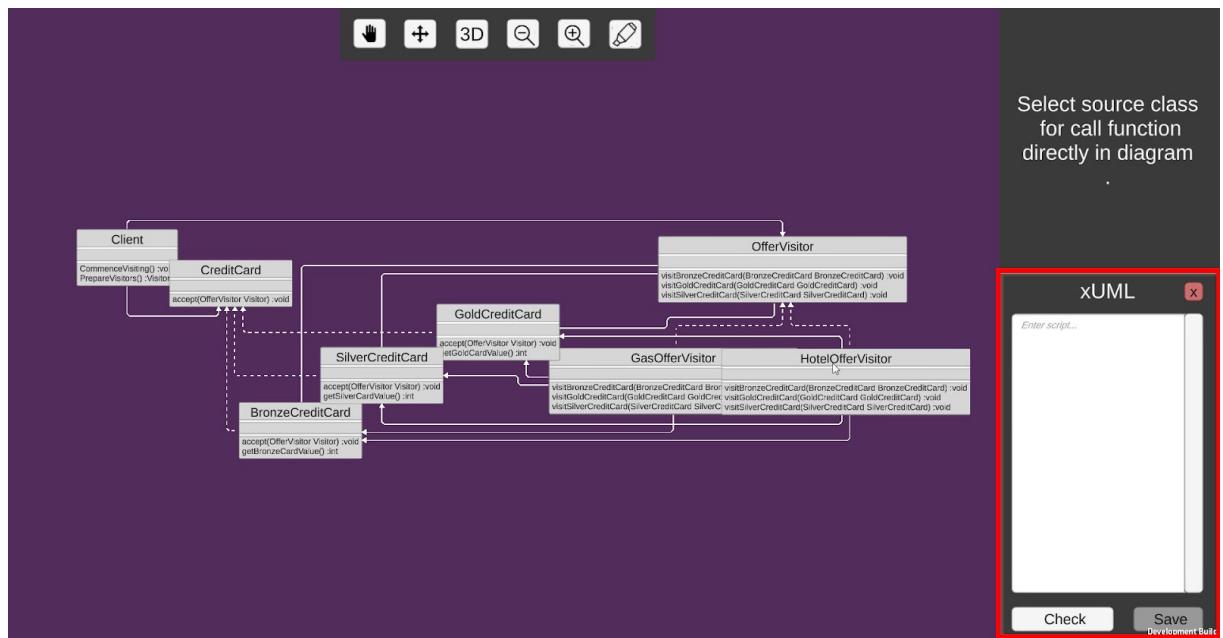
x = 0;
while x < 2
call from GasOfferVisitor::visitGoldCreditCard() to OfferVisitor::visitGoldCreditCard() across R8;
x = x + 1;
end while;
  
```

7. Testovanie vnorený “WHILE” a “IF”

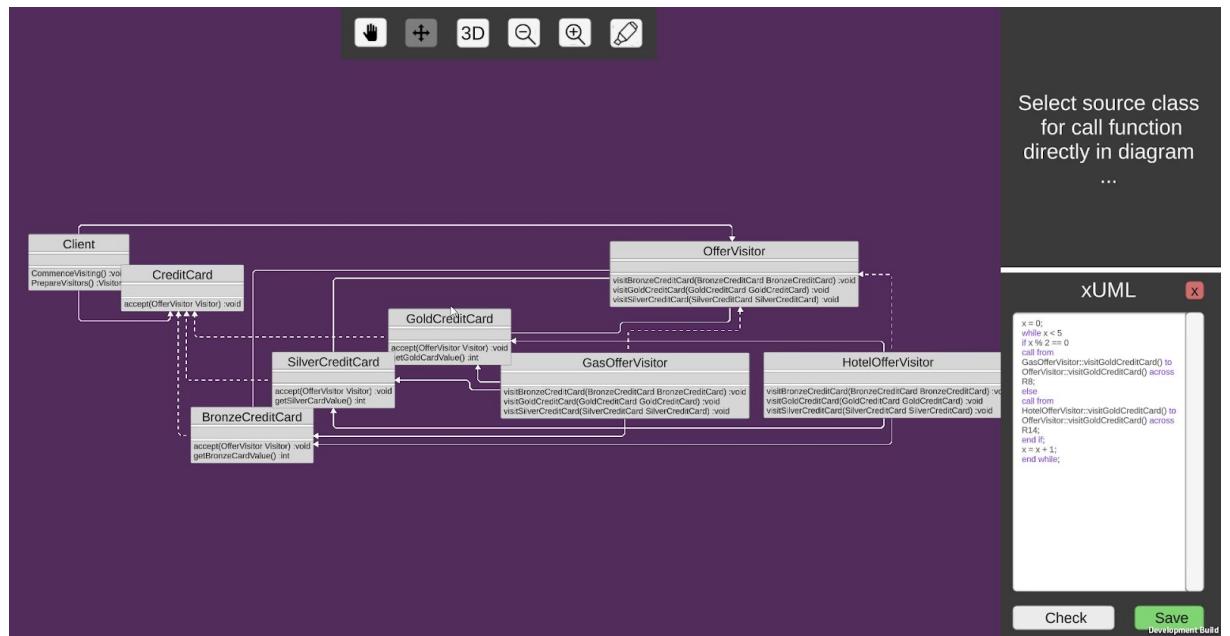
- a. Klikneme na tlačidlo Create.



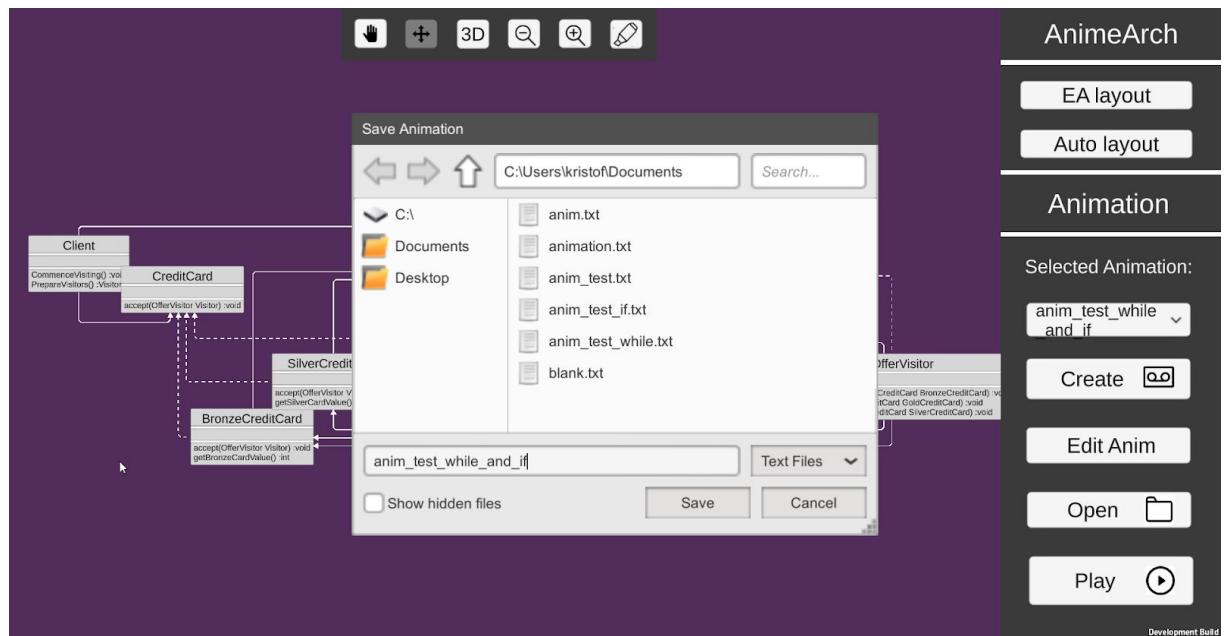
- b. Do xUML panelu napíšeme kód ktorý obsahuje vnorený “WHILE” a “IF”



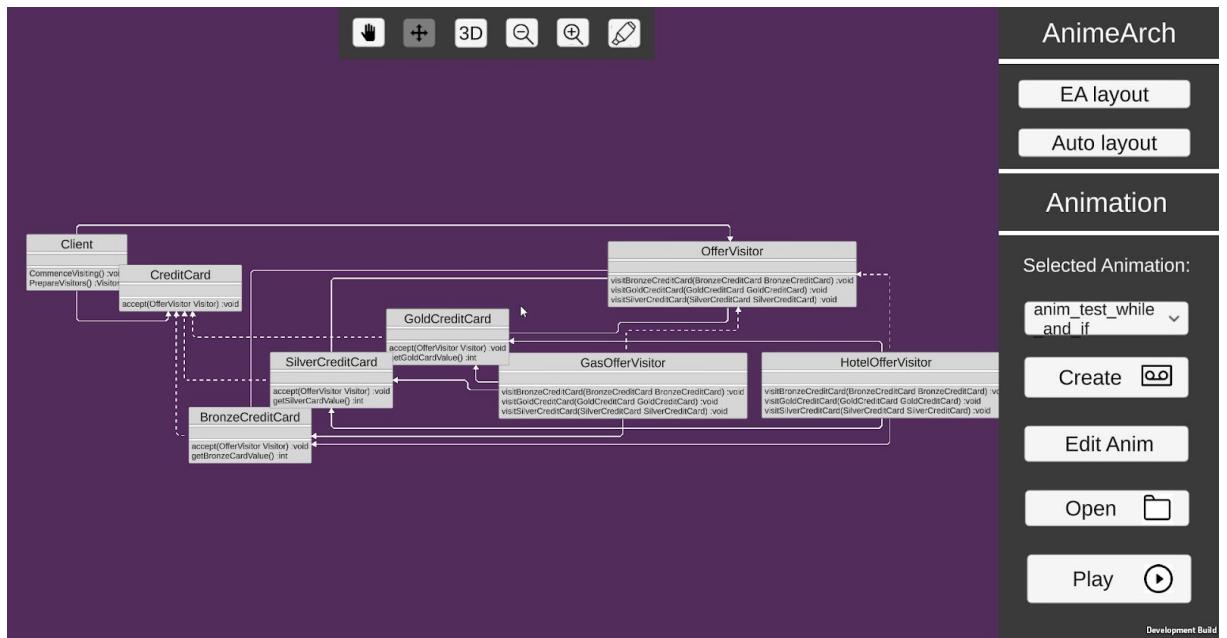
- c. Kliknutím na tlačidlo check sa skontroluje správnosť kódu a odblokuje sa tlačidlo Save.



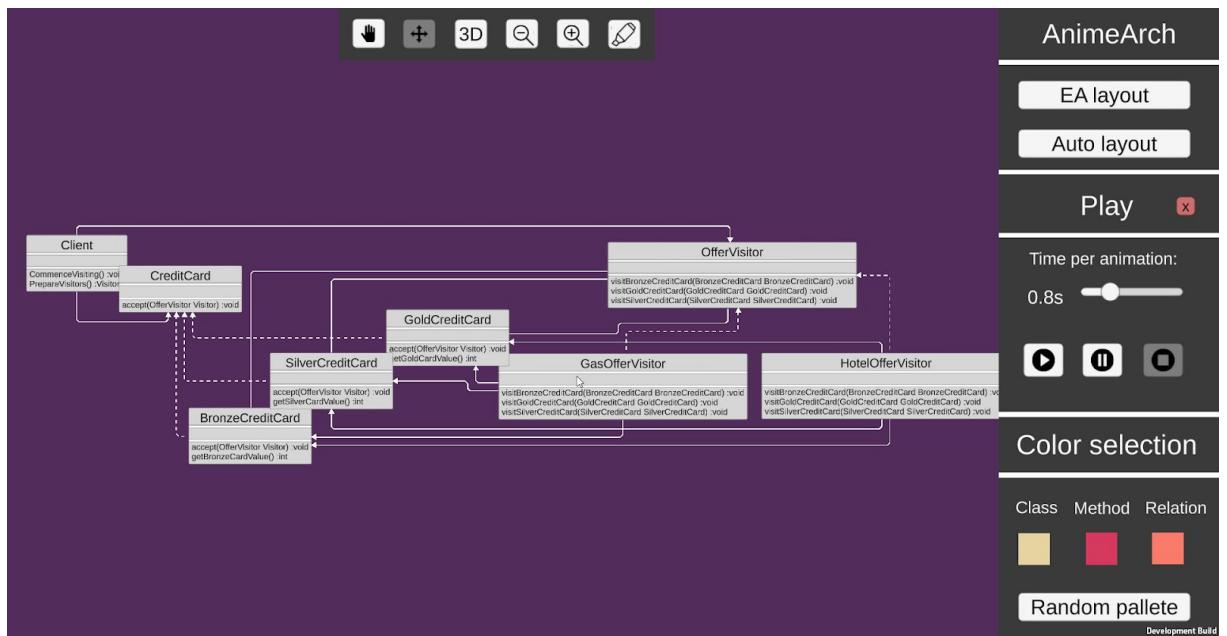
- d. Kód animácie je potrebné uložiť kliknutím na tlačidlo Save. Po kliknutí na tlačidlo Save sa zobrazí dialógové okno v ktorom vyberieme miesto na disku, kam chceme uložiť animáciu. Tá sa uloží vo formáte .txt.



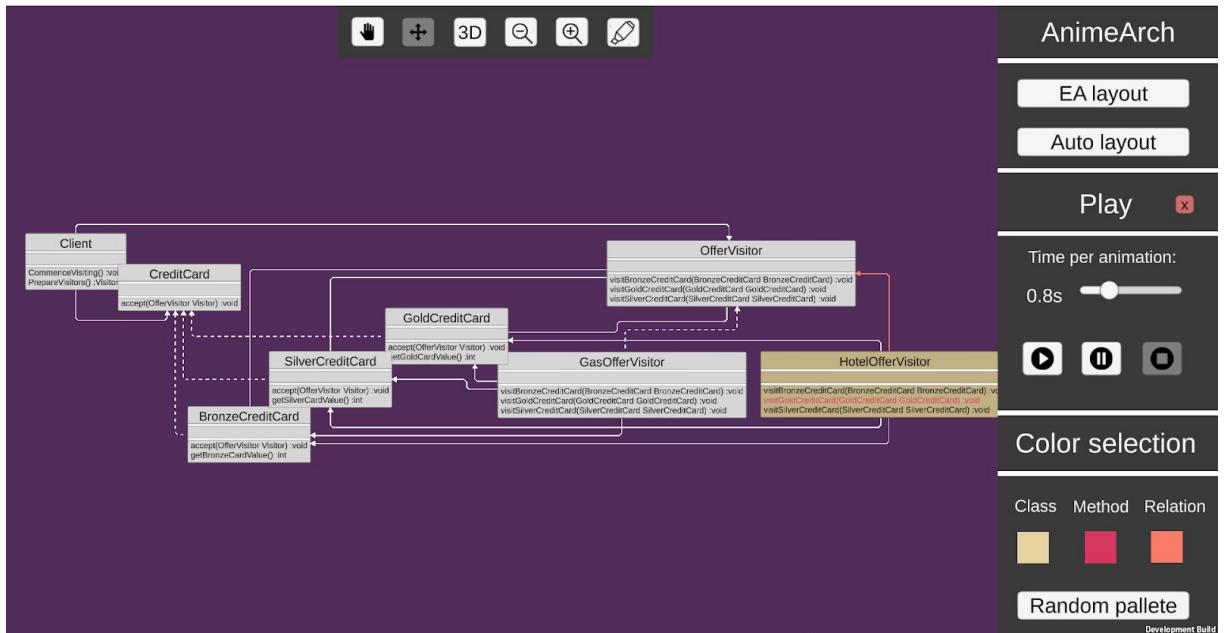
e. Po uložení animácie ju spustíme kliknutím na tlačidlo Play.



f. Na pravom paneli sa zobrazí menu prehrávania animácie. Kliknutím na tlačidlo s ikonou Play sa spustí prehrávanie animácie.



g. Animácia sa vykonáva s vnoreným cyklom "WHILE" a podmienkou "IF dovtedy, kým sú splnené podmienky.

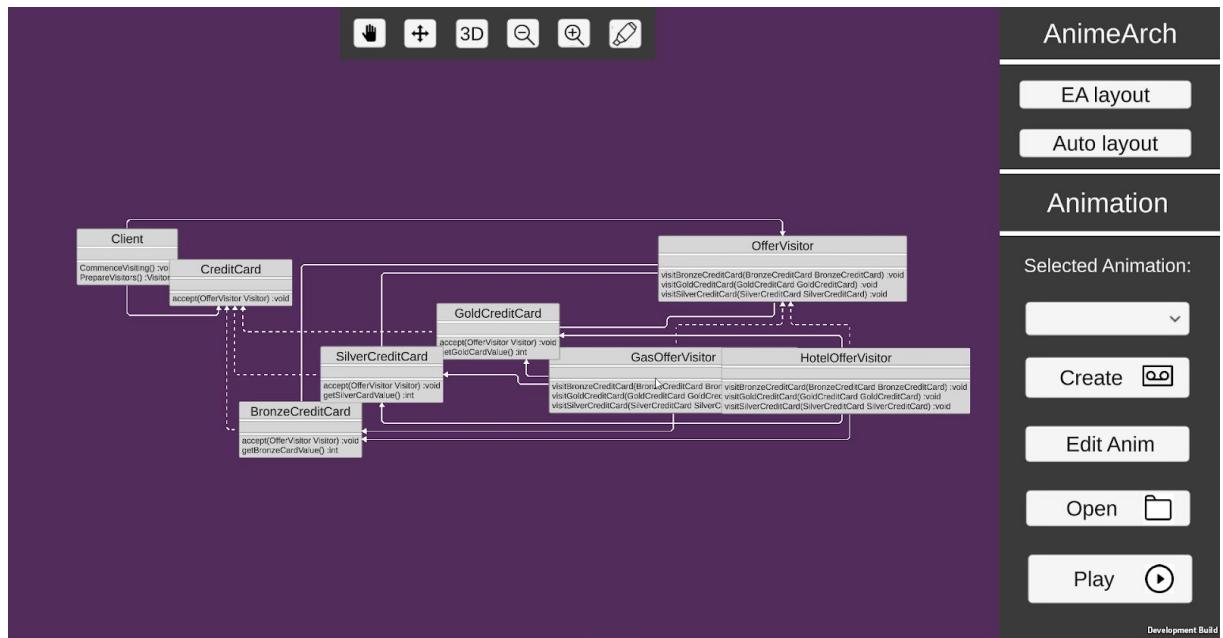


Použitý kód počas testovania:

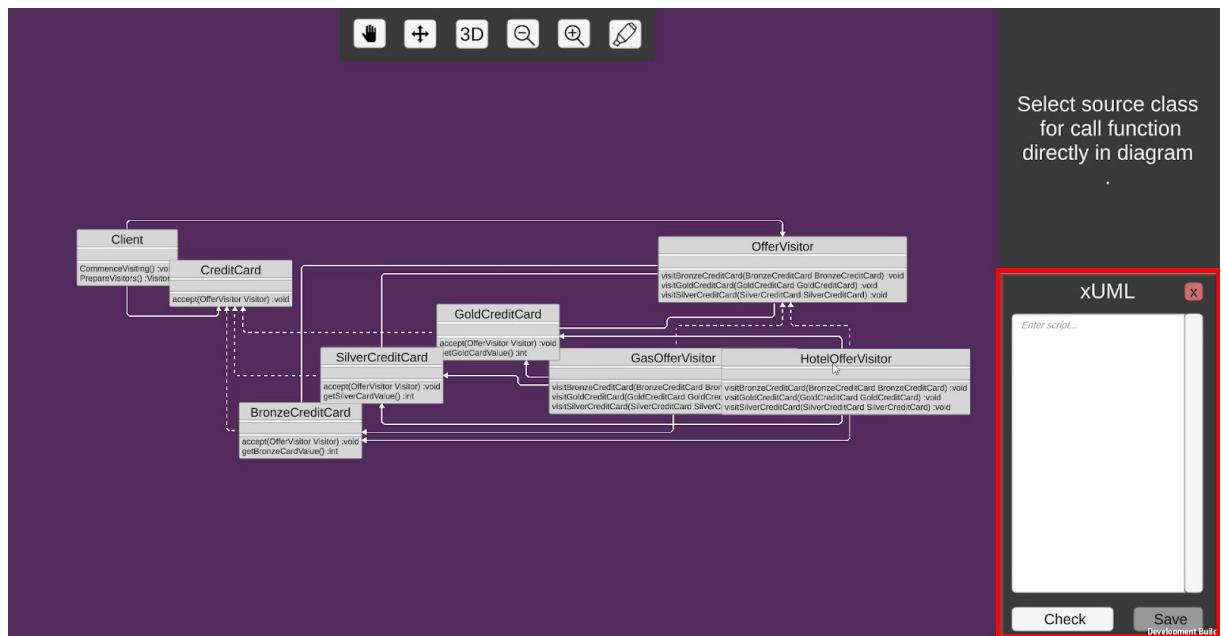
```
x = 0;
while x < 5
if x % 2 == 0
call from GasOfferVisitor::visitGoldCreditCard() to OfferVisitor::visitGoldCreditCard() across R8;
else
call from HotelOfferVisitor::visitGoldCreditCard() to OfferVisitor::visitGoldCreditCard() across R14;
end if;
x = x + 1;
end while;
```

8. Testovanie paralelizmu

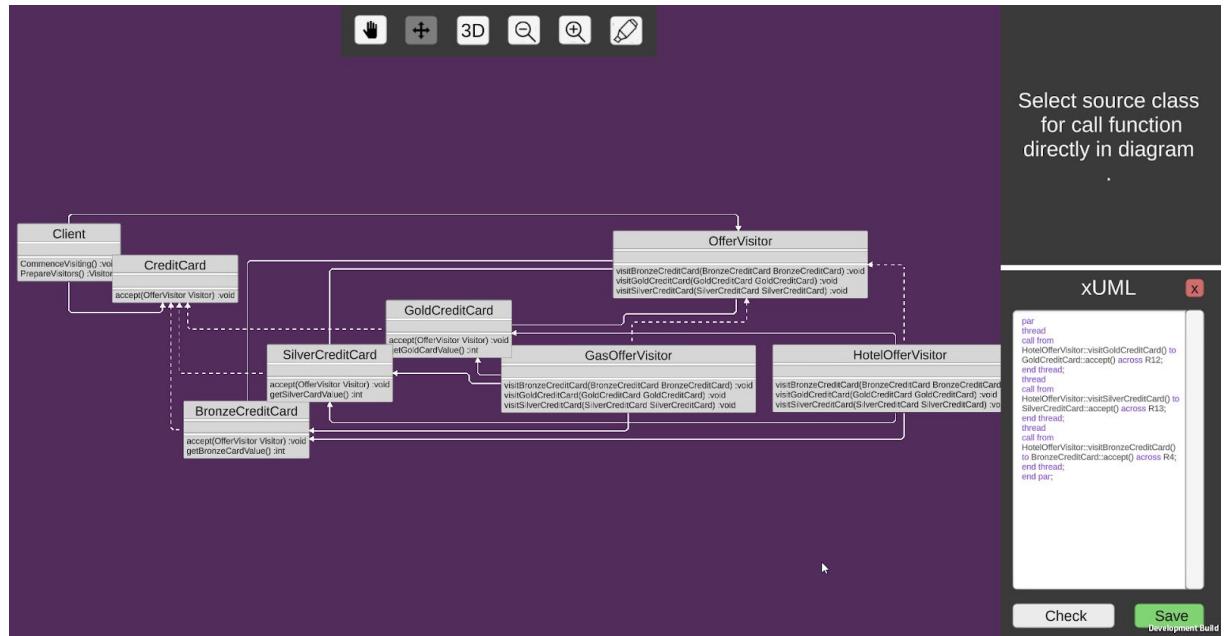
a. Klikneme na tlačidlo Create.



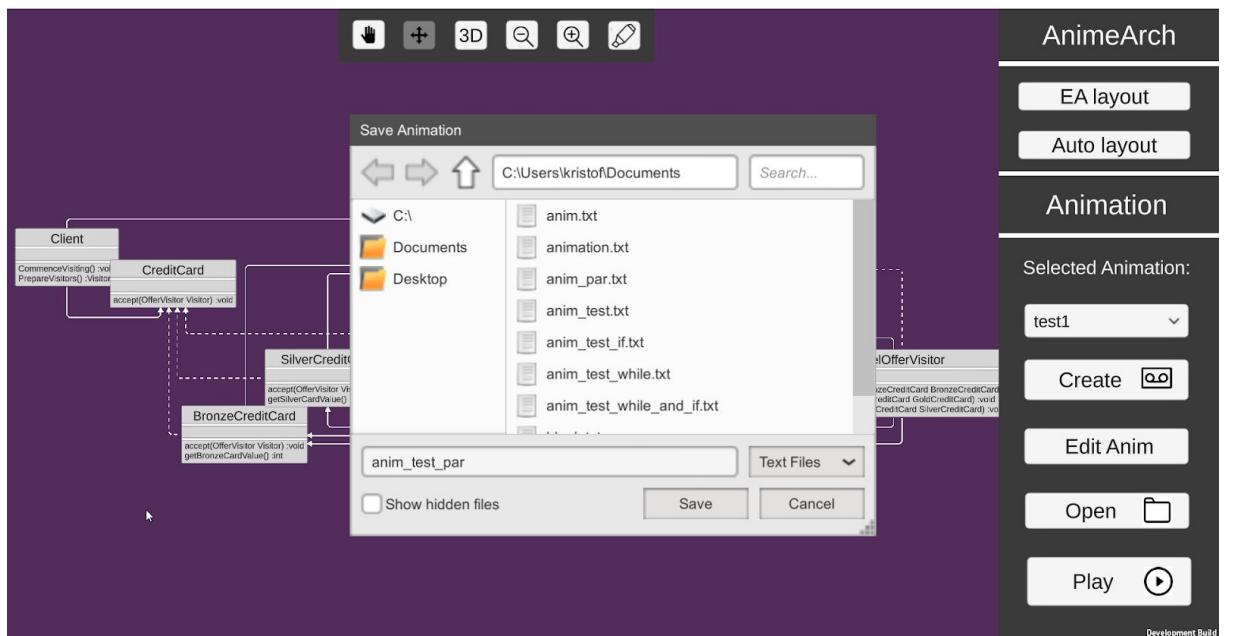
b. Do xUML panelu napíšeme kód pre paralelizmus.



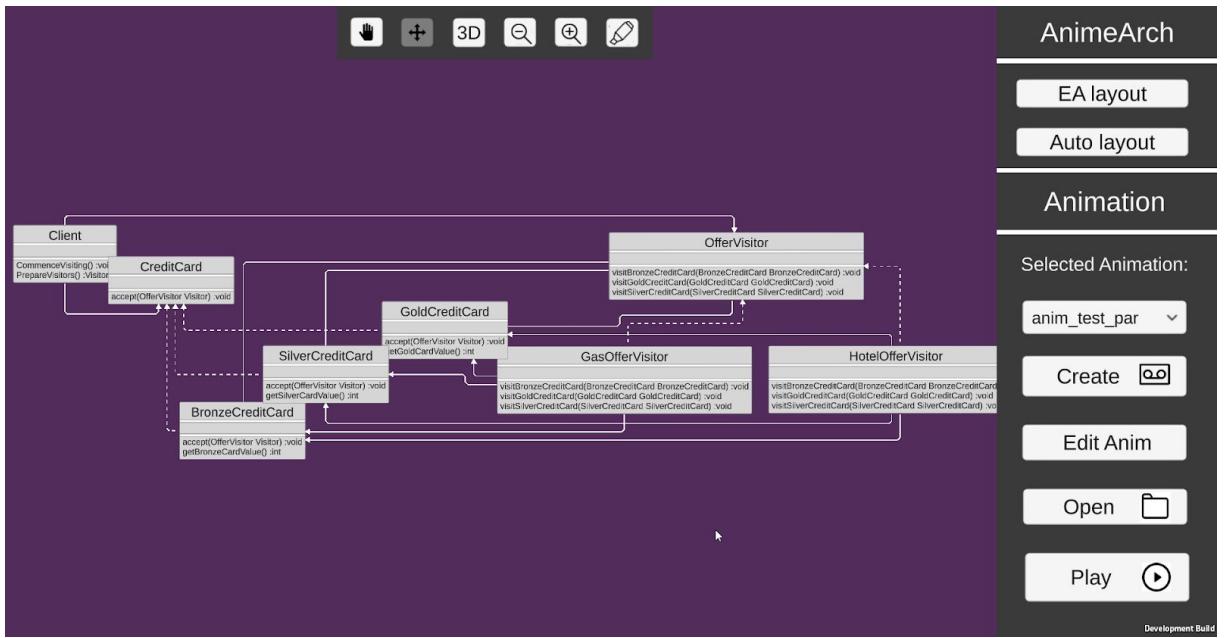
- c. Kliknutím na tlačidlo check sa skontroluje správnosť kódu a odblokuje sa tlačidlo Save.



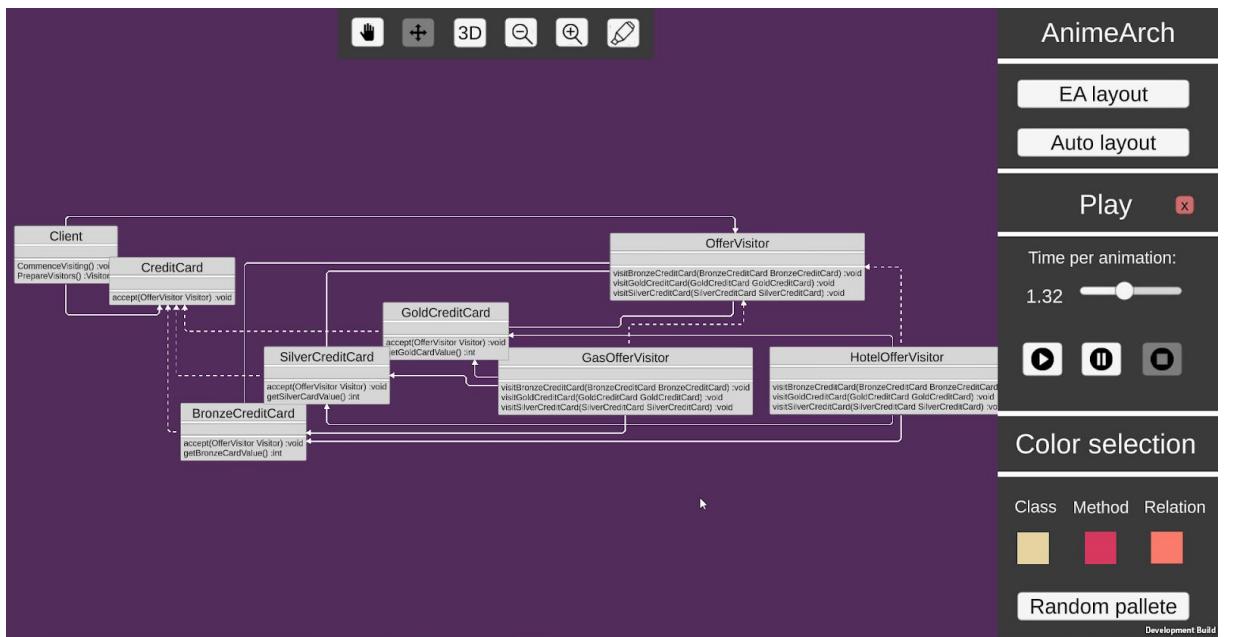
- d. Kód animácie je potrebné uložiť kliknutím na tlačidlo Save. Po kliknutí na tlačidlo Save sa zobrazí dialógové okno v ktorom vyberieme miesto na disku, kam chceme uložiť animáciu. Tá sa uloží vo formáte .txt.



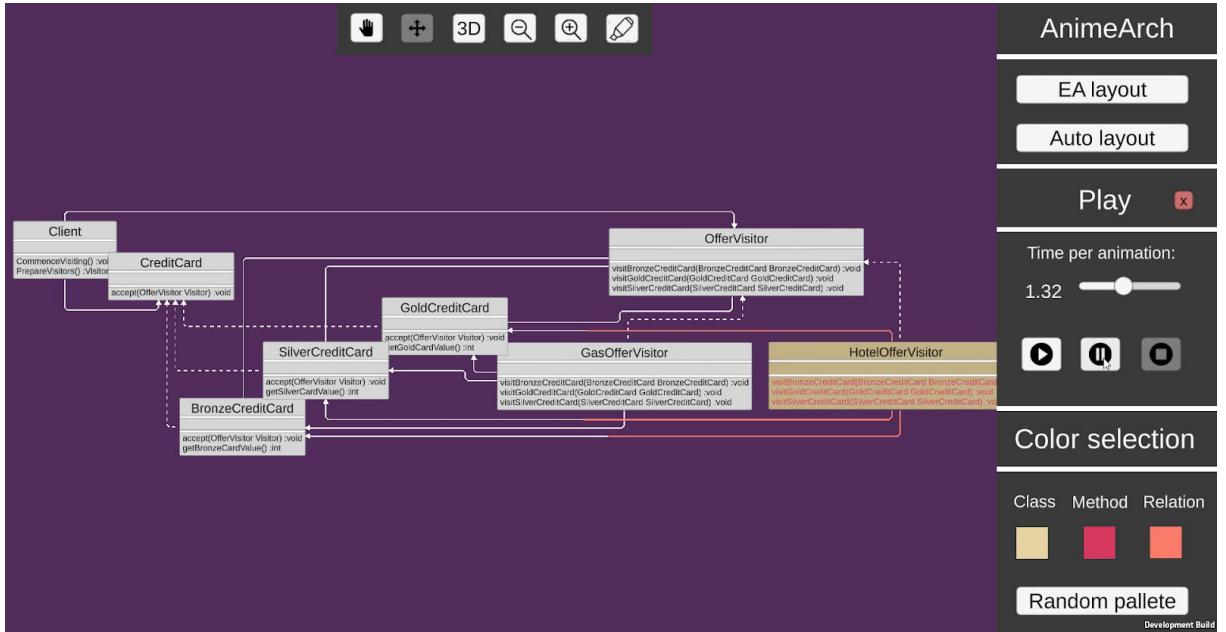
e. Po uložení animácie ju spustíme kliknutím na tlačidlo Play.



f. Na pravom paneli sa zobrazí menu prehrávania animácie. Kliknutím na tlačidlo s ikonou Play sa spustí prehrávanie animácie.



g. Animácia sa vykonáva paralelne.



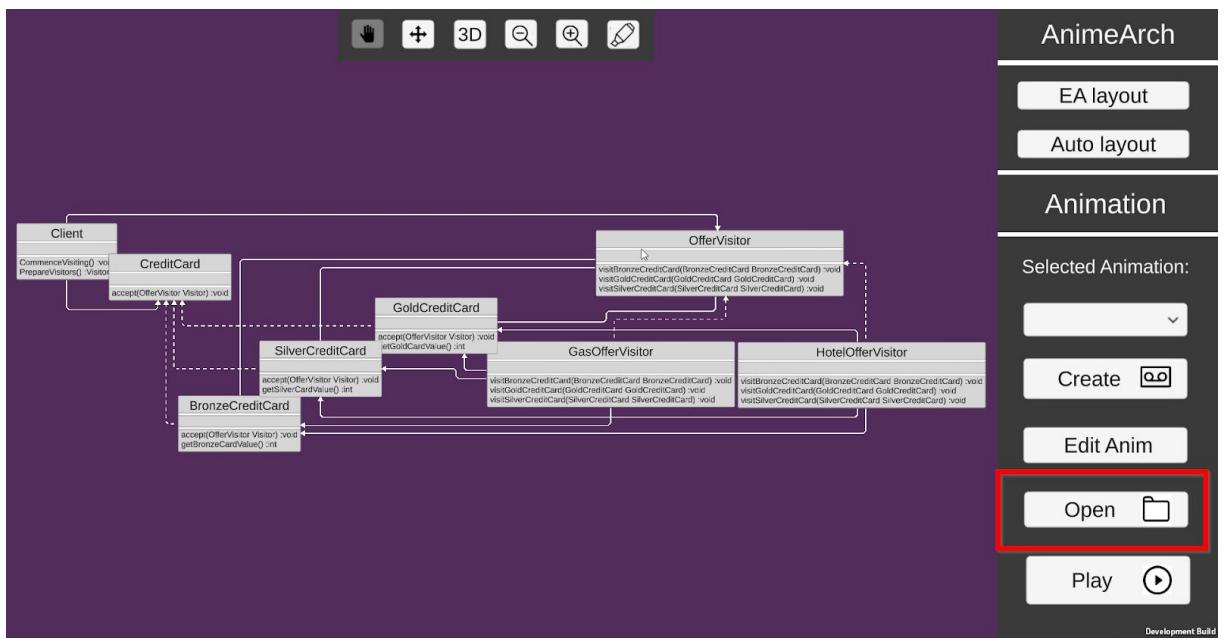
Použitý kód počas testovania:

```

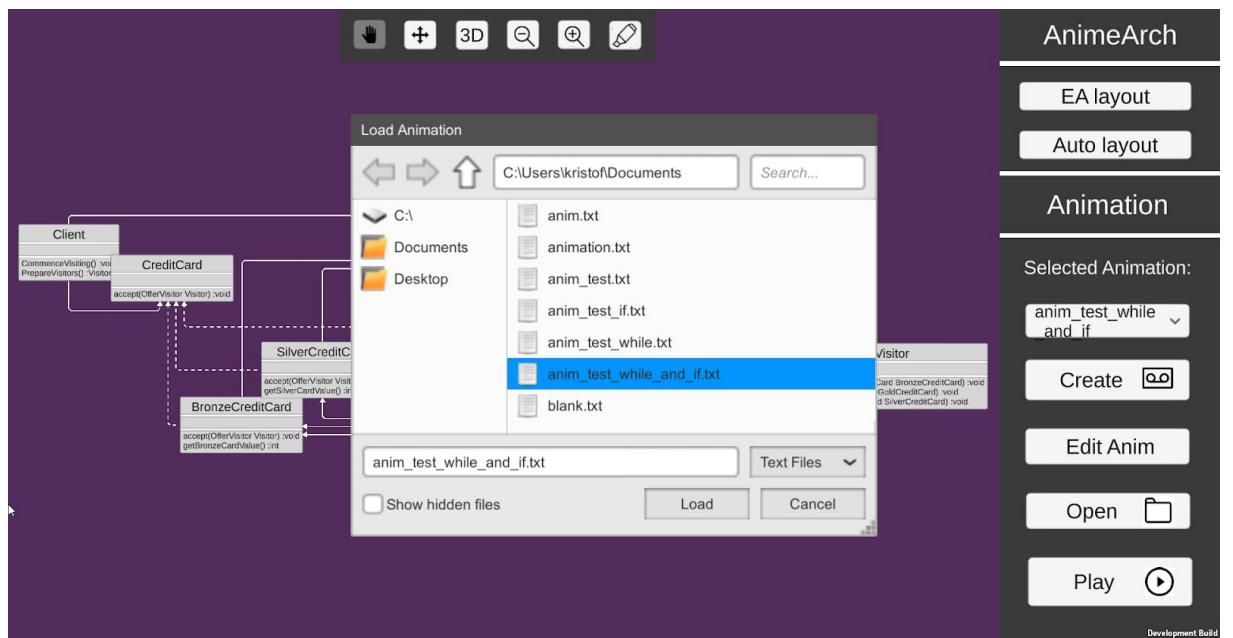
par
thread
call from HotelOfferVisitor::visitGoldCreditCard() to GoldCreditCard::accept() across R12;
end thread;
thread
call from HotelOfferVisitor::visitSilverCreditCard() to SilverCreditCard::accept() across R13;
end thread;
thread
call from HotelOfferVisitor::visitBronzeCreditCard() to BronzeCreditCard::accept() across R4;
end thread;
end par;
  
```

9. Testovanie načítania animácie

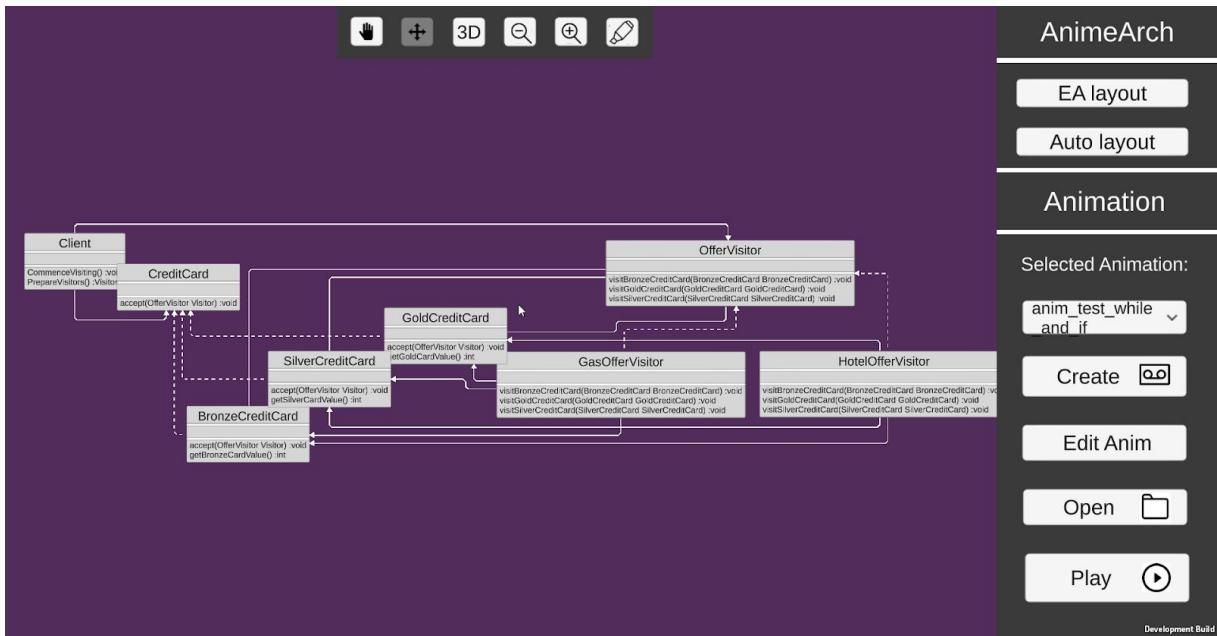
- Klikneme na tlačidlo Open.



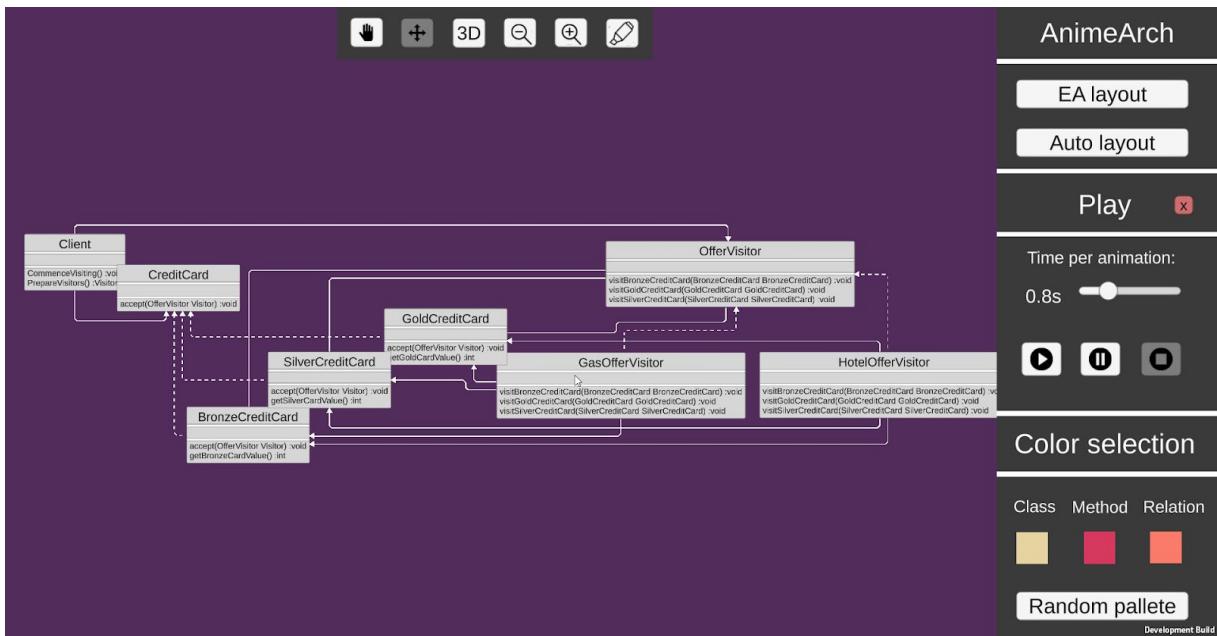
- Vyberieme uloženú animáciu a klikneme na tlačidlo Load.



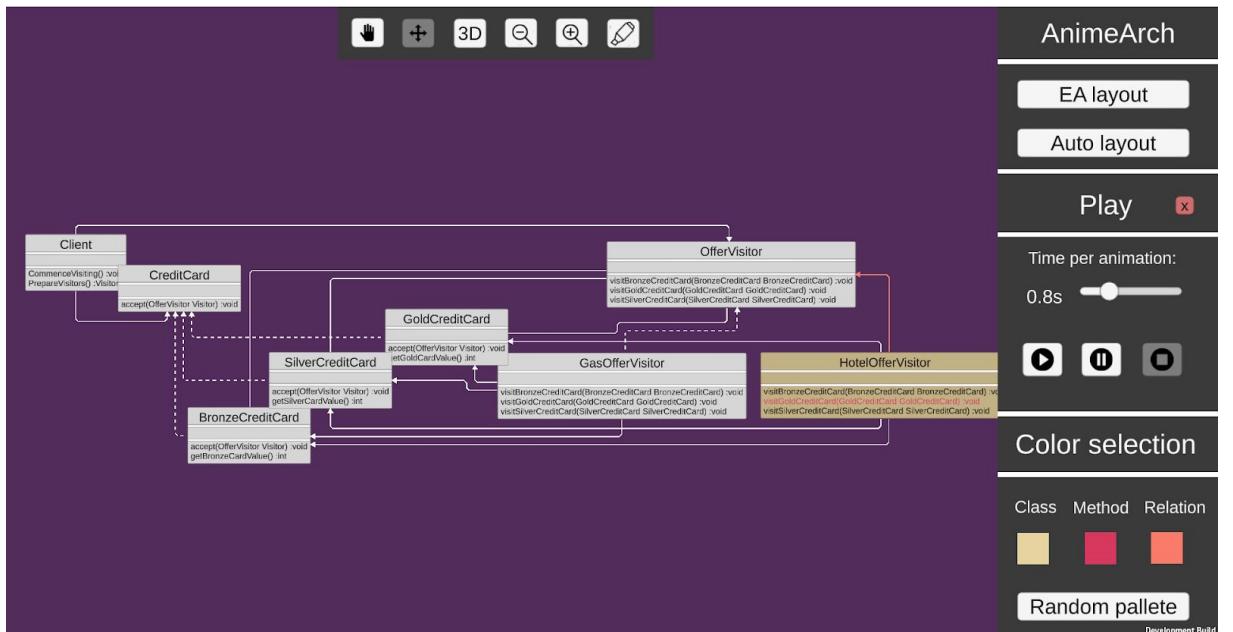
c. Po načítaní animácie ju spustíme kliknutím na tlačidlo Play.



d. Na pravom paneli sa zobrazí menu prehrávania animácie. Kliknutím na tlačidlo s ikonou Play sa spustí prehrávanie animácie.



e. Vykonáva sa načítaná animácia.



Príloha B - Dokumentácia modifikovaného jazyka OAL

Tento dokument popisuje všetky príkazy našej verzie OAL, ich štruktúru, význam a mapovanie na triedy nášho projektu. Má poslúžiť ako návod pre tvorcov parsera, ale pomôže komukoľvek, kto chce poznať našu verziu OAL. Manuál (ako aj samotná naša verzia OAL) vychádza z oficiálnej dokumentácie OAL.

Vysvetlivky syntaxe

Biele znaky

Je nutné uvažovať nad existenciou bielych znakov, t.j. medzery, tabulátora a konca riadku. Tie sa môžu vyskytovať kdekoľvek, okrem:

- v strede názvu alebo klúčového slova
- medzi dvoma dvojbodkami
- medzi dvoma znakmi porovnávacích operátorov.

<i><instance handle></i>	názov premennej ukazujúcej na jednu inštanciu nejakej triedy
<i><instance handle set></i>	názov premennej ukazujúcej na pole inštancií nejakej triedy
<i><handle></i>	<i><instance handle></i> alebo <i><instance handle set></i>
<i><keyletter></i>	názov triedy
<i><where expression></i>	boolean expression, kde sa navyše môže vyskytovať špeciálna premenná s názvom "selected"
<i><start></i>	instance handle, od ktorej sa začína navigácia vzťahmi pri selecte

<i><relationship specification></i>	názov vzťahu, v tvare R1, R2, R3...
<i><relationship link></i>	názov vzťahu v hranatých zátvorkách, v tvare [R1], [R2], [R3] ...
<i><method name></i>	názov metódy nejakej triedy
<i><boolean var></i>	názov premennej typu boolean
<i><arithmetic var></i>	názov premennej typu int alebo decimal
<i><string var></i>	názov premennej typu string
<i><boolean expression></i>	výraz, ktorý vracia boolean hodnotu
<i><arithmetic expression></i>	výraz, ktorý vracia int alebo decimal hodnotu
<i><string expression></i>	výraz, ktorý vracia string hodnotu

Komentáre

Akýkoľvek text za dvojicou znakov // je považovaný za komentár až do konca riadka, nie je súčasťou príkazov. Výnimkou je, ak sa tieto dva znaky nachádzajú v reťazci, teda napríklad “asldad//+5asd”.

Kľúčové slová

Kľúčové slová jazyka môžu byť napísané *uppercase*, *lowercase* alebo *so začiatočným písmenom veľkým a zvyšnými malými písmenami*.

across and any assign

break by

cardinality call continue create

delete each elif

else empty end

false for from

if in instance instances

many not not_empty object

of or

par relate related

select selected

to thread true

unrelate using where while

Názvy

Názvy premenných/tried/metód/atribútov

- obsahujú len znaky [a-z][A-Z][0-9][_#]
- nesmú začínať číselným znakom.

Dopyty

OAL je objektovo-orientovaný. Inštancie tried však nevznikajú a nezanikajú v scopoch, ale sú držané v globálnej datábaze tried a inštancií, podobne aj vzťahy medzi nimi. V scopoch vznikajú a zanikajú len premenné ukazujúce na tieto inštancie.

Jazyk preto obsahuje dopyty nad touto databázou tried, fungujúce podobne ako dopyty o akejkoľvek inej databázy.

Create class instance

Dopyt pre vytvorenie novej inštancie danej triedy a jej priradenie do premennej.

Syntax: *create object instance <instance handle> of <keyletter>;*

create object instance of <keyletter>;

Mapuje sa na triedu: EXECommandQueryCreate

Príklady:

- *create object instance observer1 of Observer;*
- *create object instance my_dog of Dog;*
- *create object instance myUserAccount of UserAccount;*
- *create object instance of Visitor;*

Create relationship between class instances

Dopyt pre vytvorenie vzťahu medzi dvoma existujúcimi inštanciami tried.

Syntax: *relate <instance handle> to <instance handle> across <relationship specification>;*

Mapuje sa na triedu: EXECommandQueryRelate

Príklady:

- relate subject to observer across R15;
- relate dog to owner across R7;

Select class instance - by class

Dopyt pre získanie inštancie/inštancií danej triedy z databázy tried. Ak je použitá klauzula *where*, musí obsahovať aspoň raz slovo *selected*.

Syntax: *select any <instance handle> from instances of <keyletter> [where <where expression>];*

select many <instance handle set> from instances of <keyletter> [where <where expression>];

Mapuje sa na triedu: EXECommandQuerySelect

Príklady:

- *select any dog from instances of Dog;*
- *select many all_dogs from instances of Dog;*
- *select any young_dog from instances of Dog where selected.age < 5;*
- *select many young_rexes from instances of Dog where selected.age < 5 AND selected.name == "Rex";*

Select class instance - by relationship specification

Dopyt pre získanie inštancie/inštancií triedy danej reťazom vztáhov z databázy tried. Ak je použitá klauzula *where*, musí obsahovať aspoň raz slovo *selected*.

Syntax: *select {any|many} <instance handle> related by <start> -> <relationship link> [-> <relationship link> ...]; [where <where expression>];*

Mapuje sa na triedu: EXECommandQuerySelectRelatedBy

Príklady:

- *select any my_random_dog related by current_owner->Dog[R4];*
- *select many all_my_dogs related by owner_veronica->Dog[R4];*
- *select any my_random_young_dog related by current_owner->Dog[R4] where selected.age < 5;*
- *select many my_young_dogs related by current_owner->Dog[R4] where selected.age < 5;*
- *select many all_rexos_in_village related by current_village->House[R12]->Person[R176]->Dog[R1] where selected.name == "Rex";*

Delete class instance

Dopyt pre odstránenie danej inštancie triedy.

Syntax: *delete object instance <instance handle>;*

Mapuje sa na triedu: EXECommandQueryDelete

Príklady:

- *delete object instance current_user;*
- *delete object insatnce observer3;*

Delete relationship between class instances

Dopyt pre odstránenie vzťahu medzi dvoma existujúcimi inštanciami tried.

Syntax: *unrelate <instance handle> from <instance handle> across <relationship specification>;*

Mapuje sa na triedu: EXECommandQueryUnrelate

Príklady:

- *unrelate subject from observer across R15;*

- *unrelate dog from owner across R7;*

Priradenie

Priradenie priradzuje hodnotu do premennej, alebo do atribútu premennej. Premenné nemajú explicitne uvedené typy, ani ich netreba deklarovať. Premenná vzniká v momente priradenia a jej typ je daný typom prvej hodnoty do nej priradenej. Priradovať možno len primitívne hodnoty/výrazy, t.j. string, boolean, int a decimal.

Priradenie do premennej

Priradenie do premennej primitívneho typu. Klúčové slovo *assign* môže, ale nemusí byť pred príkazom. Jedná sa o syntactic sugar.

Syntax: *[assign] <boolean var> = <boolean expression>;*

[assign] <arithmetic var> = <arithmetic expression>;

[assign] <string var> = <string expression>;

Mapuje sa na triedu: EXECommandAssignment

Príklady:

- $x = 15.0 * y;$
- $x = y;$
- $x = "ahoj" + "te";$
- $assign result = (x < 15 \text{ or } y < 66) \text{ and } is_allowed;$
- $i = i + 1;$

Priradenie do atribútu inštancie triedy

Priradenie do premennej primitívneho typu. Klúčové slovo *assign* môže, ale nemusí byť pred príkazom. Jedná sa o syntactic sugar.

Syntax: [assign] <instance handle>. <attribute> = <expression>;

Mapuje sa na triedu: EXECommandAssignment

Príklady:

- *object.coordinate_x = 17.22;*
- *archer.hp = archer.hp * 0.50 - 16;*
- *assign dog.name = "Koronko";*
- *assign door.locked = not (door.key == used_key);*

Volanie metód

Volanie metód

Jeden krok animácie, teda volanie metódy z inej metódy. Vzťah môže, ale nemusí byť špecifikovaný, ak volaná aj volajúca trieda je tá istá. Inak musí byť špecifikovaný.

Syntax: call from <keyletter>::<method name>() to <keyletter>::<method name>()
[across <relationship specification>];

Mapuje sa na triedu: EXECommandCall

Príklady:

- *call from Observer::init() to Subject::register() across R4;*
- *call from Subject::register() to Subject::addObserver();*
- *call from Subject::register() to Subject::addObserver() across R7;*

Jednoduché riadiace konštrukcie

Pravidlá jazyka OAL nevynucujú, aby *break* a *continue* boli len vnútri cyklu. Môžu byť teda kdekoľvek.

Break

Prerušenie cyklu.

Syntax: *break;*

Mapuje sa na triedu: *EXECCommandBreak*

Príklady:

- *break;*

Continue

Prerušenie aktuálnej iterácie cyklu.

Syntax: *continue;*

Mapuje sa na triedu: *EXECCommandBreak*

Príklady:

- *continue;*

Výrazy

Výraz je zápis, ktorého vypočítaním získame bool, int, decimal alebo string hodnotu. Môže využívať zátvorky, unárne, binárne a n-árne operátory a ich operandy. Výraz nesmie byť prázdny, t.j. musí obsahovať aspoň 1 koncovú hodnotu. Každý operátor musí mať vhodný počet operandov.

Je nutné uvažovať nad existenciou zátvoriek. Zátvorky môžu ovplyvniť podobu stromu.

Koncová hodnota

Predstavuje najmenšiu jednotku výrazu, teda primitívnu hodnotu typu bool, int, decimal, string. Tiež to môže byť názov premennej, atribútu, alebo kľúčové slovo *UNDEFINED*, ktoré predstavuje neinicializovanú hodnotu atribútu. Tiež to môže byť kľúčové slovo *selected*, ktoré sa vyskytuje vo výrazoch *where_condition* v príkazoch *select*.

Syntax: {<bool>|<int>|<decimal>|<string>|<variable name>|<attribute name>|*UNDEFINED*|*selected*}

Mapuje sa na triedu: EXEASTNodeLeaf

Príklady:

- 0
- -5
- 42
- 13.5
- 14.0
- 1.0000000000000099
- -11.902
- TRUE
- FALSE
- “ahoj”
- “Dnes je dazdivo.\n”
- “Povedal:\\"Prsi!\\\".”
- observer
- count
- id
- name

- *UNDEFINED*
- *selected*

Operácia

Predstavuje zložitejšiu jednotku výrazu, teda operáciu nad operandmi. Operátor môže byť unárny (1 operand), binárny (2 operandy) alebo n-árny (2 a viac operandov). Operandom môže byť koncová hodnota alebo ďalšia operácia.

V OAL je zápis operácií infixný, teda operátor sa nachádza *medzi* operandmi.

Syntax: *<unary operator> <expression>*

<expression> <binary operator> <expression>

<expression> <n - ary operator> <expression> [<n - ary operator> <expression>

....]

Mapuje sa na triedu: EXEASTNodeComposite

Príklady:

- *NOT locked*
- *dog1.age > dog2.age*
- *5 + 6 * 4*
- *(5 + 6) * 4*
- *x + cardinality dogs*
- *NOT (x < 10 and y < 20 and z < 30) or NOT door.locked*

Typy a operátory

Unárne operátory

Syntax	Príklady	Návratový typ
empty <handle>	empty dog empty dogs	boolean
not_empty <handle>	not_empty dog not_empty dogs	boolean
cardinality <handle>	cardinality dog cardinality dogs	int
not <boolean expression>	not TRUE not toilet.free not (x<15 or y>700)	boolean

Binárne operátory

Syntax	Prípustné typy argumentov	Návratový typ
==	int, decimal, string, boolean	boolean
!=	int, decimal, string, boolean	boolean
<	int, decimal, string	boolean
>	int, decimal, string	boolean
<=	int, decimal, string	boolean
>=	int, decimal, string	boolean

N - árne operátory

Syntax	Prípustné typy argumentov	Návratový typ
+	int, decimal, string	int, decimal, string

-	int, decimal, string	int, decimal, string
*	int, decimal, string	int, decimal, string
/	int, decimal, string	int, decimal, string
%	int, decimal, string	int, decimal, string
or	boolean	boolean
and	boolean	boolean

Zložené príkazy

If - elif - else

Syntax: *if*<boolean expression>

<statements>

end if;

if(<boolean expression>) //brackets are not compulsory

<statements>

elif(<boolean expression>) //0-n elifs

<statements>

else //optional

<statements>

end if;

Mapuje sa na triedu: EXEScopeCondition

Príklady:

- *if* $x > 5$

$x = x - 1;$

end if;

- *if* (*door.lock_id == key.id*)

door.opened = TRUE;

response_text = "Door opens";

else

```

response_text = “Wrong key”;

end if;

• if(action.id == 0)

    x = 15 * y;

    elif(action.id == 1)

        x = 7,5 * y - 16;

    elif(action.id == 2)

        x = 0,2 * y + 1,0;

    end if;

• if count > 5

    if(x == 0)

        y = 0;

    else

        y = 1 / x;

    end if;

end if;

```

While

Syntax: *while (<boolean expression>)*

<statements>

end while;

Mapuje sa na triedu: EXEScopeLoopWhile

Príklady:

- *while (x < 100)*

```
create object instance enemy of Enemy;  
enemy.name = "Beholder";  
enemy.hp = x * 2;
```

end while;

- *while (TRUE)*

```
x = x * 2;  
if x > 4096  
    break;  
end if;  
end while;
```

Foreach

Syntax: *for each <instance handle> in <instance handle set>*

<statements>

end for;

Mapuje sa na triedu: EXEScopeForEach

Príklady:

- *foreach observer in my_observers*

call from Subject::Notify() to Observer::Update() across R4;

notified_observer_count = notified_observer_count + 1;

end for;

- *foreach dog in my_dogs*

relate dog to my_wife across R8;

end for;

Parallel - thread

Syntax: *par*

```
thread //2 - n threads  
  <statements>  
end thread;  
  
thread  
  <statements>  
end thread;  
  
end par;
```

Mapuje sa na triedu: EXEScopeParallel (par), EXEScope (thread)

Príklady:

- *par*
 thread
 call from Clock::Update() to AnalogWidget::Update() across R8;
 end thread;
 thread
 call from Clock::Update() to DigitalWidget::Update() across R12;
 end thread;
 thread
 call from Clock::Update() to CalendarWidget::Update() across R14;
 end thread;
 end par;

EBNF Pôvodného OAL

Uvádzame EBNF pôvodného OAL, ktoré môže pomôcť pri tvorbe EBNF našej verzie. Obsahuje veľa syntaxe, ktorá v našej verzii neexistuje. Neobsahuje našu syntax príkazov *Volanie metód(vysvietenie)* a *Parallel - thread*.

```
action ::= block EOF ;  
  
block ::= ( statement )* ;  
  
statement ::= ( implicit_ib_transform_statement | function_statement  
implicit_assignment_statement |  
implicit_invocation_statement | assignment_statement | control_statement | break_statement  
|  
bridge_statement | send_statement | continue_statement | create_object_statement |  
create_event_statement | delete_statement | for_statement | generate_statement | if_statement  
|  
relate_statement | return_statement | select_statement | transform_statement |  
while_statement |  
unrelate_statement | debug_statement | empty_statement ) Semicolon ;  
  
assignment_statement ::= ASSIGN assignment_expr ;  
  
break_statement ::= BREAK ;  
  
bridge_statement ::= BRIDGE ( ( ( member_access | param_data_access ) EQUAL  
bridge_invocation ) |  
bridge_invocation ) ;  
  
send_statement ::= SEND ( ( ( member_access | param_data_access ) EQUAL
```

message_invocation) | *message_invocation*) ;

continue_statement ::= *CONTINUE* ;

create_event_statement ::= *CREATE EVENT INSTANCE local_variable OF event_spec* ;

create_object_statement ::= *CREATE OBJECT INSTANCE ((local_variable OF)=> local_variable)? OF object_keyletters* ;

delete_statement ::= *DELETE OBJECT INSTANCE inst_ref_var* ;

empty_statement ::= ;

for_statement ::= *FOR EACH local_variable IN inst_ref_set_var block END FOR* ;

generate_statement ::= *GENERATE (event_spec | member_access)* ;

if_statement ::= *IF expr block ((ELIF expr block)+)? (ELSE block)? END IF* ;

implicit_assignment_statement ::= *assignment_expr* ;

implicit_invocation_statement ::= *invocation* ;

implicit_ib_transform_statement ::= *transform_ib_invocation* ;

relate_statement ::= *RELATE inst_ref_var TO inst_ref_var ACROSS relationship (DOT phrase)? (USING assoc_obj_inst_ref_var)? ;*

return_statement ::= *RETURN (expr)? ;*

select_statement ::= *SELECT (ONE local_variable object_spec | ANY local_variable object_spec | MANY local_variable object_spec) ;*

```

transform_statement ::= TRANSFORM ( ( member_access | param_data_access ) EQUAL
transform_invocation ) | transform_invocation ) ;

function_statement ::= DOUBLECOLON function_invocation ;

unrelate_statement ::= UNRELATE inst_ref_var FROM inst_ref_var ACROSS relationship (
DOT

phrase)? ( USING assoc_obj_inst_ref_var )? ;

while_statement ::=  WHILE expr block END WHILE ;

assignment_expr ::= ( member_access EQUAL expr |( PARAM DOT )=>
param_data_access EQUAL
expr ) ;

bridge_invocation ::= ee_keyletters DOUBLECOLON bridge_function LPAREN (
invocation_parameters)?

RPAREN ;

message_invocation ::= interface_or_port_name DOUBLECOLON message_name LPAREN
(invocation_parameters)? RPAREN (TO ( rval ))? ;

invocation ::= identifier DOUBLECOLON invocation_function LPAREN (
invocation_parameters)?

RPAREN ;

bridge_expr ::= BRIDGE bridge_invocation ;

invocation_expr ::= invocation ;

enumerator_access ::= enum_data_type DOUBLECOLON enumerator ;

```

event_spec ::= *event_label* (*TIMES*)? (*COLON event_meaning*)? (*LPAREN* (*supp_data*)? *RPAREN*)?

TO (((*object_keyletters CLASS*)=> *object_keyletters CLASS* | *object_keyletters CREATOR*) |

(*inst_ref_var_or_ee_keyletters*));

invocation_parameters ::= *data_item COLON expr* (*COMMA data_item COLON expr*)*;

inst_ref_var_or_ee_keyletters ::= (*local_variable* | *GENERAL_NAME* | *kw_as_id2*);

interface_or_port_name ::= *general_name* ;

message_name ::= *general_name* ;

instance_chain ::= (*ARROW object_keyletters LSQBR relationship DOT phrase*)? *RSQBR*)+ ;

object_spec ::= (*RELATED BY local_variable instance_chain (where_spec)? | FROM INSTANCES OF*

object_keyletters (where_spec)?);

supp_data ::= *supp_data_item COLON expr* (*COMMA supp_data_item COLON expr*)*;

function_invocation ::= *function_function LPAREN invocation_parameters ? RPAREN* ;

transform_ib_invocation ::= *inst_ref_var DOT transformer_function LPAREN (invocation_parameters)? RPAREN* ;

RPAREN ;

transform_invocation ::= *object_keyletters DOUBLECOLON transformer_function LPAREN (invocation_parameters)? RPAREN* ;

where_spec ::= *WHERE expr* ;

```

assoc_obj_inst_ref_var ::= inst_ref_var ;

bridge_function ::= function_name ;

invocation_function ::= function_name ;

data_item ::= data_item_name ;

data_item_name ::= general_name ;

enum_data_type ::= general_name ;

enumerator ::= general_name ;

keyletters ::= general_name ;

ee_keyletters ::= keyletters ;

event_label ::= general_name ;

event_meaning ::= ( phrase ) ;

general_name ::= ( limited_name | GENERAL_NAME | kw_as_id1 | kw_as_id2 | kw_as_id3
) ;

svc_general_name ::= ( limited_name | GENERAL_NAME | kw_as_id1 | kw_as_id2 |
kw_as_id3 |
kw_as_id4 ) ;

limited_name ::= ID | RELID ;

inst_ref_set_var ::= local_variable ;

inst_ref_var ::= local_variable ;

local_variable ::= root_element_label ;

root_element_label ::= ( SELECTED | SELF | limited_name | kw_as_id1 ) ;

```

```

element_label ::= general_name ;

function_name ::= general_name ;

svc_function_name ::= svc_general_name ;

identifier ::= general_name ;

object_keyletters ::= keyletters ;

phrase ::= ( TICKED_PHRASE | svc_general_name ) ;

relationship ::= RELID ;

supp_data_item ::= data_item_name ;

function_function ::= svc_function_name ;

transformer_function ::= function_name ;

expr ::= sub_expr ;

sub_expr ::= conjunction ( OR conjunction )* ;

conjunction ::= relational_expr ( AND relational_expr )* ;

relational_expr ::= addition ( comparison_operator addition )? ;

addition ::= multiplication ( plus_or_minus multiplication )* ;

multiplication ::= boolean_negation | sign_expr ( mult_op sign_expr )* ;

sign_expr ::= ( plus_or_minus )? term ;

boolean_negation ::= NOT term ;

term ::= cardinality_op | empty_op | not_empty_op | rval | LPAREN ( assignment_expr |
expr ) RPAREN ;

cardinality_op ::= CARDINALITY local_variable ;

```

$\text{empty_op} ::= \text{EMPTY local_variable} ;$
 $\text{not_empty_op} ::= \text{NOT_EMPTY local_variable} ;$
 $\text{instance_start_segment} ::= \text{root_element_label} (\text{array_refs})? ;$
 $\text{instance_access_segment} ::= \text{element_label} (\text{array_refs})? ;$
 $\text{member_access} ::= \text{instance_start_segment} (\text{DOT instance_access_segment})^* ;$
 $\text{param_data_access} ::= \text{PARAM DOT member_access} ;$
 $\text{event_data_access} ::= \text{RCVD_EVT DOT member_access} ;$
 $\text{array_refs} ::= (\text{LSQBR expr RSQBR})^+ ;$
 $\text{rval} ::= \text{DOUBLECOLON function_invocation} \mid \text{transform_ib_invocation} \mid \text{invocation_expr} \mid$
 $\text{enumerator_access} \mid \text{member_access} \mid \text{constant_value} \mid (\text{RCVD_EVT DOT}) \Rightarrow$
 $\text{event_data_access} \mid$
 $\text{bridge_expr} \mid (\text{PARAM DOT}) \Rightarrow \text{param_data_access} \mid \text{QMMARK} ;$
 $\text{constant_value} ::= \text{FRACTION} \mid \text{NUMBER} \mid \text{STRING} \mid \text{TRUE} \mid \text{FALSE} ;$
 $\text{comparison_operator} ::= \text{DOUBLEEQUAL} \mid \text{NOTEQUAL} \mid \text{LESSTHAN} \mid \text{LE} \mid \text{GT} \mid \text{GE} ;$
 $\text{plus_or_minus} ::= \text{PLUS} \mid \text{MINUS} ;$
 $\text{mult_op} ::= \text{TIMES} \mid \text{DIV} \mid \text{MOD} ;$

Príloha C - Protokol z externého testovania

Externé testovanie aplikácie AnimArch vykonal tím č. 2 pod vedením Juraja Vincúra z predmetu tímový projekt. Testovanie bolo vykonané na základe testovacích scenárov v časti [Príloha A - Testovacie scenáre](#). Uvádzame protokol vypracovaný na základe získaných hlásení.

Pozitíva:

1. Pokiaľ postupujeme podľa inštalačných krokov tak je to jednoduché a rýchle. Žiadny problém nevznikol a plugin po inštalácii fungoval.
2. UI pri diagrame otvorenom cez plugin je intuitívne a teda sa s ním ľahko pracuje.
3. Animácie sú dobre zobrazené.
4. Ak sa riadim podľa test casov tak fungujú v poriadku.

Negatíva

1. Pokiaľ sú názvy v diagrame s diakritikou, v plugine sa nezobrazujú.
2. Stalo sa, že pri komplikovanejšom diagrame ho plugin nezobrazil.
3. Ak má trieda viac ako 5 metód, tak sa nezobrazia na výber všetky pri vytváraní animácie.
4. Stalo sa, že pri výbere metódy triedy bola po kliknutí zvolená trieda, ktorá bola za touto časťou UI.
5. Niekedy nebolo možné vytvoriť animáciu medzi triedami, konkrétnie (source: Pouzivatelske rozhranie - objednavka -> metóda potvrdit objednavku target: objednavka -> potvrdit objednavku)

Príloha D - Gramatika našej verzie OAL

grammar OAL;

lines

```
:      line+ EOF
;
;
```

line

```
:      exeCommandQueryCreate
|      exeCommandQueryRelate
|      exeCommandQuerySelect
|      exeCommandQuerySelectRelatedBy
|      exeCommandQueryDelete
|      exeCommandQueryUnrelate
|      exeCommandAssignment
|      exeCommandCall
|      continueCommand
|      breakCommand
|      whileCommand
|      ifCommnad
|      foreachCommand
|      parCommand
;
;
```

parCommand

```
:      'par'('thread' line+ 'end thread;')+ 'end par;'
```

ifComnnad

```
:      'if'('expr')' line* ('elif'('expr')' line+)* ('else' line+)? 'end if;'  
;  
;
```

whileCommand

```
:      'while"('expr')' line+ 'end while;'  
;  
;
```

foreachCommand

```
:      'for each' variableName 'in' variableName line+ 'end for;'  
;  
;
```

continueCommand

```
:      'continue;'  
;  
;
```

breakCommand

```
:      'break;'  
;  
;
```

exeCommandQueryCreate

```
:      'create object instance 'instanceHandle' of 'keyLetter';'  
|      'create object instance of 'keyLetter';'  
;  
;
```

exeCommandQueryRelate

: 'relate 'instanceHandle' to 'instanceHandle' across
'relationshipSpecification';'
;
;

exeCommandQuerySelect

: 'select any 'instanceHandle' from instances of 'keyLetter (' where '
whereExpression)?';'
| 'select many 'instanceHandle' from instances of 'keyLetter (' where '
whereExpression)?';'
;
;

exeCommandQuerySelectRelatedBy

: 'select any 'instanceHandle' related by 'start'->'className
relationshipLink ('->'className relationshipLink)* (' where ' whereExpression)?';'
| 'select many 'instanceHandle' related by 'start'->'className
relationshipLink ('->'className relationshipLink)* (' where ' whereExpression)?';'
;
;

exeCommandQueryDelete

: 'delete object instance 'instanceHandle';'
;
;

exeCommandQueryUnrelate

: 'unrelate 'instanceHandle' from 'instanceHandle' across
'relationshipSpecification';'
;
;

exeCommandAssignment

```
:   variableName'='expr;'  
|   'assign 'variableName'='expr';'  
|   'instanceHandle'.attribute'='expr';'  
|   'assign 'instanceHandle'.attribute'='expr';'  
;  
;
```

exeCommandCall

```
:   'call from keyLetter::'methodName() to 'keyLetter::'methodName()'(''  
across 'relationshipSpecification)?';'  
;  
;
```

commands

```
:   'create object instance 'instanceHandle' of 'keyLetter';'  
|   'create object instance of 'keyLetter';'  
|   'relate 'instanceHandle' to 'instanceHandle' across  
'relationshipSpecification';'  
|   'unrelate 'instanceHandle' from 'instanceHandle' across  
'relationshipSpecification';'  
|   'select any 'instanceHandle' from instances of 'keyLetter (' where '  
whereExpression)?';'  
|   'select many 'instanceHandle' from instances of 'keyLetter (' where '  
whereExpression)?';'  
|   'select any 'instanceHandle' related by 'start'->'className  
relationshipLink ('->'className relationshipLink)* (' where ' whereExpression)?';'  
|   'select many 'instanceHandle' related by 'start'->'className  
relationshipLink ('->'className relationshipLink)* (' where ' whereExpression)?';'  
|   'delete object instance 'instanceHandle';'  
|   variableName'='expr';'
```

```
|   instanceHandle'.atribute'='expr';'  
|   'assign 'variableName'='expr';'  
|   'assign 'instanceHandle'.atribute'='expr';'  
|       'call from 'keyLetter::'methodName() to 'keyLetter::'methodName()'('across 'relationshipSpecification)?';'  
;  
;
```

relationshipLink

```
:      '['RelationshipSpecification']'  
;  
;
```

instanceHandle

```
:      VariableName  
;  
;
```

keyLetter

```
:      VariableName  
;  
;
```

whereExpression

```
:      expr  
;  
;
```

start

```
:      VariableName  
;  
;
```

className

: *VariableName*
;
;

variableName

: *VariableName*
;
;

methodName

: *VariableName*
;
;

anyOrMany

: *AnyOrMany*
;
;

attribute

: *VariableName*
;
;

expr

: *Digit* | *VariableName* | *Text*
| *VariableName'.'VariableName*
| 'i' *cardinality* 'VariableName'
| *expr* ('*' | '/' | '%') *expr*
| *expr* ('+' | '-') *expr*

```

|      expr ('<' | '>' | '<=' | '>=') expr
|      ('empty' | 'not_empty') VariableName
|      ('NOT' | 'not') expr
|      expr ('==' | '!=') expr
|      ('( expr )')
|      expr ('AND' | 'OR' | 'and' | 'or') expr
;

```

relationshipSpecification

```

:      RelationshipSpecification
;

```

AnyOrMany

```

:      ('any' | 'many')
;

```

RelationshipSpecification

```

:      'R'Digit
;

```

VariableName

```

:      Nondigit ( Nondigit | Digit )*
;

```

Text

```

:      """( Nondigit | Digit | ' ')**"""

```

;

Digit

: [0-9]+(.'[0-9]+)?

;

Nondigit

: [a-zA-Z_#]

;

Whitespace

: [\t]+

-> skip

;

NewLine

: ('r'? '\n' | '\r')+

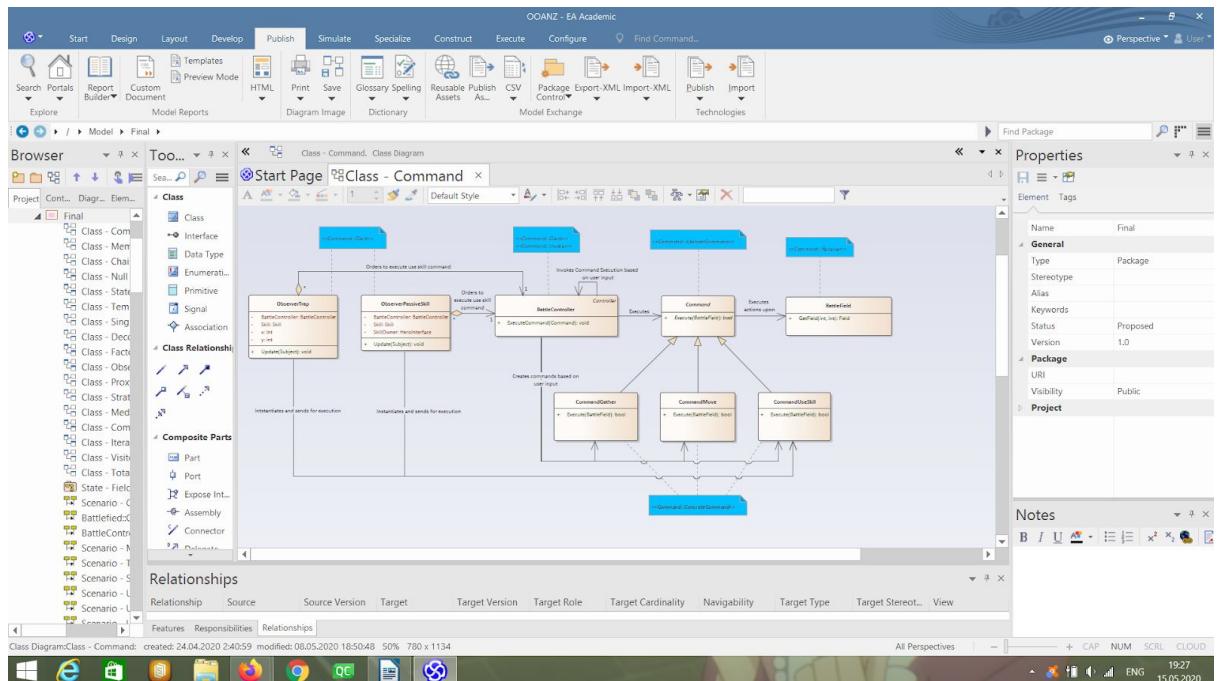
-> skip

;

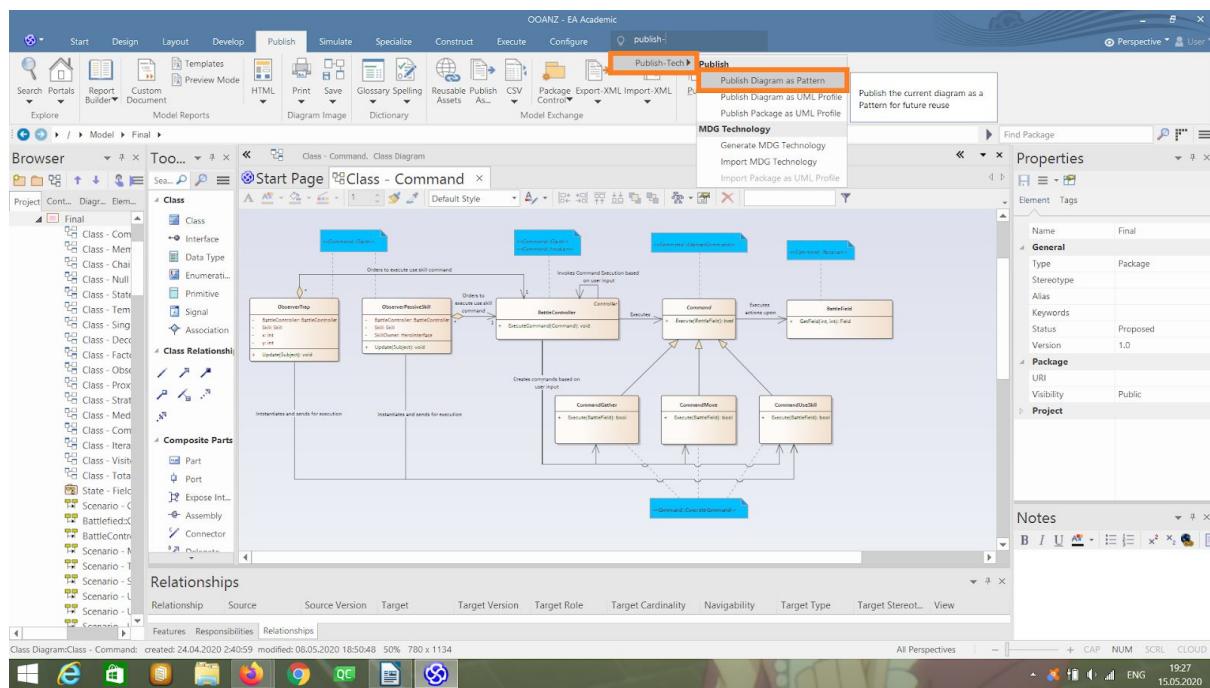
Príloha E - Podklady pre katalóg vzorov

Postup pri ukladaní vzorov v Enterprise Architect

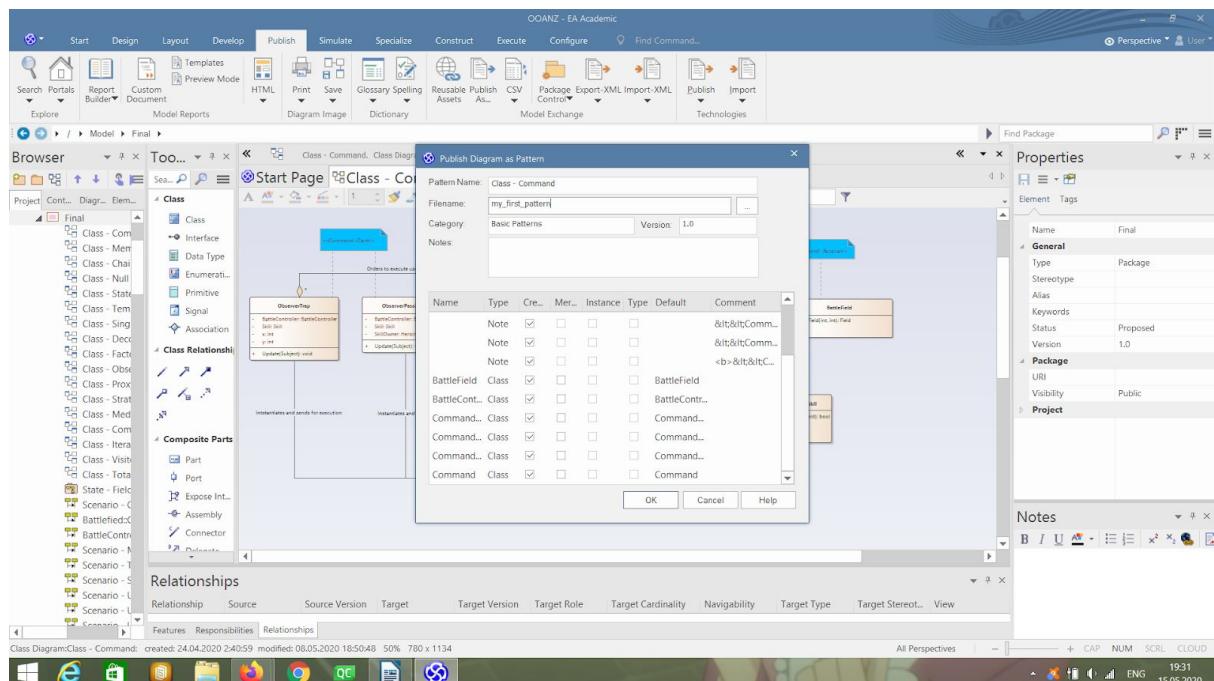
- V Enterprise Architect vytvorte alebo otvorte hotový diagram.



- Ked' je diagram otvorený, vyberte si v menu Publish-tech → Publish Diagram As Pattern



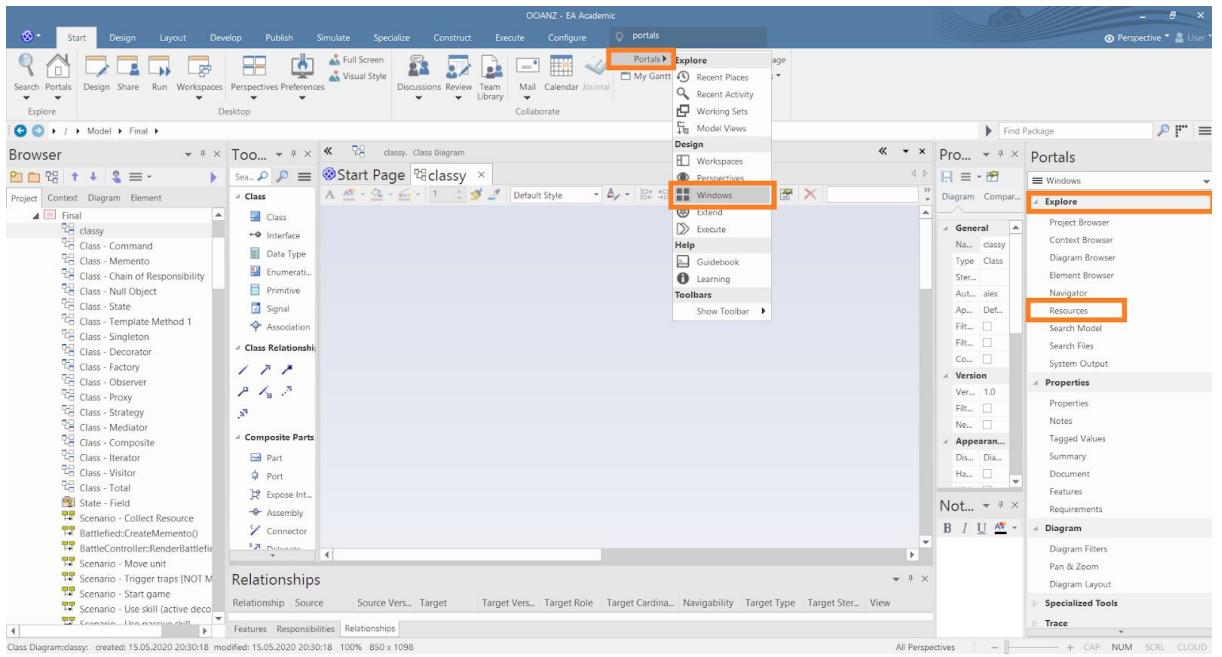
3. V otvorenom okienku “Publish Diagram As Pattern” si zadajte meno vzoru, meno súboru, kategóriu v ktorej bude vzor uložený, a jeho verziu. Potom si vyberte všetky elementy z diagramu, ktoré budú prítomné vo vzore.



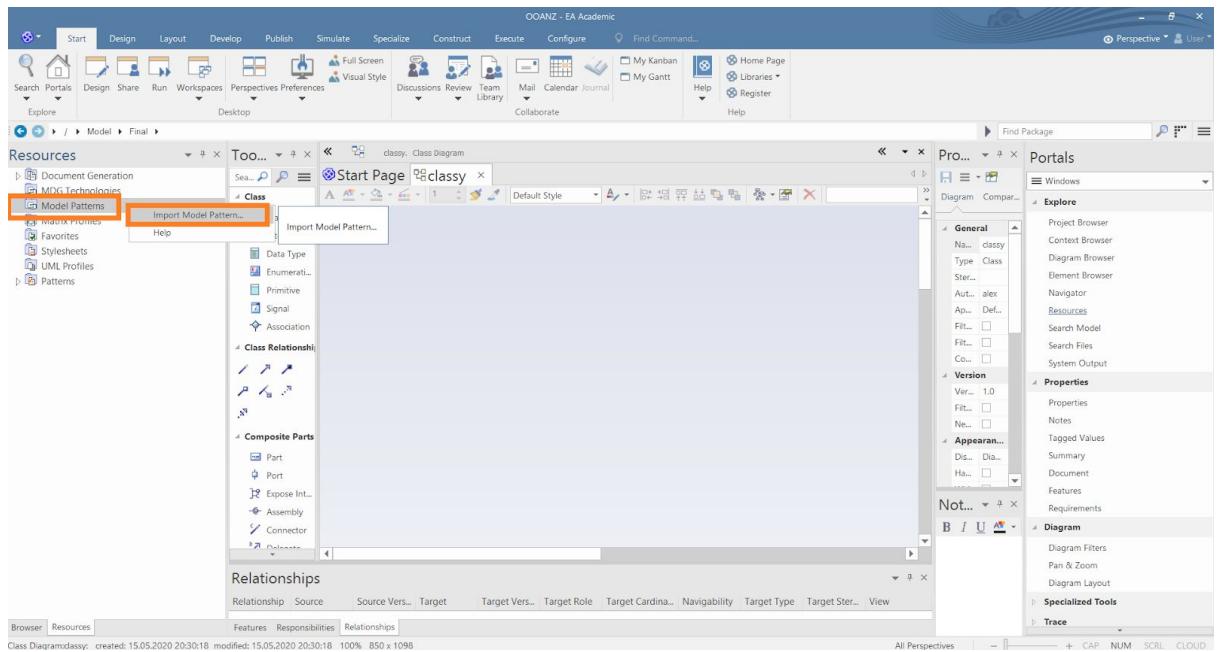
S týmto sa vytvorí pattern vo vybranom súbore.

Postup pri využití vzorov v Enterprise Architect

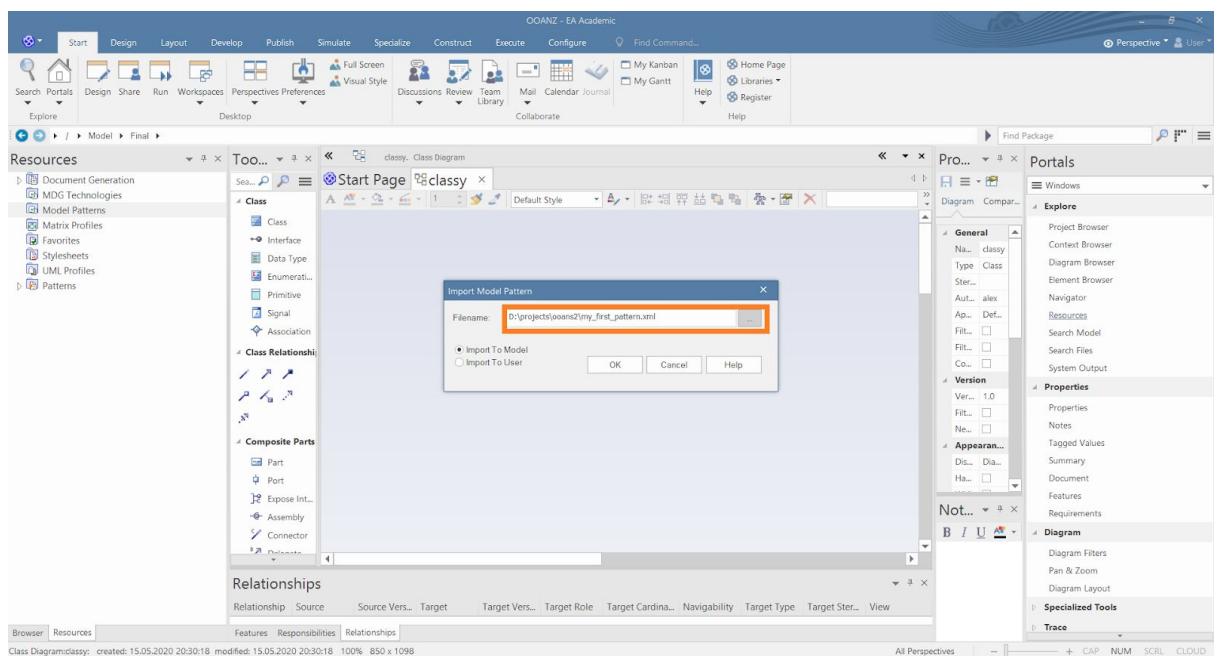
1. V Enterprise Architect si otvorte okná pomocou Portals → Windows. V okne Portals si vyberte možnosť Explore → Resources

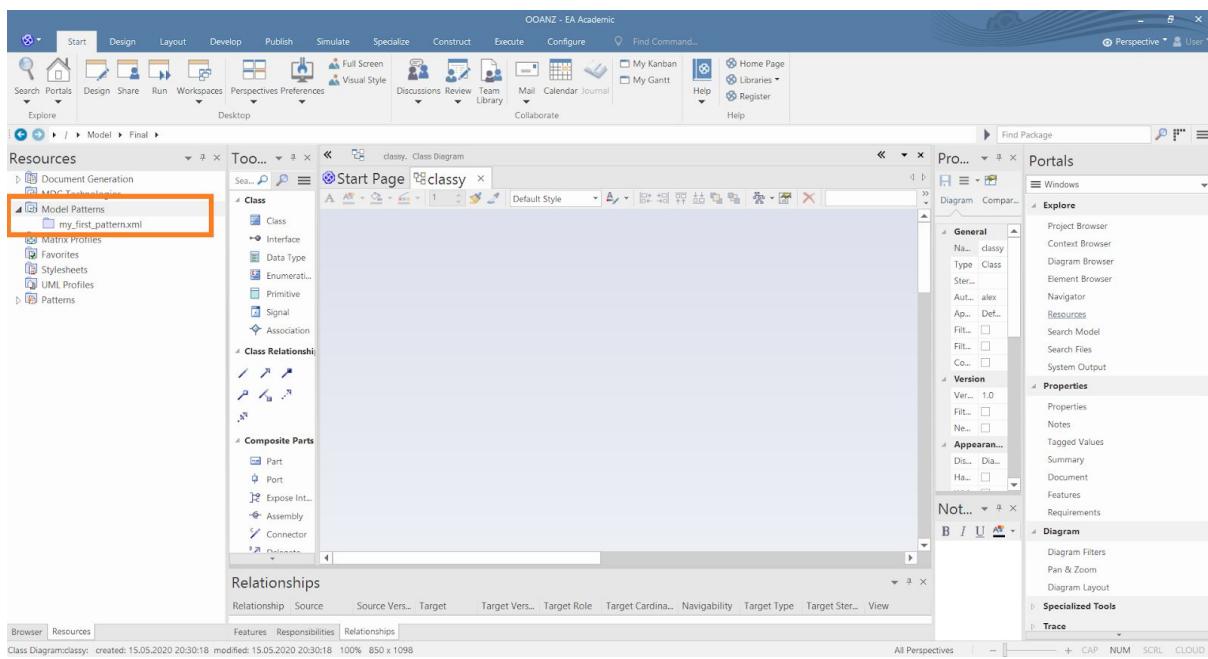


2. V otvorenom okne “Resources” otvorte pravým tlačidlom nad “Model Patterns” kontextové menu a vyberte si možnosť “Import Model Pattern”



3. Vyberte meno vzoru na importovanie.





4. Vzor bol importovaný do kategórie “Model Patterns” ako .xml súbor.

Pridajte ho do priečinku “Patterns” a môžete ho dať do otvoreného diagramu pomocou kliknutia pravým tlačidlom a výberu “Add Pattern to Diagram”, alebo prosté využitím Drag'n'Drop.

