VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

PRÍPADOVÁ ŠTÚDIA ZROVNÁVAJÚCA MODELOM RIADENÉ PRÍSTUPY K TVORBE SOFTWARU

CASE STUDY COMPARING MODEL-DRIVEN APPROACHES TO SOFTWARE DESIGN

BAKALÁRSKA PRÁCA
BACHELOR'S THESIS

AUTOR PRÁCE — **LUKÁŠ OSADSKÝ**
AUTHOR

VEDÚCI PRÁCE — **Ing. RADEK KOČÍ, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Osadský Lukáš**
Program: Informační technologie
Název: **Případová studie srovnávající modelem řízené přístupy k tvorbě softwaru**
Case Study Comparing Model-Driven Approaches to Software Design
Kategorie: Softwarové inženýrství

Zadání:

1. Seznamte se s přístupem k tvorbě softwarových systémů nazývaný *Model Driven Engineering* (MDE).
2. Prostudujte možnosti tvorby softwarových systémů a po dohodě s vedoucím práce se zaměřte na vybrané techniky MDE.
3. Navrhněte kritéria pro vyhodnocení jednotlivých technik návrhu.
4. Demonstrujte návrh a realizaci softwarových systémů s využitím vybraných technik MDE na případové studii konferenčního recenzního systému.
5. Srovnejte zkoumané techniky podle předem definovaných kritérií.

Literatura:

- T. Buchmann, A. Rimer. Unifying Modeling and Programming with ALF. SOFTENG 2016: The Second International Conference on Advances and Trends in Software Engineering.
- J. Bezivin. Model Driven Engineering: An Emerging Technical Space. Generative and Transformational Techniques in Software Engineering, International Summer School, GTTSE 2005, Braga, Portugal, 2005.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Kočí Radek, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

Abstrakt

Táto bakalárska práca sa zaoberá problematikou vývoja softwaru za pomoci rôznych prístupov k návrhu a implementácii softwaru. Hlavným cieľom práce je na základe navrhnutých kritérií porovnať klasický prístup k vývoju softwaru s alternatívnym prístupom založeným na transformácii abstraktných modelov, zvaným Model-Driven Engineering (MDE). Tieto dva prístupy sú následne demonštrované na prípadovej štúdii jednoduchého konferenčného recenzného systému. V závere práce sú diskutované možnosti a obmedzenia techník MDE.

Abstract

This bachelor thesis deals with various approaches to software design and its implementation. The main goal of this thesis is to compare the classical approach of software development to an alternate approach called Model-Driven Engineering (MDE), which is based on abstract models transformations. These two approaches are then demonstrated on a case study of simple conference review system. At the end of the work, the possibilities and limitations of MDE techniques are discussed.

Kľúčové slová

Model-Driven Engineering, ALF, konferenčný recenzný systém, prípadová štúdia

Keywords

Model-Driven Engineering, ALF, conference review system, case study

Citácia

OSADSKÝ, Lukáš. *Prípadová štúdia zrovnávajúca modelom riadené prístupy k tvorbe softwaru*. Brno, 2020. Bakalárska práca. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedúci práce Ing. Radek Kočí, Ph.D.

Prípadová štúdia zrovnávajúca modelom riadené prístupy k tvorbe softwaru

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Radka Kočího, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Lukáš Osadský
17. júna 2020

Poďakovanie

Chcem poďakovať vedúcemu mojej bakalárskej práce pánovi Ing. Radkovi Kočímu, Ph.D za vedenie a odbornú pomoc pri vypracovávaní tejto práce.

Obsah

Kapitola 1

Úvod

V dnešnej dobe sa so softwarom stretávame pri rôznych činnostiach a sprevádza náš každodenný život. Rok čo rok rastú požiadavky na software a s nimi aj jeho funkcionalita a komplexita. Okrem bežného softwaru sa vyvíjajú aj platformy, nad ktorými tento software operuje. Z tohoto dôvodu je nutné software neustále upravovať a prispôsobovať ho platformám, na ktorých má operovať, prípadne je nutné vyvíjať nový software pre nové platformy. Neustály vývoj softwaru je celkovo časovo a finančne náročný proces a preto v posledných desaťročiach vznikla nová myšlienka prístupu k vývoju softwaru zvaná Model Driven Engineering (MDE). Tento prístup pracuje s myšlienkou využitia abstraktných modelov pre potreby implementácie softwaru. Prístup MDE je však skôr teoretický a nie je príliš vhodný na využitie na vývoj softwaru. Na základe MDE, organizácia Object Management Group (OMG) vytvorila štandard zvaný Model Driven Architecture (MDA), ktorý priamo využíva modely, ako artefakty, na tvorbu spustiteľného softwaru. Jedná sa o konkrétnu realizáciu MDE za využitia technológií, pomocou ktorých je možné vytvoriť z UML modelov funkčný software. MDA využíva modely rôznych úrovní abstrakcie na popis softwaru od biznis požiadaviek až po jeho implementačné detaily. Myšlienkou MDA je generovať funkčný kód na základe UML modelov. Grafická reprezentácia UML modelov však nemusí stačiť na zachytenie všetkých implementačných detailov a preto organizácia OMG vytvorila jazyk Action Language for foundation UML (ALF), ktorý slúži na textuálny popis správania týchto modelov a za pomoci tohoto jazyka je možné vytvárať spustiteľné programy.

Cieľom tejto práce je porovnať dva prístupy vývoja software, konkrétne tradičné programovanie s prístupom MDA za pomoci jazyka ALF. Oba prístupy demonštrovať na príklade konferenčného recenzného systému a tieto systémy na základe predom definovaných kritérií porovnať. Postupným štúdiom materiálov, testovaním dostupných technológií a komunikáciou s viacerými stranami, bolo zistené, že jazyk ALF môže slúžiť na vývoj jednoduchšieho softwaru. Čo sa týka vývoja komplexnejších systémov, jazyk ALF zaostáva v porovnaní s bežnými programovacími jazykmi z dôvodu, že jeho vysoká miera abstrakcie neposkytuje dostatočnú funkcionalitu a taktiež pre jeho potreby nie je voľne dostupná sada nástrojov, ktorá by to plne umožňovala. Vzhľadom na to, že systém nebolo možné implementovať za pomoci jazyka ALF, táto práca popisuje implementáciu systému za použitia programovania, skúma možnosti jazyka ALF a porovnáva tieto dva prístupy k vývoju softwaru.

Práca je členená do niekoľkých kapitol, v ktorých je popísaná teória týkajúca sa vývoja softwaru a celkový postup práce. Kapitola 2 je venovaná softwarovému inžinierstvu. Je v nej popísaný životný cyklus softwaru, jeho životné modely, metodiky a prístupy využívajúce sa pri jeho implementácii. V kapitole 3 je popísaná prípadová štúdia, demonštrovaná na príklade konferenčného systému. Kladie sa dôraz na funkčné a systémové požiadavky,

požiadavky na role a technológie vhodné na využitie k vývoju daného systému. V kapitole 4 je popísaný návrh, postup práce a konkrétna implementácia systému za použitia zvolených technológií. Nakoniec v kapitole 5 sú definované kritériá na základe ktorých sú zvolené prístupy k vývoju softwaru porovnané.

Kapitola 2

Softwarové inžinierstvo

Softwarové inžinierstvo je systematický prístup k vývoju softwaru, jeho nasadeniu a údržbe. Jedná sa o podoblasť systémového inžinierstva, ktorá zasahuje do viacerých oblastí ako počítačové a manažérske vedy. Celý proces vývoja softwaru sa nazýva životný cyklus softwaru a skladá sa z niekoľkých krokov, ktoré na seba priamo nadväzujú. Jedná sa o zber požiadaviek od zákazníka na nový produkt. Ďalej návrh softwaru, ktorý vzniká zo zozbieraných požiadaviek od zákazníka. Na základe návrhu sa začína vývoj softwaru, ktorý sa súbežne testuje. Po otestovaní je software nasadený do produkčného prostredia. Poslednou fázou je údržba nasadeného softwaru [29].

2.1 Životný cyklus softwaru

Životný cyklus softwaru je systematický prístup k tvorbe softwaru. Tento prístup definuje jednotlivé kroky týkajúce sa tvorby softwaru od plánovnia a zbierania požiadaviek, cez návrh a implementáciu až po nasadene a údržbu softwaru. Celý proces pozostáva z niekoľkých častí, kde každá časť popisuje jednotlivú etapu životného cyklu softwaru. V tejto kapitole budú popísané jednotlivé etapy a prístupy k vývoju softwaru.

2.1.1 Zber požiadaviek

Jedná sa o činnosť, ktorej hlavnou úlohou je zistiť požiadavky na software od zainteresovaných strán, stakeholderov. Môžu na to byť použité rôzne metódy, ako napríklad JAD[1] sedenia. Jedná sa o proces, ktorého cieľom je zapojenie klienta, prípadne koncového používateľa do vývoja produktu, z dôvodu lepšieho zozbierania biznis požiadaviek a prispieva k vyššej spokojnosti zákazníka, keďže je priamo zapojený do vývoja. Ďalej sa vykonáva analýza existujúcich dokumentov, zadefinovanie si účelu produktu, jeho limitácie a podobne. Po zozbieraní všetkých požiadaviek nasleduje ich analýza. Cieľom tejto činnosti je podrobný rozbor požiadaviek, možnosti ich naplnenia a rozsahu, v akom je možné dané požiadavky naplniť. Tretím krokom je vyhotovenie špecifikácie pre potrebu uchovania získaných znalostí, slúžiacej na efektívne predanie informácie. To zahŕňa zhotovenie prípadov užitia, definovanie userstories, funkčných požiadaviek na produkt a podobne. Posledným krokom je validácia, ktorá potvrdzuje, či bola splnená množina požiadaviek na vytvorenie riešenia, ktoré spĺňa biznis požiadavky. Výstupom môže byť Software Requirement Specification (SRS) dokument, ktorý obsahuje súpis všetkých požiadaviek.

[1] http://www.umsl.edu/ sauterv/analysis/JAD.html

2.1.2 Softwarový návrh

Jedná sa o proces, v ktorom sú transformované užívateľské požiadavky zo SRS do vhodnej formy pre potreby vývoja softwaru. Softwarový návrh je fáza životného cyklu softwaru, ktorá špecifikuje to, ako naplniť požiadavky zo SRS. Úlohou softwarového návrhu je popísať dátové modely, štruktúru systému, rozhranie medzi komponentami systému, prípadne algoritmy pre potreby implementácie. Softwarový návrh sa dá rozdeliť do troch úrovní. Návrh najvyššej úrovne abstrakcie sa nazýva Architektonický návrh. Definuje software ako systém s niekoľkými komponentami, ktoré medzi sebou komunikujú. Cieľom tejto abstrakcie je pochopiť myšlienku navrhovaného riešenia. Ďalším krokom je vytvoriť návrh, ktorý popisuje daný software na nižšej úrovni abstrakcie. Popisuje konkrétne komponenty softwaru, určuje rozhrania, cez ktoré spolu komunikujú a vzťahy medzi nimi. Detailný návrh, sa zaoberá implementačnými detailami, definuje logickú štruktúru každého modulu, z ktorého sa software skladá a detailne popisuje, ako rozhrania medzi sebou komunikujú. Výstupom návrhu môže byť dokumentácia, pseudo kódy, diagramy popisujúce štruktúru a správanie softwaru a detailné popisy funkčných a ostatných požiadaviek. Potom, čo je návrh produktu hotový, prebieha verifikácia dizajnu. Cieľom verifikácie je odhaliť potenciálne chyby, ktoré by v neskorších fázach životného cyklu softwaru mohli spôsobovať problémy.

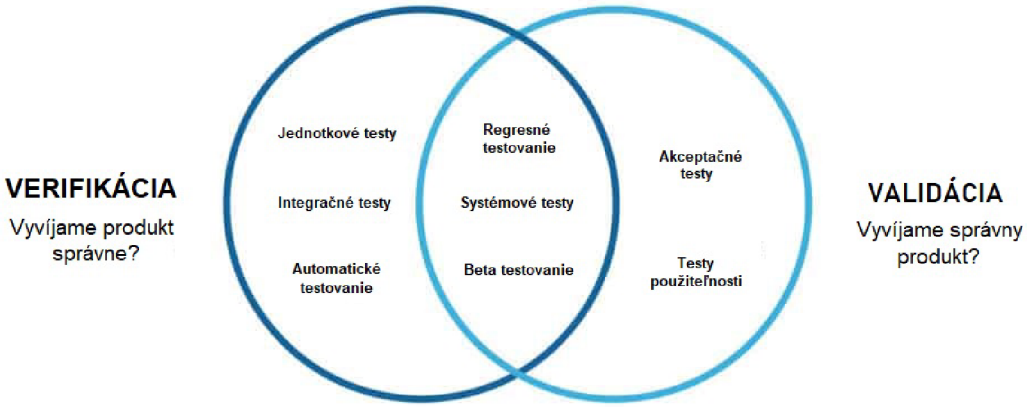
2.1.3 Vývoj softwaru

Potom, čo je návrh hotový, prichádza na rad vývoj softwaru. Je to iteratívny logický proces, ktorého cieľom je vytvoriť funkčný počítačový kód schopný vykonávať požadované operácie. Implementácia softwaru je transformácia jednotlivých komponentov systému a ich väzieb do programovej realizácie. Implementáciu môžeme vykonať viacerými spôsobmi, napríklad progamovaním alebo postupnou transformáciou modelov na iné modely, tento postup bude bližšie popísaný v sekcii Modelovanie 2.3. Súbežne s vývojom softwaru prebieha aj jeho testovanie [17]. Testovanie zahŕňa činnosti, ako validácia a verifikácia. Jedná sa o proces evaluácie vyvíjaného softwaru voči požiadavkam plynúcim zo SRS. Podstatou validácie je určiť, či vyvíjaný software spĺňa užívateľské požiadavky, inak povedané "vyvíjame správny produkt". Zahŕňa to činnosti ako akceptačné testovanie produktu alebo testy použiteľnosti. Verifikácia je evaluácia produktu v určitej fáze vývoja s cieľom určiť, či vyvíjaný produkt spĺňa funkčné požiadavky. Pýtame sa otázku "vyvíjame produkt správne". Táto fáza zahŕňa písane rôznych scenárov testov (angl. test case), za účelom zistenia, či daný produkt funguje správne [27]. Vzťah validácie a verifikácie možno vidieť na obrázku 2.1.

2.1.4 Nasadenie a údržba softwaru

Poslednou fázou vývojového cyklu softwaru je jeho nasadenie a následná údržba. Nasadenie softwaru zahŕňa niekoľko aktivít. Každý systém je unikátny, čo znamená, že dané aktivity môžu prebiehať na strane zákazníka aj na strane producenta. Tento proces zahŕňa aktivity ako Release softwaru - vydanie finálnej verzie produktu, ďalej inštalácia a aktivácia produktu, rôzne vydania updatov a iné. Poslednou fázou je údržba nasadeného systému. Do tejto fázy spadajú všetky modifikácie po nasadení softwaru a taktiež zákaznícka podpora. Údržba systému je najdrahšia z celého životného cyklu software. [7]

[2] https://www.plutora.com/blog/verification-vs-validation



Obr. 2.1: Typy testov a ich vzťah k verifikácii a validácii[2]

2.2 Modely životného cyklu softwaru

V predchádzajúcej sekcii sme boli oboznámení so životným cyklom softwaru. V tejto kapitole budú popísané niektoré metodiky slúžiace na vývoj produktov. Ako sa softwarové inžinierstvo v priebehu času vyvíjalo, vyvíjal sa aj prístup vývoja softwaru. Vznikli rôzne druhy metodík, ktoré slúžili ako súhrn spôsobov na vývoj softwaru, každá táto metodika popisuje jednotlivé etapy vývoja softwaru.

2.2.1 Vodopádový model

Vodopádový model je prvý typ modelu životného cyklu softwaru, ktorý bol predstavený. V tomto modeli sú jednotlivé etapy radené za sebou, pred začatím ďalšej etapy je nutné dokončiť predchádzajúcu etapu. Vodopádový model sa skladá z piatich etáp.

- Zber požiadaviek a ich analýza
- Systémový návrh na základe požiadaviek
- Implementácia návrhu a unit testing
- Celkové testovanie produktu
- Prevádzka a údržba

Tento model je jednoduchý na porozumenie a použite, jeho fázy sa dajú jednoducho spravovať a je dobre použiteľný na menšie projekty. Problémom pri použití tohoto modelu je vysoká miera risku pri vývoji. Užívateľ uvidí výsledný produkt až na konci, v prípade, že na začiatku boli nepresne definované požiadavky, alebo zákazník prišiel s novým typom požiadavkov, je nutné celý proces začať od znova, čo sa odzrkadlí na čase a zdrojoch. V praxi je možné použiť tento model, ale uprednostňované sú iné typy modelov. [17].

2.2.2 Iteratívny model

Tento model vychádza z vodopádového modelu a celá myšlienka spočíva v tom, že miesto dodania celého produktu naraz sa dodávajú jeho časti. Každou iteráciou, čo sa dá chápať ako inštancia vodopádového modelu, sa užívateľovi poskytne časť softwaru a získa sa od neho spätná väzba. Prináša to výhody ako napríklad rýchlejšie vyvíjanie častí projektu, jednoduchšie testovanie, zmenšenie miery rizika pri použití modelu na väčšie projekty a iné. Nevýhodou iteratívneho modelu je potreba vyššej réžie manažmentu. Aj napriek faktu, že v prípade zmeny požiadaviek sú menšie náklady na realizáciu, tento model nie je vhodný na ich zmenu a ani na použitie pri malých projektoch [17].

2.2.3 Špirálový model

Špirálový model spája myšlienku iteratívneho modelu so systematicky kontrolovanými aspektami vodopádového modelu s vysokým dôrazom na analýzu rizík a umožňuje inkrementálne vydávať prototypy produktu. Tento model pozostáva z cyklov a každý cyklus zo štyroch fáz.

- Špecifikácia problému - fáza zbierania požiadaviek
- Návrh - zahŕňa plánovanie, architektonický, detailný návrh
- Implementácia a testovanie
- Vyhodnotenie a analýza rizík - zahŕňa estimácie, užívateľské ohodnotenie a poskyuje spätnú väzbu

Od prvej iterácie je k dispozícii prototyp softwaru, ktorý overuje správne pochopenie požiadaviek, vlastností a podobne. Tým pádom sú skôr odhalené chyby. Nevýhodou je, že tento model je náročný na riadenie. [17]

2.2.4 Rational Unified Process

Metodika Rational Unified Process (RUP) je použiteľný model pre riadenia softwarových projektov. Jedná sa o iteratívnu metodiku, ktorá je generická, to znamená, že je možné prispôsobiť ju na konkrétnu firmu a projekt. Na konci každej iterácie tejto metodiky je spustiteľný program. Vznikla zo súboru osvedčených postupov a praktík pri vývoji softwaru. Životný cyklus pozostáva zo štyroch fáz a to inception, elaboration, construction a transition. Jej výhodou je, že je použiteľná na projekty akéhokoľvek rozsahu, pričom sa kladie dôraz hlavne na analýzu, návrh, plánovanie a dokumentáciu [17].

2.2.5 Agilné metodiky

Sú kombináciou iteratívneho a inkrementálneho procesu. Hlavnou výhodou tejto metodiky je schopnosť reagovať na zmeny a dodávať funkčný software po častiach. Každá iterácia trvá zvyčajne dva týždne a zahŕňa v sebe aktivity ako plánovanie, analýza požiadaviek, návrh, implementácia požiadaviek, ich funkčné testovanie a nakoniec akceptačné testovanie. Na konci každého cyklu je implementovaný produkt demonštrovaný zákazníkovi. Výhodou je včasné odhalenie chýb, skorá užívateľská spätná odozva a možnosť rýchlo reagovať na zmeny. Tieto metodiky sú ťažsie na porozumenie, to so sebou prináša väčšiu réžiu riadenia a je potrebný celkový plán vývoja. Medzi tieto metodiky patrí napríklad SCRUM, Extreme Programming, Crystal metodiky a iné. [6]

2.3 Modelovanie

Nasledujúca sekcia je venovaná modelovaniu, rôznym prístupom a prostriedkom, ktoré sa tejto činnosti týkajú. V predchádzajúcej sekcii sme si povedali niečo o životnom cykle softwaru a o jednotlivých metodikách. Tieto metodiky sú v podstate modely, ktoré popisujú rôzne prístupy k vývoju softwaru. Modely sú základom modelovania a používajú sa vo všetkých častiach vývojového cyklu softwaru. Využívajú sa pre potreby špecifikácie, dokumentácie, návrhu a podobne. Nemecký filizof Herbert Stachowiak ich popísal na základe troch vlastností. [30]

- Mapovanie - Model je vždy modelom originálu. Originál môže byť niečo reálne alebo imaginárne.
- Redukcia - Modely vo všeobecnosti nezachytávajú všetky atribúty originálu ale iba tie, ktoré sú relevantné pre účely modelovania.
- Pragmatizmus - Modely nie sú priradené ku svojim originálom vo vzťahu jedna k jednej, plnia vždy funkciu nahradenia a aby spĺňali svoj účel musia byť použiteľné namiesto originálu.

Inak povedané, jedná sa o určitú formu abstrakcie, ktorá popisuje stav, vlastnosti, dynamiku alebo štruktúru modelovaného objektu, systému prípadne ich časti. Entitu, ktorú modelujeme môžeme popísať viacerými modelmi. Jedným modelom je možné popísať jej štruktúru, iným vlastnosti, prípadne správanie v rôznych mierach abstrakcie. V softwarom inžinierstve sa využívajú modely napríklad pri špecifikáciách požiadaviek od zákazníka, kde je účelom zachytiť vlastnosti požadovaného systému, ktoré sú využité ako predloha na vytvorenie systému. V iných prípadoch môžu modely slúžiť ako hlavné artefakty pri implementácii systémov.

Pri modelovaní sa môžeme stretnúť s množstvom termínov s ním súvisiacich [5]. Model-Based Engineering (MBE) alebo Model-Driven Engineering (MDE) je softwarové paradigma, ktoré uplatňuje princípy vizuálneho modelovania počas životného cyklu vývoja softwaru. Jedná sa o zastrešujúci výraz pre disciplíny ako Model-Driven Development (MDD), Model-Based System Engineering (MBSE), Business Process Modeling (BPM) a Ontology Engineering. MDD je poddisciplína zaoberajúca sa aplikáciou modelovo riadených technológií na aktivity týkajúce sa vývoja softwaru. Ďalšou disciplínou zaoberajúcou sa vývojom softwaru je Model-Driven Architecture (MDA), jedná sa o konkrétnu implementáciu MDD. MBSE je disciplína systémového inžinierstva, ktorej hlavným cieľom je využívať modely na výmenu informácií medzi inžiniermi. BPM slúži na popis podnikových procesov pomocou diagramov. Ontology Engineering je disciplína špecializovaná na vytváranie ontológií. Bližšie sa táto práca bude zaoberať disciplínou MDD a MDA v sekcii 2.6.

2.4 Programovanie

Tak ako ľudia, aj počítače používajú rôzne jazyky na spôsob komunikácie. V našom prípade sa jedná o angličtinu, nemčinu, posunkovú reč a iné, ktorých cieľom je výmena informacií. Majú určitú syntaktickú skladbu a sémantický význam. To isté platí aj pre počítače. Tak ako modely, aj programovacie jazyky sa podieľajú na tvorbe softwaru vo viacerých fázach životného cyklu software. Tvorba softwaru za pomoci jazyka sa nazýva programovanie. Jedná

sa o proces návrhu a tvorenia spustiteľného programu, ktorého účelom je konkrétna výpočetná operácia. Zahŕňa v sebe činnosti ako návrh algoritmov, písanie a ladenie zdrojového kódu, taktiež jeho údržbu a testovanie. Zjednodušene povedané, programovanie je proces, v ktorom programátor dáva počítaču inštrukcie, ktorými mu hovorí, že má vykonať nejakú operáciu. Sú to operácie, ako napríklad ulož hodnotu, zaokrúhli číslo, vyhodnoť výraz a podobne. Tieto inštrukcie sú zapísané v špecifickej syntakticej forme, programovacom jazyku, ktorej počítač rozumie, tak ako človek rozumie hovorenej reči. Existuje množstvo programovacích jazykov, ktoré sa používajú ako prostriedok pre jednoznačné zapísanie inštrukcií vykonávaných na počítači. Pomocou nich je možné implementovať rôzne typy softwaru. K tomuto využívajú rôzne prístupy a fungujú na iných úrovniach abstrakcie. Všeobecne je možné programovacie jazyky podľa abstrakcie rozdeliť do dvoch skupín a to nízkoúrovňové a vysokoúrovňové. Príkladom nízkoúrovňového jazyka môže byť Assembler (výpis 2.1). Ide o jazyk, ktorý slúži na písanie symbolických inštrukcií pre určitý procesor. Do vysokoúrovňových jazykov patrí napríklad jazyk C (výpis 2.2), ktorý nemusí byť použitý len na určitom type procesoru, ale je možné ho zkompilovať a využiť takmer pre každý typ počítača. Jedná sa o štrukturovaný jazyk, ktorý je možné deliť do blokov, z dôvodu lepšej štrukturalizácie kódu. Do kategórie vysokoúrovňových jazykov patria aj objektovo orientované jazyky ako napríklad Java (výpis 2.3), kde úroveň abstrakcie je ešte vyššia ako v prípade štruktorovaných jazykov. Tento prístup je založený na využívaní dátových štruktúr nazývaných objekty. Objekty majú zadefinované svoje vlastnosti a správanie, na základe ktorého komunikujú medzi sebou. Vyššie spomenuté jazyky sa podľa spôsobu prekladu nazývajú kompilované. Tieto jazyky sú pred spustením najskôr preložené kompilátorom do strojového kódu a až potom je možné ich spustiť. Ďalšou skupinou sú interpretované jazyky. Tieto jazyky sa vykonávajú za behu programu a nie je nutné ich prekladať do strojového kódu. Do tejto skupiny patria skriptovacie jazyky ako Javascript (výpis 2.4). Vyššie spomenuté jazyky sa nazývajú General-purpose languages (GPL) [4]. Sú to jazyky so všeobecným použitím a je možné ich použiť naprieč doménami. Druhou skupinou sú Domain-specific languages (DSL), ktoré sú optimalizované na použitie v určitej doméne a ponúkajúce vyššiu mieru abstrakcie na riešenie špecifického problému [3]. Príkladom môže byť jazyk SQL (výpis 2.5), ktorý slúži na manipuláciu a definíciu dát v databáze.

Výpisy nižšie demonštrujú zápis funkcie na výpočet N-tej mocniny čísla X v jazykoch Assembler, C, Java a JavaScript. Výpis pre jazyk SQL zobrazuje dotaz na získanie dát z databáze.

```
pow(int, int):
        push rbp
        mov rbp, rsp
        mov DWORD PTR [rbp-20], edi
        mov DWORD PTR [rbp-24], esi
        mov DWORD PTR [rbp-4], 1
        mov DWORD PTR [rbp-8], 0
.L3:
        mov eax, DWORD PTR [rbp-8]
        cmp eax, DWORD PTR [rbp-24]
        jge .L2
        mov eax, DWORD PTR [rbp-4]
        imul eax, DWORD PTR [rbp-20]
        mov DWORD PTR [rbp-4], eax
```

```
        add DWORD PTR [rbp-8], 1
        jmp .L3
.L2:
        mov eax, DWORD PTR [rbp-4]
        pop rbp
        ret
```

Výpis 2.1: Príklad zdrojového kódu v jazyku Assembler

```
int pow(int x, int n)
{
    int result = 1;
    for (int i = 0; i < n; i++) {
        result *= x;
    }
    return result;
}
```

Výpis 2.2: Príklad zdrojového kódu v jazyku C

```
public class SomeClass
{
    public int pow(int x, int n)
    {
        int result = 1;
        for (int i = 0; i < n; i++) {
            result *= x;
        }
        return result;
    }
}
```

Výpis 2.3: Príklad zdrojového kódu v jazyku Java

```
function pow(x, n) {
    let result = 1;
    for (let i = 0; i < n; i++) {
        result *= x;
    }
    return result;
}
```

Výpis 2.4: Príklad zdrojového kódu v jazyku Javascript

```
    SELECT Customer.id, Customer.name, Product.name, Product.license
    FROM Customer
    JOIN Product ON Product.customerId = Customer.id
    WHERE Product.name = 'Product'
```

Výpis 2.5: Príklad zdrojového kódu v jazyku SQL

2.5 Unified Modeling Language

Unified Modeling Language (UML) je grafický jazyk štandardizovaný organizáciou Object Management Group (OMG), slúžiaci na konštrukciu, vizualizáciu, popis správania a dokumentáciu systémových artefaktov. UML má široké uplatnenie. Nejedná sa o programovací jazyk, ale skôr o modelovací jazyk, ktorého cieľom je vizualizovať systémy v takej podobe, ako sú nadizajnované. Je možné ho použiť na tvorbu informačných systémov, bankových a finančných servís, distribuovaných systémov, v telekomunikáciách a iných doménach. UML je používané na vyobrazenie správania a štrukúry systému, je ním možné vizualizovať systémovú architektúru a jej elementy, ako sú komponenty systému, rôzne aktivity, spôsob akým medzi sebou komunikujú, užívateľské rozhrania a iné. UML poskytuje sady diagramov na vizualizáciu systému, tieto diagramy sa skladajú z rôznych entít a prepojení medzi nimi. UML pozostavá z troch stavebných blokov, ktoré sú popísane ako Things (predmety), Relationships (relácie) a Diagrams (diagramy). [10] [8]

2.5.1 Predmety

Predmety sú dôležité bloky UML, jedná sa o abstrakciu modelov. Dajú sa rozdeliť na štrukturálne, behaviorálne, zoskupovacie a anotačné.[8]

Štrukturálne predmety

Štrukturálne predmety v UML definujú statickú časť modelu, patria sem triedy, rozhrania, kolaborácie, prípady užitia, komponenty a uzly.

- Triedy - reprezentujú množinu objektov, ktoré majú rovnaké vlastnosti
- Rozhrania - definujú množinu operácií, ktoré definujú správanie odvodených tried
- Kolaborácie - popisujú interakciu medzi jednotlivými elementami
- Prípady užitia - reprezentujú množinu užití, ktoré sa môžu v systéme vykonaváť
- Komponenty - popisujú konkrétne časti systému
- Uzly - definujú fyzické elementy, ktoré existujú za behu systému

Behaviorálne predmety

Behaviorálne predmety zachytávajú dynamiku modelov. Patria sem interkacie, ktoré definujú komunikáciu medzi elementami a stavové mašiny zachytávajúce stav modelu.

Zoskupovacie predmety

Zoskupovacie predmety, sú mechanizmy, ktoré zoskupujú elementy UML modelu, patria sem balíčky, ktoré zoskupujú štrukturálne a behaviorálne predmety.

Anotačné predmety

Anotačné predmety definujú mechanizmy na potreby popisu modelu, patria sem (Notes) poznámky.

2.5.2 Relácie

Ďalším stavebným blokom UML sú relácie. Tieto relácie znázorňujú ako sú elementy systému asociované medzi sebou. Jednotlivé asociácie popisujú funkcionalitu systému. Patria sem Dependency (závislosť), Association (asociácia), Generalization (generalizácia) a Realization (realizácia). [8]

- Závislosť - Popisuje závislosť jedného objektu na druhom.
- Asociácia - Popisuje vzťahy medzi jednotlivými elementami.
- Generalizácia - Popisuje väzbu medzi elementami, konkrétne medzi všeobecným a špecifickým elementom, inak povedané popisuje dedičnosť.
- Realizácia - Popisuje vzťah dvoch elementov, ktoré sú prepojené. Jeden z nich môže popisovať vlastnosti elementu a druhý ich môže implementovať.

2.5.3 Diagramy

Posledným stavebným blokom UML sú Diagramy. Jedná sa o grafickú reprezentáciu množiny elementov v kombinácii predmetov a relácií. Diagramy sa využívajú na vizualizáciu systému z rôznych uhlov pohľadu. Diagramy pomáhaju lepšie pochopiť štruktúru a fungovanie systému. UML obsahuje niekoľko typov diagramov, každý tento typ je možné použiť na vizualizáciu inej časti prípadne perspektívy systému. Rôzne typy diagramov sú inak graficky reprezentované a majú iný význam. UML diagramy môžeme rozdeliť do troch kategórií, podľa toho, aké časti a vlastnosti systému modelujú. UML poskytuje dva základné typy diagramov a to štrukturálne diagramy a behaviorálne diagramy. Na obrázku 2.2 je zobrazená štrukturá diagramov v UML. [10] [9]

- **Štrukturálne diagramy** - reprezentujú statické aspekty softwaru, stabilnú štruktúru systému
- **Behaviorálne diagramy** - reprezentujú dynamické aspekty softwaru, meniace sa časti systému

Štrukturálne diagramy

Ako už bolo spomenuté, štrukturálne diagramy reprezentujé stále, nemenné časti sytému ako jeho štruktúra. Tieto statické časti môžu reprezentovať rôzne triedy, objetky, rozhrania, komponenty a iné. Do štrukturálnych diagramov patria napríklad diagramy tried, komponentov, objektov, balíčkov a iné [9]. V tejto sekcii sú popísané niektoré z nich.

Diagram tried zachytáva logickú štruktúru systému a reprezentuje objektovo-orientovaný pohľad na systém. Reprezentuje systémové rozhrania, triedy, ich atribúty operácie prípadne metódy a vzťahy medzi nimi. Tento typ diagramu sa využíva na detailnú reprezentáciu systému na účely programovania, prípadne na účely dátovehó modelovania. Jedná sa o najčastejšie používaný diagram, ktorý slúži na konštrukciu systému.

Diagram objektov tak, ako aj diagram tried reprezentuje statický pohľad na systém s rozdielom, že reprezentuje už konkrétne inštancie tried, čiže tieto diagramy skôr zodpovedajú implementovénemu systému v reálnom čase. Využívajú sa pri tvorbe prototypov. [10]

Diagram komponent zobrazuje ako sú komponenty systému navzájom pospájané za účelom vyobrazenia systému. Primárne slúži na vizualizáciu štruktúry komplexných systémov. Tieto diagramy môžu byť využité ako komunikačný nástroj medzi programátormi a stakeholdermi, prípadne môže slúžiť ako plán implementácie.[10]

Diagram balíkov znázorňuje usporiadanie elementov modelu do balíkov a vizualizuje závislosti medzi nimi.

Behaviorálne diagramy

Úlohou behaviorálnych diagramov je vizualizovať, špecifikovať, vytvoriť a dokumentovať dynamické aspekty systému. Popisujú správanie systému a interakcie medzi entitami v ňom ako pohyb dát v systéme a stav systému. Používajú sa na popis funkcionality systému. Medzi tieto diagramy patria napríklad aktivity diagramy, diagramy prípadov užitia, diagramy stavu systému a iné [9]. Podľa niektorých zdrojov existuje tretí typ diagramov a to diagramy interakcie. V tejto práci budú brané ako podmnožina behaviorálnych diagramov, lebo tak isto ako behaviorálne diagramy, tak aj o diagramy interakcie popisujú správanie systému. Patria sem napríklad sekvenčné diagramy alebo časové diagramy.

Diagram prípadov užitia zachytáva určitú funkcionalitu systému. Tento diagram popisuje prípady užitia a vzťahy medzi účastníkmi systému a systémom samotným. Používa sa napríklad vo fáze zbierania biznis požiadaviek od užívateľa. Príklad diagramu prípadov užitia je možno vidieť na obrázku 3.1.

Sekvenčný diagram je diagramom interakcie a zobrazuje sekvenciu správ vymieňaných medzi komponentami systému v čase. Príklad sekvenčného diagramu je na obrázku 4.3.

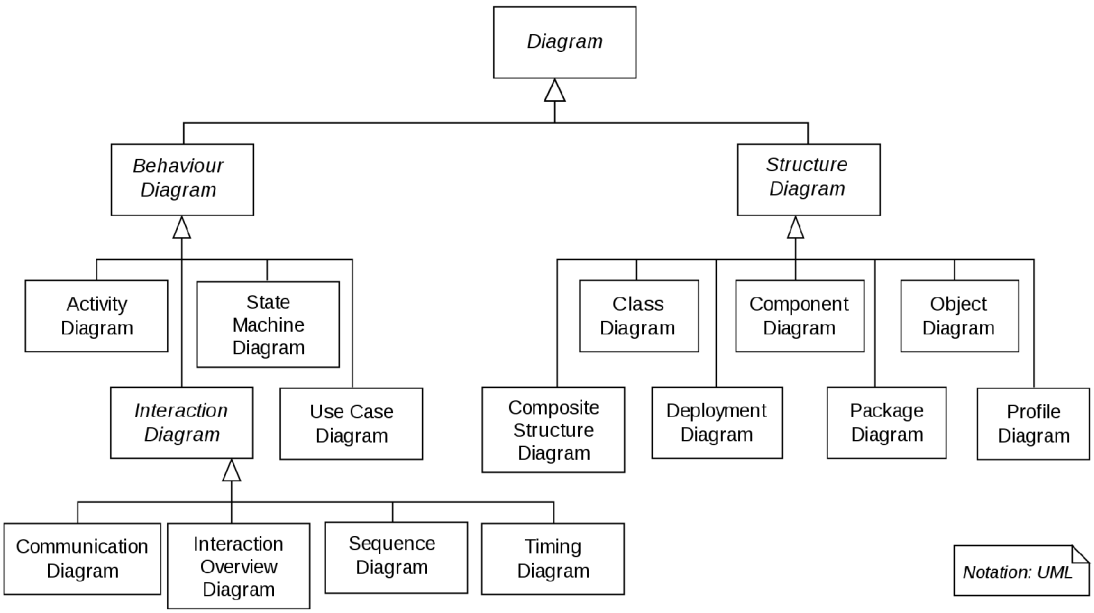
Diagram aktivít sa používa na popis kontrolného toku v systéme, inak povedané slúži na zachytenie funkcionality systému za behu, popisuje nejaký proces v systéme.

Všetky tieto diagramy sa využívajú pri tvorbe systémov.

2.6 Model-Driven Development

Ako bolo spomenuté v sekcii 2.3, Model-Driven Development (MDD) je poddisciplínou Model-Driven Engineeringu zaoberajúca sa aplikáciou modelovo riadených technológií na aktivity týkajúce sa vývoja softwaru. Je to prístup, ktorý využíva modely ako prvotriedne entity na vytváranie produktov. Motiváciou prečo používať modelovo riadené prístupy je to, že bežný software sa neustále vyvíja a s ním sa vyvíjajú aj platformy, na ktorých tento software pracuje. Tým pádom je nutné dané produkty neustále upravovať pre platformy, na ktorých majú fungovať, to so sebou prináša potrebu investovať viac času a zdrojov do ich úprav. Jedným z cieľov modelovo riadených prístupov je práve tomuto zamedziť a ušetriť zdroje. MDE posúva vývoj softwaru na vyššiu úroveň abstrakcie, tým pádom nie je nutné prispôsobovať modely každej zmene v platforme. Jeden abstraktný model môže byť znovupoužitý pre viaceré platformy bez nutnosti zásahu. Okrem toho je ľahšie aj ich udržiavanie.

[3] https://en.wikipedia.org/wiki/Unified_Modeling_Language



Obr. 2.2: Hierarchia UML diagramov[3]

Modely ako artefakty na vývoj sú ľahšie pochopiteľné aj pre nezainteresované strany. Jedným z problémov využitia modelov na vývoj je schopnosť zachytiť všetky implementačné detaily [12].

2.6.1 Model-Driven Architecture

Model-Driven Architecture (MDA) je konkrétna realizácia MDD od spoločnosti OMG. Táto realizácia je založená na viacerých štandardoch OMG, ako napríklad Meta-Object Facility (MOF), XML Metadata Interchange (XMI), Object Constraint Language (OCL), UML a iné, ktoré budú bližšie popísané nižšie. MDA pracuje na báze UML modelov, ktoré sú popísané pomocou Domain Specific Language (DSL). DSL je jazyk, ktorý sa špecializuje na určitú aplikačnú doménu. Príkladom môže byť jazyk SQL alebo jazyk ALF, ktorý bude bližšie popísaný v sekcii 2.7.2. MDA je založené na troch princípoch [12].

- **Priama reprezentácia** - umožňuje spájanie problémov s riešeniami pomocou DSL
- **Automatizácia** - prvky predstavené DSL, majú byť spracované nástrojmi, ktoré spájajú medzeru doménových konceptov s implementačnými technológiami
- **Štandard** - umožňuje prepojiť technické riešenia

Hlavnou myšlienkou MDA je to, že každý software môže byť použitý na rôznych platformách. Z tohoto dôvodu sa v MDA využívajú modely, ktoré sú pomocou transformačných pravidiel pretransformované na iné modely. Jednotlivé transformácie sa dejú na základe QVT štandardu[4]. V MDA sú definované tri typy modelov [12].

- **Computation independent model (CIM)** - Tento model sa zameriava na zachytenie obecných požiadaviek systému, detailná štruktúra je neurčená.

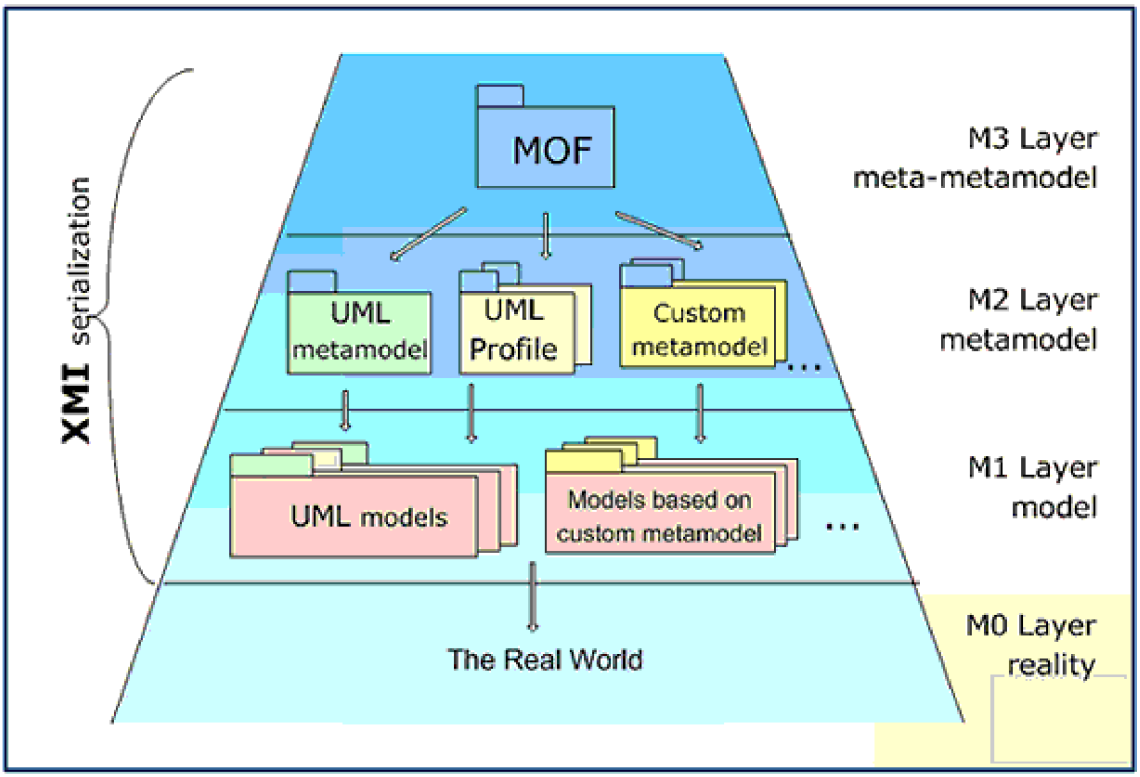
[4] https://www.omg.org/spec/QVT/1.3/PDF

- **Platform independent model (PIM)** - Zachytáva vlastnosti systému, ktoré sú nepodstatné pre platformu, čiže zmena v platforme sa ich nedotkne. Popisuje štruktúru a chovanie systému.
- **Platform specific model (PSM)** - Model závislý na konkrétnej platforme obsahujúci všetky detaily.

Meta-Object Facility

Meta-Object Facility je prostredie od spoločnosti OMG, ktoré umožňuje exportovať modely z jednej aplikácie a importovať ich do inej. Poskytuje hierarchický systém na definíciu modelov, tieto modely rozdeľuje do štyroch úrovní [23].

- **M3 úroveň** - metametamodel, určuje štruktúru a globálny typový systém pre nižšie úrovne
- **M2 úroveň** - metamodel, popisuje typy elemenetov úrovne M1, ich vzťahy, usporiadania a obmedzenia
- **M1 úroveň** - model popisujúci objekty
- **M0 úroveň** - konkrétny objekt



Obr. 2.3: Hierarchia MOF modelov[5]

[5] http://www.jot.fm/issues/issue_2006_11/article4/

XML Metadata Interchange

XML Metadata Interchange (XMI) je štandard spoločnosti OMG, slúžiaci na výmenu dát za pomoci XML. XMI môže byť použitý na uloženie metadát na popis modelov, ktoré môžu byť v MOF[24]. Najčastejšie sa XMI používa na serializáciu modelov, ale môže byť použitý aj ako výmenný formát pre UML modely [12].

Object Constraint Language

Object Constraint Language je formálny jazyk popisujúci pravidlá aplikujúce sa na UML modely. Pomocou OCL môžeme špecifikovať operácie, ktoré keď spustíme, menia stav systému. Ďalej je možné špecifikovať ním operácie, ktoré obmedzujú modely. Taktiež je ním možné špecifikovať invarianty v triedach prípade sa dá použiť ako dotazovací jazyk. V MDA je ho možné využiť ako navigačný jazyk, ktorý slúži na komunikáciu medzi viacerými technickými priestormi [22].

V MDA platí, že všetko je model, aj transformácie modelov sa považujú za model. Ďalej pre modely platí, že model nižšej úrovne musí odpovedať modelu vyššej úrovne. Hovoríme, že model vyhovuje metamodelu vtedy, keď má každý element modelu definovaný metaelement v metamodeli. To isté platí aj vo vzťahu medzi metamodelom a metametamodelom [12].

2.7 Executable UML

Executable UML (xUML) je koncept softwarového vývoja a zároveň aj vysoko abstraktný jazyk, ktorého cieľom je kompilovať a spustiť UML modely. Jedná sa o modifikáciu UML, v ktorej je možné popísať detaily modelu do takej úrovne, aby bolo možné ho spustiť, prípadne generovať z neho funkčný kód. xUML spája podmnožinu UML modelov so spúšťateľnou sémantikou. Modely v tomto prostredí je možné spúšťať, ladiť, testovať a kompilovať do menej abstraktných jazykov. xUML podporuje MDA, umožňuje napríklad preklad PIM modelov do PSM modelov [19].

2.7.1 Foundational Subset for Executable UML

Foundational Subset for Executable UML (fUML) je štandard OMG. Jedná sa o platformovo nezávislú spustiteľnú podmnožinu štandardného UML, ktorá poskytuje modelovacie koncepty pre definovanie UML tried a pre definovanie správania týchto tried [26]. Táto podmnožina zahŕňa typické modelovacie konštrukcie UML ako triedy, dátové typy, asociácie a enumerácie. Taktiež poskytuje prostriedky pre zadefinovanie chovania modelu pomocou UML aktivít. fUML umožňuje nielen špecifikovať správanie UML tried, ale taktiež aj správanie MOF metatried. fUML využíva rôzne typy diagramov z UML, napríklad na zadefinovanie statickej štruktúry používa diagramy tried a na zadefinovanie správania diagramy aktivít [18].

2.7.2 Action Language for Foundational UML

Modelovanie systémov pomocou grafickej reprezentácie nemusí byť vždy dostačujúce a nie je ním možné zachytiť všetky detaily systému, preto sa na popis modelov používa aj textová reprezentácia. Existuje niekoľko možností na textuálny popis modelu. Jednou z nich je jazyk Action Language for Foundational UML (ALF) od spoločnosti OMG [25]. ALF okrem

konkrétnej textuálnej syntaxe na popis fUML modelov poskytuje aj exekučnú sémantiku pomocou mapovania ALF syntaxe na abstraktnú syntax fUML modelov. Konkrétna syntax jazyka ALF je popísaná na základe bezkontextovej gramatiky zapísanej v Enhanced-Backus-Naur-Form (EBNF) forme a jeho abstraktná sytax je reprezentovaná UML modelom [11]. Primárnym cieľom ALF-u je pomocou textuálnej syntaxe špecifikovať spustiteľné správanie v rámci modelu. Napríklad špecifikovať telá metód tried potrebných pre operácie obsiahnuté v diagramoch tried. Syntax jazyka ALF je podobná syntaxi jazyka Java, používa implicitný typový systém a má zaistenú statickú typovú kontrolu. Podľa špecifikácie, je ALF kód možné spúšťať tromi spôsobmi [13].

- **Interpretatívna exekúcia** - ALF model je možné priamo interpretovať a spúšťať.
- **Kompilatívna exekúcia** - ALF model sa preloží do UML modelu tak, aby odpovedal fUML a je spustený podľa sémantiky špecifikovanej fUML.
- **Prekladová exekúcia** - ALF model je preložený do kódu, ktorý je spustiteľný na zvolenej platforme

Kapitola 3

Prípadová štúdia

Táto kapitola sa venuje konkrétnej prípadovej štúdii konferenčného recenzného systému. Je tu popísaná špecifikácia systému, ktorá zahŕňa všetky požiadavky na vyvíjaný systém. Patria sem funkčné a systémové požiadavky, požiadavky na roly v systéme a taktiež rozbor technológií, ktoré možno použiť na jeho vývoj.

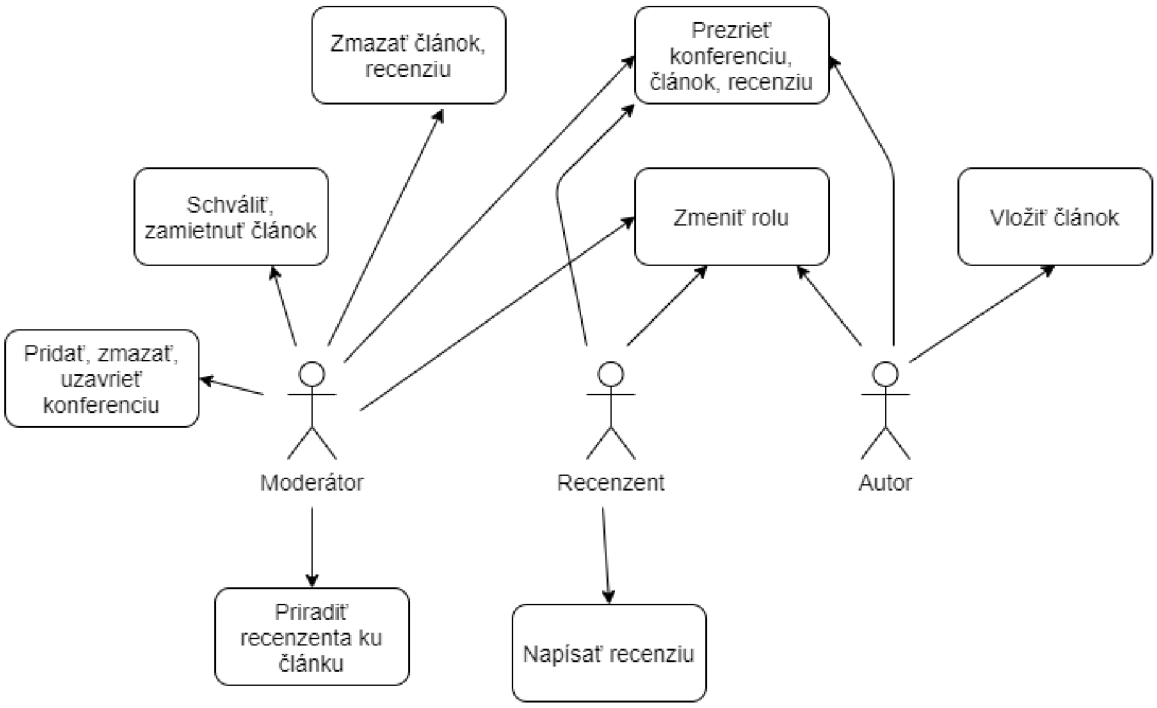
Súčasťou tejto práce je demonštrovať zvolené prístupy k vývoju softwaru na prípadovej štúdii konferenčného recenzného systému, ktorý umožňuje vytvárať konferencie, pridávať do nich články, ktoré následne možno schvaľovať a písať k nim recenzie. Tento systém je implementovaný v dvoch verziách. Prvá verzia je implementovaná programovaním. Druhá verzia systému je zrealizovaná metódou Model Driven Development. Systémy prinášajú nulovú biznis hodnotu a ich účel slúži čisto na demonštračné a experimentálne účely. Systémy spĺňajú základné funkčné požiadavky, ktoré sú popísané v sekciách nižšie.

3.1 Roly

V systéme vystupuje niekoľko typov užívateľov, každý z týchto typov zastáva inú rolu, môže vykonávať rôzne operácie a má iné oprávnenia. Konkrétne sa jedná o tri typy užívateľov, Moderátor, Recenzent a Autor.

- Typ užívateľa Moderátor je zodpovedný za vytváranie konferencií. Rozhoduje o témach v konferencii, schvaľuje a zamieta články priradené do konkrétnych konferencií, otvára a uzatvára konferencie na zverejňovanie článkov. Ďalšou jeho úlohou je priraďovať ku článkom recenzentov.
- Recenzent je typ užívateľa, ktorý je zodpovedný za písanie recenzií ku nemu priradeným článkom.
- Poslednou rolou v systéme je Autor. Jeho úlohou je pridávať články, čakajúce na schválenie k jednotlivým konferenciám.

Jeden užívateľ môže zároveň zastávať viaceré roly, ktoré si môže pri prihlásení podľa oprávnení zvoliť. Prípady užitia v systéme zobrazuje nasledujúci obrázok 3.1.



Obr. 3.1: Model prípadov užitia v systéme

3.2 Požiadavky

V tejto sekcii sú popísané všetky funkčné aj systémové požiadavky na implementovaný systém a ich detailný popis.

3.2.1 Funkčné požiadavky

Systém musí umožniť vytvárať nové užívateľské účty, pomocou prihlasovacieho mena a hesla. Po vytvorení konta je možné sa pomocou loginu a hesla prihlásiť do systému. Potom, čo sa užívateľ prihlási, má k dispozícii zoznam rolí, z ktorých si môže podľa oprávnení vybrať a pod touto rolou bude v systéme vystupovať. Užívateľ môže zmeniť rolu v systéme na hlavnej stránke. Na hlavnej stránke sa nachádza zoznam všetkých konferencií. Konferencie a ich obsah sú prístupné všetkým prihláseným užívateľom a je možné ich rozkliknúť. V rozkliknutej konferencii sa nachádza jej meno, popis a zoznam priradených článkov. Z konferencie sa užívateľ pomocou tlačítka dostane naspäť na zoznam konferencií. Konferencia nadobúda dva stavy. V prípade, že je konferencia otvorená môžu k nej užívatelia pridávať nové články a recenzie k nim. V prípade, že je konferencia uzavretá, nie je možné pridávať nové články alebo recenzie. V konferencii je možné kliknúť na daný článok, po rozkliknutí sa zobrazí obsah, meno článku, autor, jej text a zoznam všetkých recenzií napísaných ku danému článku. Po skončení je možné sa zo sytému odhlásiť.

3.2.2 Požiadavky na roly

Moderátor

Typ užívateľa Moderátor môže vytvoriť alebo odstrániť konferenciu. Na vytvorenie konferencie je potrebné, aby Moderátor zadal jej meno a popis. Ďalej je jeho úlohou pridať recenzentov ku novovloženým článkom. Všetky články na priradenie sú dostupné na separátnej stránke. Ďalšou úlohou Moderátora je schváliť alebo zamietnuť články, ku ktorým bola napísaná recenzia. Všetky články na schválenie sú zobrazené na separátnej stránke. Moderátor môže konferenciu uzamknúť, následne do nej už nebude možné pridávať nové články ani recenzie, stále je však možné prezerať si jej obsah.

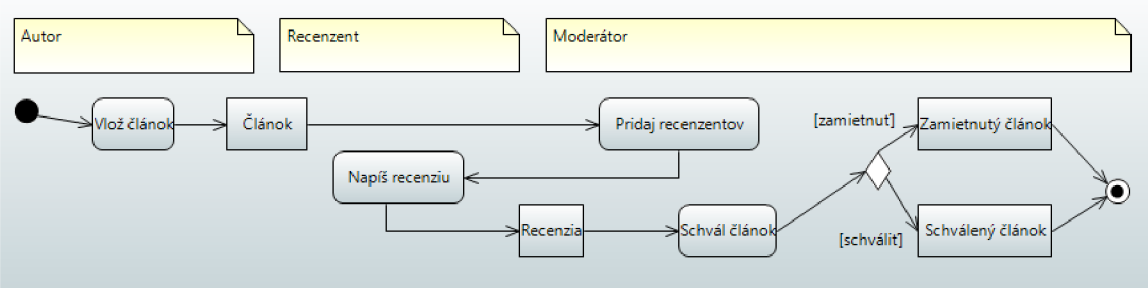
Autor

Jedinou úlohou autora je pridať článok ku konferencii. Rozklikne konferenciu, ku ktorej chce článok pridať a pomocou tlačítka môže vložiť článok. Na pridanie článku musí užívateľ zadať názov článku a text.

Recenzent

Recenzent si môže zobraziť jemu priradené články na recenzovanie a prezrieť si ich na separátnej stránke. Po rozkliknutí článku sa článok načíta a je k nemu možné pridať recenziu.

Všetky schválené články sa zobrazia v im prislúchajúcej konferencii. Schválené články si môže užívateľ rozkliknúť. Po rozkliknutí článku je k dispozícii jeho názov a text. Pod článkom sa nachádza zoznam recenzií vo forme komentárov, ktoré je možné si prezrieť. Z článku sa užívateľ pomocou tlačítka dostane naspäť do konferencie. Na obrázku 3.2 je zobrazený diagram aktivít znázorňujúci proces pridania a rencezovania článkov.



Obr. 3.2: Activity diagram znázorňujúci proces pridania článku do konferencie

3.2.3 Systémové požiadavky

Vzhľadom na to, že sa jedná o informačný systém musí byť vytvorený tak, aby bol dostupný naraz pre viacerých užívateľov, musí byť konkurentný a neblokujúci. Z tohoto dôvodu bola zvolená trojvrstvová architektúra systému. Systém sa skladá z troch častí - databáza, serverová aplikácia a klientská aplikácia.

- **Databáza** - obsahuje uložené všetky dáta systému

- **Serverová aplikácia** - zabezpečuje asynchrónny prístup ku dátam z databázy a obsahuje výpočtovú logiku
- **Klientská aplikácia** - slúži na zobrazovanie dotazovaných dát pre užívateľa a na interakciu so systémom

Databáza

Systém musí mať miesto na ukladanie dát, jedno z takýchto miest môže byť databáza.

Server

Serverová aplikácia musí zabezpečovať prístup k dátám z databázy, z toho dôvodu je nutné mať vrstu v aplikácii, ktorá umožní pristupovať k týmto dátam, zapisovať do databázy a mazať z nej. Hlavnú časť tvorí biznis vrstva, ktorá spracováva dáta predtým, ako sa odošlú klientskej aplikácii. Ďalšia časť, ktorú musí aplikácia obsahovať je komunikačné rozhranie, cez ktoré komunikuje s klientskou aplikáciou. Tým rozhraním môže byť napríklad Representational state transfer (REST)[1].

Klient

Systém musí byť dostupný pre užívateľov využívajúcich rôzne typy zariadení, preto je vhodné klienta implementovať ako webovú aplikáciu. Hlavnou úlohou klienta je zobrazovať dáta a zabezpečiť interakciu užívateľa so sytémom. Ideálne je použiť návrhový vzor Model-View-Controller (MVC). Tento návrhový vzor sa skladá z troch častí.

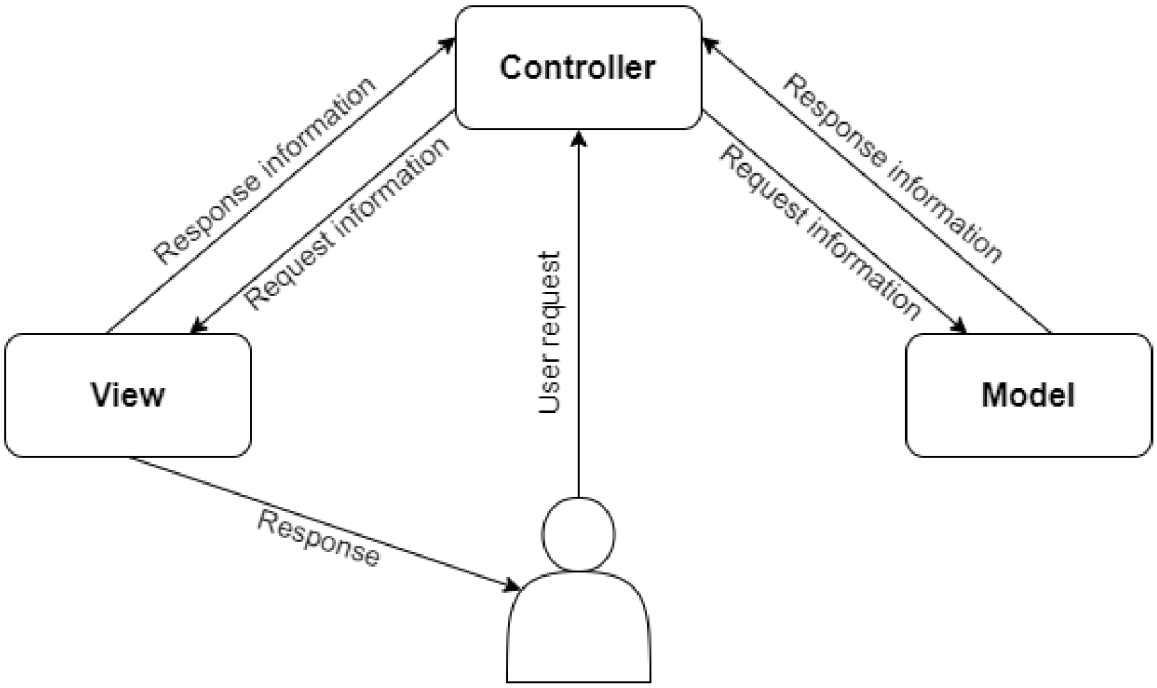
- **Model** - obsahuje aktuálne dáta nezávislé na užívateľskom rozhraní
- **View** - jedná sa o prezentačnú vrstvu, ktorá zobrazuje dáta užívateľovi
- **Controller** - riadiaca jednotka, ktorá spracováva prichádzajúce požiadavky, vykonáva všetku logiku, manipuluje s dátami a komunikuje s View.

Obrázok 3.3 zobrazuje princíp fungovania MVC. Užívateľ zadá do prehliadača URL adresu, ktorú aplikácia porovná s preddefinovanými cestami. V prípade zhody cesty sa zavolá príslušný Controller, ktorý preberá riadenie. Controller využíva Model aby získal všetky potrebné dáta, typicky prevolá servisy, ktoré dopytujú backend o dáta. Controller načíta View a získané dáta mu predá na zobrazenie. [15]

3.3 Technológie

Na implementáciu systému je potrebné vhodne zvoliť technológie. Neexistuje jednotná technológia, ktorou by sa dal celý systém implementovať a preto je potrebné zvoliť niekoľko kompatibilných technológií na vývoj tohoto systému. Pri programovaní je k dispozícii niekoľko možností ako aplikáciu implementovať. Vzhľadom na to, že systém sa skladá z troch častí - databáza, serverová aplikácia a klientská aplikácia, každú túto časť je nutné implementovať rôznymi technológiami.

[1] https://www.codecademy.com/articles/what-is-rest



Obr. 3.3: Princíp fungovania Model-View-Controlleru

3.3.1 Databáza

Aktuálne je na trhu niekoľko rôznych typov databáz, ako napríklad relačné, grafové, objektové a iné. Pre túto prácu bol zvolený relačný typ databázy, z toho dôvodu, že je najpoužívanejší. Konrétne sa jedná o databázu MariaDB.

3.3.2 Serverová aplikácia

Tak ako v prípade databázy, aj pri serverovej aplikácii je k dispozícii množstvo technológií, ktoré možno použiť na jej implementáciu. Na vývoj boli zvolené nasledujúce technológie.

- **Node.Js** - Open source runtime environment pre JavaScript, ktorý umožňuje spúšťať kód Javascriptu nie v prehliadači, ale na strane serveru. Umožňuje písať asynchrónny kód, ktorý je vhodný na vytváranie serverových aplikácií.
- **Express.Js** - Jedná sa o minimalistický webový framework, slúžiaci na vývoj webových aplikácií. Ponúka množstvo HTTP utilít, umožňuje definovať vlastný middleware a uľahčuje prácu s parsovaním payloadu, s manipuláciou cookies, ukladaní session, pri routovaní a pri iných.
- **TypeScript** - Nadmnožina jazyka JavaScript, ktorá dodáva statickú typovú kontrolu. Ďalej umožňuje vytvárať rozhrania a triedy a tým lepšie štrukturovať kód. TypeScript je kompatibilný s JavaScriptom, do ktorého sa kompiluje. Je použiteľný ako na backende tak aj na frontende.
- **TypeORM** - Objektovo-relačný manažment modul, ktorý beží v Node.Js, slúžiaci na objektovo-relačné mapovanie. Automaticky umožňuje vytvárať databázové tabuľkové

schémy na základe užívateľom definovaných modelov. Ponúka operácie na jednoduchú manipuláciu s dátami v databáze.

3.3.3 Klientská aplikácia

Na tvorbu webových aplikácií je k dispozícii množstvo webových frameworkov, ktoré slúžia na vývoj reaktívnych aplikácií. Najznámejšie z nich sú Angular, Vue.Js A React. Každý z týchto webových frameworkov má svoje klady a protiklady. Pre túto prácu boli zvolené nasledujúce technológie.

- **Angular** - Je framework slúžiaci na tvorbu interaktívnych webových rozhraní. Zvolený bol z dôvodu, že má pevne stanovené pravidlá na štrukturovanie kódu. Každá Angular aplikácia je delená do modulov, ktoré sa skladajú z komponentov, servís a direktív. Taktiež natívne využíva Typescript, tým pádom je vynútená statická typová kontrola. Vytvárajú sa s ním Single Page Application (SPA) [2].
- **HTML5** - Značkovací jazyk používaný na štrukturovanie a prezentovanie obsahu na webe.
- **LESS** - Jedná sa o dynamický jazyk. Less je preprocesor jazyka CSS, po kompilácii sa z neho generuje obyčajné CSS. Slúži na pridávanie štýlov pre webové dokumenty. Umožňuje vytvárať čistejšiu štruktúru kódu.
- **Bootstrap** - Open source toolkit na vývoj webov, ponúka preddefinované štýly.

3.3.4 Model Driven Development za pomoci jazyka ALF

Ako bolo spomenuté v sekcii 2.7.2, ALF poskytuje rozšírený zápis, ktorý môže byť použitý na štrukturované modelovanie elementov a je ním možné vytvoriť funkčný program. Aby sme mohli vytvárať funkčné programy pomocou jazyka ALF je nutné si k tomu rozbehnúť zodpovedajúce prostredie, v ktorom budeme vyvíjať. V tejto práci boli skúmané dve prostredia, v ktorých je možné implementovať software za pomoci jazyka ALF.

Magic draw

MagicDraw[3] je nástroj na modelovanie, ktorý podporuje UML. Slúži na návrh objektovo orientovaných systémov, databáz a iných. Poskytuje mechanizmy na generáciu kódu, ktoré podporujú rôzne programovacie jazyky ako napríklad Java, C#, C++ a iné. Umožňuje pridávať rôzne pluginy, ktoré rozširujú jeho funkcionalitu. Jedným z týchto pluginov je Alf Plugin [4]. Tento plugin umožňuje využitie jazyka ALF v MagicDraw. Poskytuje editor a prekladač jazyka a umožňuje prekladať ALF do fUML modelov. S použitím Cameo Simulation Toolkit je možné ALF kód priamo spúšťať. Nevýhodou tohoto programu je, že sa jedná o platený program. Existuje síce verzia, ktorá je zdarma, ale má limitované funkcie ako počet elementov, ktoré sa dajú použiť.

[2] https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/
[3] https://www.nomagic.com/products/magicdraw
[4] https://www.nomagic.com/product-addons/magicdraw-addons/alf-plugin

Eclipse studio

Integrované vývojové prostredie Eclipse[5] je primárne určené na vývoj Java aplikácií, ale umožňuje pridávať rôzne pluginy na prispôsobenie tohoto prostredia. Po pridaní pluginov je v ňom možné vyvíjať aj v iných jazykoch, jedným z nich môže byť aj ALF. Pre potreby vývoja aplikácií v ALF-e je nutné doinštalovať niekoľko pluginov.

- **Papyrus**[6] - Poskytuje integrované prostredie na prácu s UML modelmi.
- **Moka**[7] - Umožňuje spúšťať, ladiť a animovať modelované elementy.
- **Nebula**[8] - Pridáva sadu UI nástrojov.
- **Papyrus Software Designer**[9] - Nástroj slúžiaci na generovanie kódu a reverzné inžinierstvo pre jazyky C++ a Java.

[5] https://www.eclipse.org/ide/
[6] https://www.eclipse.org/papyrus/
[7] https://marketplace.eclipse.org/content/papyrus-moka
[8] https://www.eclipse.org/nebula/
[9] https://wiki.eclipse.org/Papyrus_Software_Designer

Kapitola 4

Návrh a implementácia

V tejto kapitole je popísaný návrh a implementácia oboch systémov, jedného pomocou programovania tradičnými jazykmi druhého pomocou jazyku ALF. Celý systém pozostáva z troch častí a to databázy, serverovej aplikácie a klientskej aplikácie. Každá časť je v tejto kapitole detailne popísaná.

4.1 Databáza

Pre potreby fungovania systému je nutné ukladať určité dáta do databázy. Vzhľadom na to, že táto práca je zameraná na konferenčný recenzný systém, je potrebné ukladať dáta o konkrétnych konferenciách, článkoch v konferenciách, recenziách napísaných ku článkom a užívateľoch. Ďalej treba uchovať dáta o recenzentoch priradených ku článkom a informácie o užívateľoch. Prvým krokom na vytvorenie databázy bol jej návrh pomocou ER diagramu následne bola databáza implementovaná pomocou relačno-objektového mapovania, ktoré vzniklo z entít vytvorených v serverovej aplikácii. Pre potreby uloženia dát bolo v databáze vytvorených päť tabuliek. Na obrázku 4.1 je zobrazený diagram tabuliek databázy a vzťahy medzi jednotlivými tabuľkami.

Tabuľka conference

Táto tabuľka obsahuje informácie o všetkých konferenciách, obsahuje atribúty:
id - jednoznačný identifikátor záznamu
name - meno konferencie
description - popis obsahu konferencie
status - určuje stav databázy, či je otvorená na pridávanie článkov a recenzií alebo uzavretá
userId - ID užívateľa, ktorý konferenciu vytvoril

Tabuľka article

Tabuľka obsahuje informácie o všetkých článkoch v systéme a obsahuje atribúty:
id - jednoznačný identifikátor záznamu
name - meno článku
content - text konkrétneho vloženého článku
status - stav článku, môže byť schválený alebo zamietnutý
conferenceId - ID konferencie, ku ktorej článok prislúcha
userId - ID užívateľa, ktorý článok vytvoril

Tabuľka review

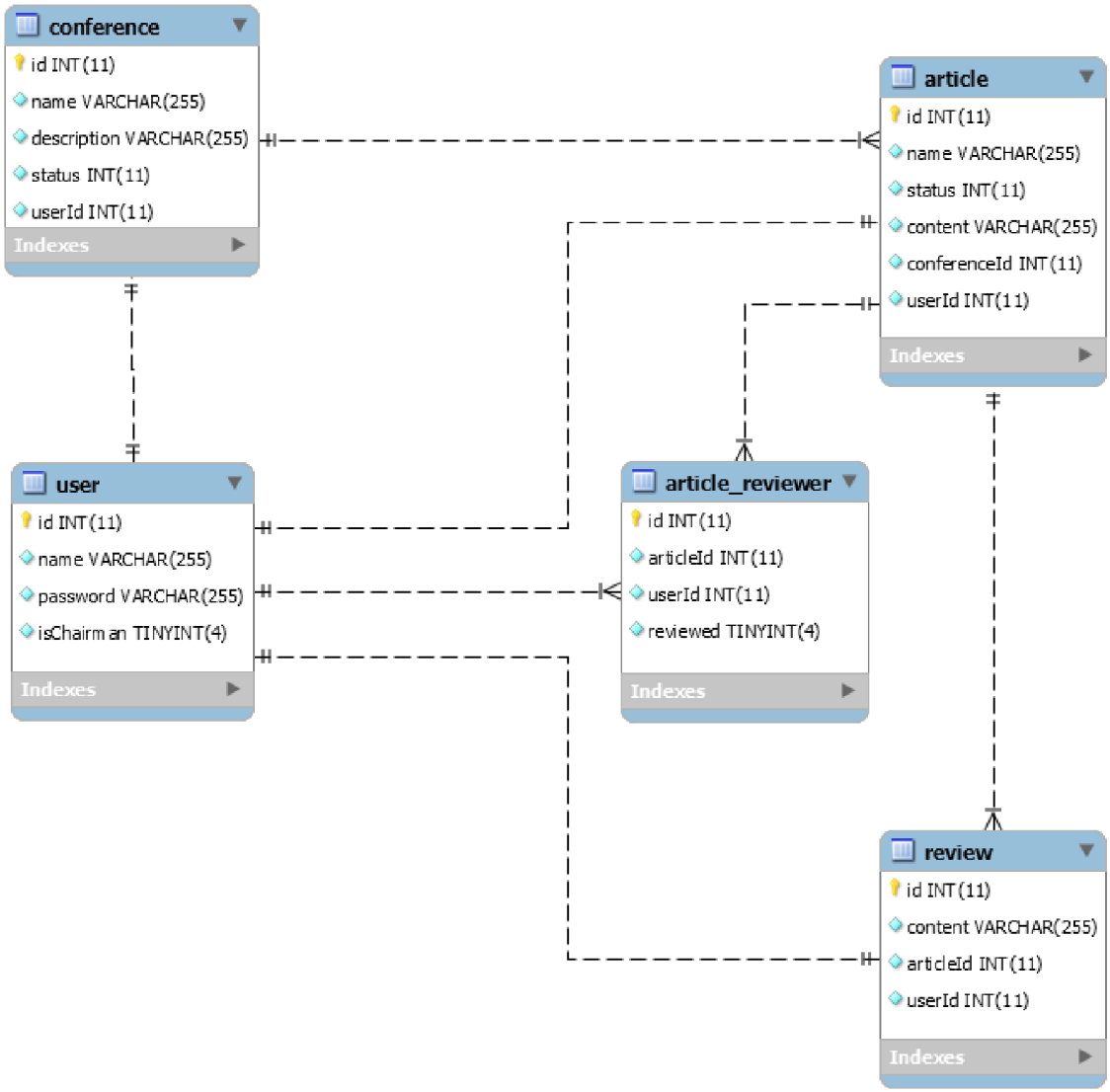
Tabuľka obsahuje informácie o recenziách článkov v systéme a má tieto atribúty:
id - jednoznačný identifikátor záznamu
content - text vloženej recenzie
articleId - ID článku, ku ktorému recenzia prislúcha
userId - ID užívateľa, ktorý článok vytvoril

Tabuľka article_reviewer

Väzobná tabuľka, ktorá zobrazuje vzťah M:N pre tabuľky article a user. Jeden článok môže mať M recenzetov a jeden recenzent môže recenzovať N článkov. Tabuľka obsahuje nasledujúce atribúty:
id - jednoznačný identifikátor záznamu
articleId - ID článku na recenziu
userId - ID recenzenta, ktorý má článok recenzovať
reviewed - hodnota udávajúca, či daný článok bol recenzovaný určeným recenzentom alebo nie

Tabuľka user

Táto tabuľka obsahuje informácie o užívateľoch systému.
id - jednoznačný identifikátor záznamu
name - meno užívateľa
password - heslo užívateľa, vo forme zašifrovaného reťazca
isChairman - hodnota definujúca oprávnenia moderátora v systéme



Obr. 4.1: Entity Relationship Diagram databázy

4.2 Server

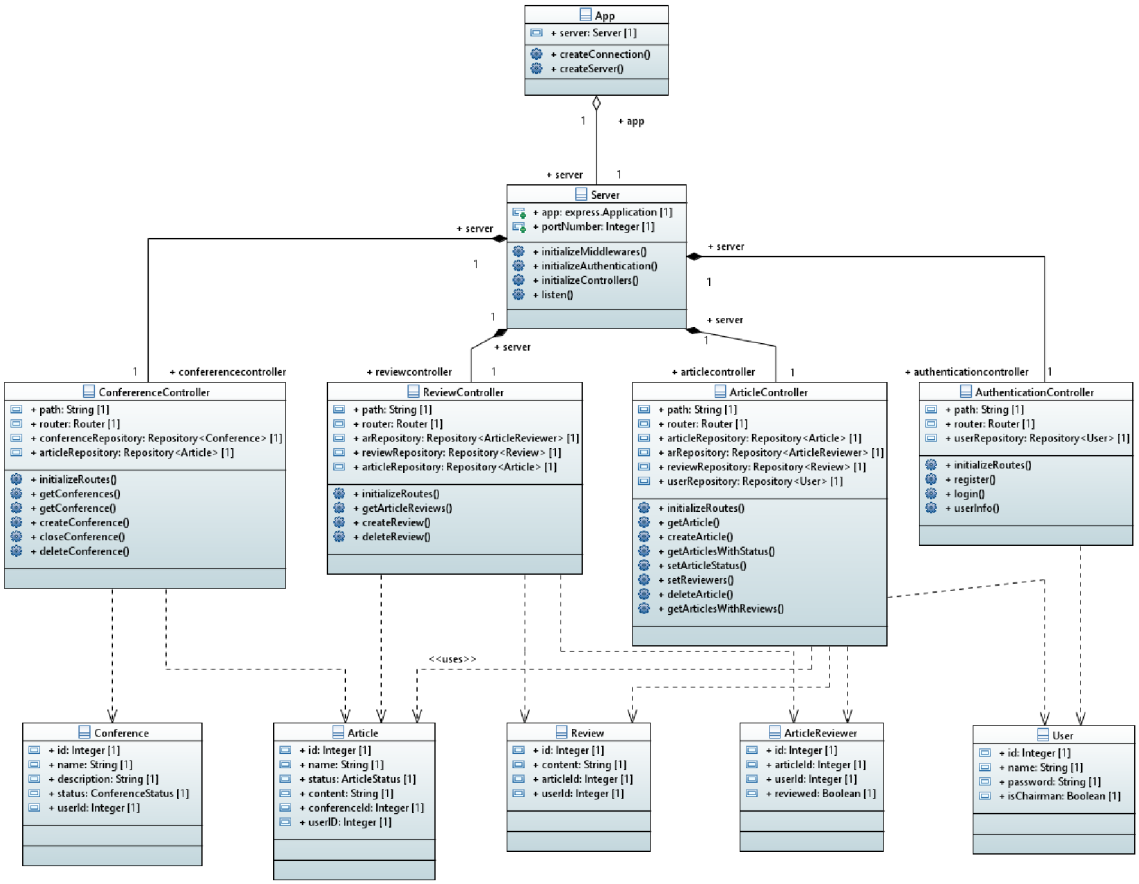
Prvým krokom bol návrh aplikácie, pomocou rôznych skečov, ktoré sa neskôr preniesli do výslednej podoby triedneho diagramu. Bol aplikovaný štandardný postup návrhu, počas ktorého boli vytvorené diagramy pre systém, Tieto diagramy boli postupne upravované. Zlý počiatočný návrh môže mať v produkčných podmienkach nepriaznivé podmienky, ako zvýšenie nákladov a predĺženie vývoja aplikácie.

Na vytvorenie serverovej časti boli použité technológie Node.js spolu s Express.Js a jazykom Typescript. Úlohou serveru je odpovedať na klientské požiadavky a získať, prípadne uložiť a zmazať dáta z databázy a taktiež autentizovať užívateľa. Celá aplikácia sa spúšťa v triede App, kde sa vytvára pripojenie k databáze a vytvára sa inštancia triedy Server.

V triede Server sú inicializované všetky kontrolery a middleware na parsovanie klientských požiadaviek, JsonWebTokenov pre potreby autentizácie užívateľa a spúšťa sa v nej naslúchanie klientským požiadavkám. Jednotlivé požiadavky spracúvajú kontroléry s prislúchajúcou API cestou. V systéme sú implementované štyri kontroléry.

- **Authentication controller** - Slúži na autentizáciu užívateľa a získavanie užívateľských informácií. Informácie ako heslo sa šifrujú pomocou bcrypt modulu.
- **Conference controller** - Spracováva všetky požiadavky týkajúce sa konferencií.
- **Article controller** - Spracováva všetky požiadavky týkajúce sa článkov v konferencii.
- **Review controller** - Stará sa o spracovanie požiadaviek týkajúcich sa recenzií.

Pre prístup k dátam v databáze sa využíva modul TypeORM. TypeORM pre potreby objektovo-orientovaného programovania zabezpečuje objektovo-relačné mapovanie, ktoré konveruje dáta medzi nekompatibilným relačným a objektovo-orientovaným prístupom. Ku každej databázovej tabuľke existuje entita (objekt), ktorá ju reprezentuje. Pre každú entitu v systéme existuje repozitár, pomocou ktorého je možné k dátam v databáze pristupovať. Tento repozitár poskytuje metódy, ktoré odpovedajú príkazom SQL jazyka. V diagrame 4.2 je zachytená štruktúra serveru za pomoci triedneho diagramu.



Obr. 4.2: Triedny diagram zobrazujúci štruktúru serveru

4.3 Klient

Klientská aplikácia je implementovaná za pomoci Javascriptového frameworku Angular. Prvým krokom bol rovnako ako pri serverovej časti návrh aplikácie, hlavne pomocou rôznych skečov. Pri začatí implementácie bol použitý tool angularCLI na vytvorenie projektu, ktorý umožní založiť nový projekt tým, že vygeneruje počiatočný modul a triedu so všetkými záležitošťami, ktoré sú potrebné na jeho fungovanie. Sú vygenerované dva moduly, `app.module` a `app-routing.module`. [1]

- **app.module** - V tomto module sú deklarované všetky komponenty, ktoré k nemu náležia, ďalej sú tu importované všetky moduly, ktoré boli využité a je tu zadefinovaný komponent, z ktorého sa celá aplikácia spúšťa.
- **app-routing.module** - Je to modul, v ktorom je deklarovaná navigácia v celej aplikacii a sú tu definované cesty, pomocou ktorých sa predáva riadenie aplikácie jednotlivým komponentom.

V kapitole 3.3.3 bolo povedané, že Angular aplikácia sa skladá z komponentov, servís a direktív. Jedná sa o stavebné bloky Angularu, kde každý z týchto blokov má svoju funkcionalitu. Celé fungovanie Angular aplikácie spočíva na základe návrhového vzoru dependency injection[1]. Kde sa každému komponentu odoberie zodpovednosť za získavanie objektov, ktoré potrebuje na svoju činnosť. Miesto toho sa tieto objekty vložia do jeho konštruktoru.

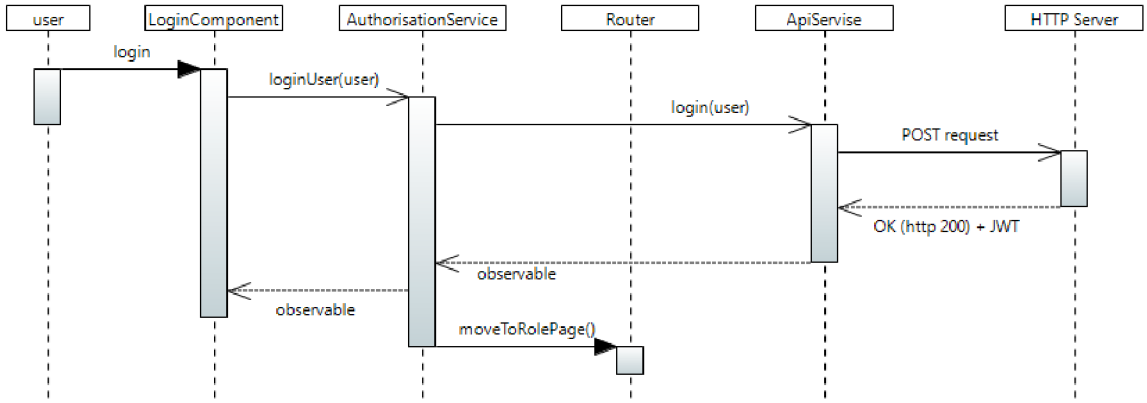
- **Komponent** - Jeden komponent pozostáva z typescriptového súboru, kde je implementovaná jeho logika. Potom z HTML šablóny, ktorá tvorí hierarchiu DOM elementov, ktoré udávajú kostru komponentu. Ďalej je tu súbor `.specs.ts`, v ktorom sú implementované jednotkové testy prislúchajúce komponentu a `.css` prípadne `.less` súbor, ktorý obsahuje kaskádové štýly. Na fungovanie komponentu je potrebný len typescript file, ktorý môže obsahovať aj šablónu HTML. Komponent vytvára DOM hierarchiu, definuje správanie a je zodpovedný za tok dát a viazanie týchto dát na HTML šablónu.
- **Direktíva** - Direktíva narozdiel od komponentu nevytvára DOM, ale modifikuje už existujúce DOM elementy v inštancii komponentu.
- **Servisa** - Je singleton objekt, ktorý obsahuje metódy na prácu s dátami. Cieľom servís je zdielať dáta a funkcie medzi komponentami aplikácie, prípadne posielať požiadavky na server. [2]

Celá aplikácia sa spúšta z `app.component`, v ktorej šablóne sa nachádza direktíva `<router-outlet>`. `<router-outlet>` a slúži ako placeholder, ktorý Angular dynamicky napĺňa podľa aktuálnej adresy a na základe neho aktualizuje HTML šablónu na zobrazenie. Aplikácia sa skladá z jedenástich komponentov. Každý komponent tvorí logiku a vzhľad jednej stránky, ktorá sa zobrazí. Potom obsahuje štyri servisy, `ApiService`, `AuthorisationService`, `DataService` a `EnumService`, kde každá plní iný účel. Na obrázku 4.3 je zobrazená aktivita prihlásenie sa užívateľa.

- **ApiService** - Slúži ako wrapper pre HTTP metódy `get` a `post`, pridáva k nim HTTP hlavičku s JsonWebTokenom slúžiacim na autentizáciu a nastavuje HTTP parametre.

[1] https://www.jamesshore.com/Blog/Dependency-Injection-Demystified.html

- **AuthorisationService** - Jej úlohou je uchovávanie stavu o aktuálne prihlásenom užívateľovi, taktiež slúži na prihlasovanie a odhlasovanie sa zo sytému.
- **DataService** - Obsahuje metódy potrebné na získanie dát zo serveru.
- **EnumService** - Slúži len na preklad enumeračných typov na stringy.



Obr. 4.3: Sekvenčný diagram popisujúci prihlásenie užívateľa

4.4 ALF

Ako už bolo spomínané v kapitole 2.7.2, grafická reprezentácia modelu nemusí postačovať na zachytenie všetkých jeho detailov. Z toho dôvodu boli zavedené jazyky ako ALF, ktoré textuálnou formou umožnia popísať správanie prípadne štruktúru systému. Preto, aby bolo vôbec možné pracovať s jazykom ALF je nutné si najskôr rozchodiť odpovedajúce prostredie. Vzhľadom na to, že sa jedná o jazyk, ktorý nie je veľmi rozšírený, neexistuje ani veľké množstvo nástrojov, v ktorých by sa dal jazyk ALF využiť. V tejto práci boli vyskúšané dve prostredia, v ktorých je možné pracovať s jazykom ALF.

4.4.1 Prostredie Eclipse

Prvým z nich bolo prostredie Eclipse s rozšíreniami Papyrus, Moka, Nebula a Papyrus Software Designer. Na rozchodenie tohoto prostredia bol využitý návod [21], ktorý okrem nastavenia prostredia znázorňuje aj to, ako spúšťať a ladiť ALF kód. Prvým krokom bolo nainštalovať si Javu 8 a následne Eclipse studio, konkrétne verziu Oxygen 3. Do prostredia Eclipse je nutné si následne nainštalovať najnovšiu verziu modelovacieho prostredia Papyrus. Spolu s ním integrovaný editor pre jazyk ALF, ďalej nástroj Moka slúžiaci na spúšťanie a ladenie modelov. Nakoniec rozšírenie Nebula 2.0, ktoré umožňuje pridávať z navigačného menu rôzne widgety do Eclipse prostredia a rozšírenie Papyrus software designer slúžiace na generáciu kódu z modelu. V návode [21] nie sú popísané všetky kroky detailne, prípadne niektoré nie sú aktuálne. S rozchodením prostredia boli komplikácie, najmä čo sa týka kompatibility jednotlivých rozšírení. Tým vznikali problémy s naištalovaním ďalších rozšírení a pri spúšťaní programu. Po nainštalovaní správnych verzií všetkých rozšírení je nutné si zobraziť z nastavení widgety potrebné na prácu s jazykom ALF a modelmi, ako napríklad

zobraziť widget na ladenie a nastavovanie ladiacich konfigurácií, widget na zobrazovanie aktuálnej hodnoty v bode breakpointu, widget na zobrazovanie vlastností modelov a iné. Potom, čo je prostredie pripravené, je možné v jazyku ALF začať modelovať systém. V prostredí je možné implementovať správanie aj štruktúru modelu za pomoci jazyka ALF, aj keď účel jazyka ALF pre toto prostredie bol zamýšľaný len na reprezentáciu správania modelov v kontexte UML. Na kompletný textuálny zápis slúži referenčná implementácia jazyka ALF [2], ktorá umožňuje namodelovať štruktúru aj správanie modelu. Na otestovanie prostredia bol využitý model spomínaný v návode [21], daný model je možné upravovať, spúšťať a ladiť. Prvé komplikácie nastali s generáciou kódu z tohoto modelu. Papyrus Software designer umožňuje generovať kód z UML modelov, pri pokuse o generáciu kódu bola vytvorená len kostra triedy, správanie implementované za pomoci jazyka ALF vygenerované nebolo. Nebolo vygenerované z dôvodu, že Papyrus Software Designer nepodporuje generovanie kódu z jazyka ALF. Pre ďalšie možnosti generovania kódu je nutné pridať do prostredia Eclipse rozšírenie Acceleo[3], ktoré slúži ako generátor kódu, prípadne umožňuje nadefinovať vlastné pravidlá generácie kódu. Tak ako Papyrus Software Designer, tak ani Acceleo nepodporuje jazyk ALF, čiže je nutné implementovať si vlastné transformačné pravidlá.

Po nasledujúcich zisteniach bol vznesený dotaz smerovaný priamo spoločnosti OMG ohľadne rozchodenia prostredia a použiteľnosti jazyka ALF.

4.4.2 Prostredie MagicDraw

Komunikáciou s OMG bolo zistené, že existuje ALF fórum[4], v ktorom ale okrem jedného dotazu nie je iný príspevok. Ďalej sa ako hlavný nástroj pre ALF používa komerčný nástroj MagicDraw od spoločnosti NoMagic[5], ku ktorému boli poskytnuté aj materiály. Vzhľadom na to, že MagicDraw nemá voľnú licenciu, bola využitá jeho demo verzia. Inštalácia a rozchodenie prostredia bolo podstatne jednoduhšie ako v prípade prostredia Eclipse. Stačí si nainštalovať MagicDraw a priamo v ňom pridať rozšírenie ALF, jedná sa o integrovaný editor a rozšírenie Cameo Simulation Toolkit[6] slúžiace na spúšťanie fUML modelov. Rovnako ako Papyrus, tak aj MagicDraw umožňuje popísať štruktúru pomocou diagramov a správanie modelu za pomoci jazyka ALF. Nijak neumožňuje v jazyku ALF popísať štruktúru, jediné, čo dovoľuje je popísať správanie častí modelu. Vrámci testovania produktu boli vytvorené jednoduché triedne diagramy s metódami, ktorých správanie bolo popísané jazykom ALF. Následne je možné celý model, alebo jeho časť spúšťať priamo v MagicDraw. Po spustení sa otvorí simulačné okno s konzolou, v ktorom je možné správanie spúšťať, nastavovať breakpointy a meniť rýchlosť animácie. Ďalej je možné písať jednoduché testy za pomoci aktivít. Na vytvorenie jednoduchého modelu v tomto prostredí sa postupovalo podľa návodu[7]. Čo sa týka generovania kódu, MagicDraw umožňuje generáciu. Preto, aby bolo možné generovať kód je nutné si vytvoriť "Code Engineering Set", v ktorom sa zvolí cieľový jazyk a množina modelov, z ktorých má byť kód vygenerovaný. Pre potreby generovania kódu v MagicDraw je k dispozícii návod[8]. Tak ako v prípade prostredia Papyrus, tak aj pri MagicDraw sa počas generácie kódu vygeneruje len kostra modelu a správanie zapísané v jazyku ALF vygenerované nie je. Po vznesení dotazu na technickú podporu spoločnosti

[2] http://modeldriven.github.io/ALF-Reference-Implementation/
[3] https://www.eclipse.org/acceleo/
[4] https://alf.discourse.group/
[5] https://www.nomagic.com/
[6] https://www.nomagic.com/product-addons/magicdraw-addons/cameo-simulation-toolkit
[7] https://docs.nomagic.com/display/ALFP185/Getting+Started
[8] https://docs.nomagic.com/display/MD190/Code+Engineering

NoMagic ohľadne generácie kódu, bolo zistené, že MagicDraw nepodporuje generáciu kódu z jazyka ALF. Taktiež demo verzia umožňuje použiť len obmedzené množstvo modelovacích elementov, čo na vytvorenie celého systému stačiť nebude. Prostredie MagicDraw je tým pádom pre túto prácu nepoužiteľné.

Po tomto zistení bol vznesený spoločnosti OMG ďalší dotaz ohľadne generácie kódu z jazyka ALF. V odpovedi, ktorú poslali bol vysvetlený zámer jazyka ALF. Pôvodne mal ALF slúžiť na uľahčenie písania kompletných spustiteľných UML modelov, vplyvom nízkeho dopytu na trhu sa od tejto myšlienky upustilo a v dnešnej dobe slúži ALF primárne ako akčný jazyk v kontexte SysML[9] modelových simulácií a toto je aj zámer jeho použitia v prostredí MagicDraw. Okrem iného neevidujú verejne dostupný generátor, ktorý by umožnil generovať kód z jazyku ALF, no v poslednej dobe zaznamenávajú zvýšený záujem o použitie ALF-u pre vývoj softwaru, ako bolo pôvodne zamýšľané.

4.4.3 Štúdie zaoberajúce sa použitím jazyka ALF

Existuje niekoľko štúdii, ktoré sa venujú možnostiam jazyka ALF a technológiám s ním spojeným. Na túto prácu bolo preštudovaných viacero zdrojov a za zmienku stojí niekoľko z nich.

Combining Alf and UML in Modeling Tools

Jednou z týchto prác je *Combining Alf and UML in Modeling Tools* [28], ktorá sa venuje využitiu jazyka ALF v prostredí Papyrus. Stručne popisuje jazyk ALF, prostredie Papyrus, jeho limity a demonštruje jeho použitie na príklade modelu objednávky produktu. V závere práce sa spomína, že prostredie Papyrus poskytuje len základné funkcie na popis modelov za pomoci jazyka ALF a ešte bude trvať nejaký čas, pokiaľ budú k dispozícii nástroje novej generácie, ktoré by sa vyrovnali bežnému použitiu programovacích jazykov.

Executable Modeling with fUML and Alf in Papyrus: Tooling and Experiments

Podobnou témou sa zaoberá aj práca *Executable Modeling with fUML and Alf in Papyrus: Tooling and Experiments* [16], ktorá popisuje použitie jazyka ALF a fUML v prostredí Papyrus. Táto práca sa zameriava na experimentovanie a popisuje jeho limity.

On Open Source Tools for Behavioral Modeling and Analysis with fUML and Alf

Ďalšou zaujímavou prácou je *On Open Source Tools for Behavioral Modeling and Analysis with fUML and Alf* [20], ktorá popisuje voľne dostupné nástroje, ktoré je možné použiť pre jazyk ALF. Tak, ako bolo spomenuté práci [28], tak aj v tejto sa na záver spomína, že nástroje síce existujú, no ich možnosti sú obmedzené a stále sú len v inkubačnej fáze.

Z ďalších štúdií a článkov preštudovaných na túto prácu bola najrelevantnejším zdrojom informácií práca *Unifying Modeling and Programming with ALF* [11] z univerzity v Bayeruthe, práce *On the Generation of Full-fledged Code from UML Profiles and ALF for Complex Systems* [14] a *On the automated translational execution of the action language*

[9] https://sysml.org/

for foundational UML [13] z Mälardalen univerzity. V týchto troch prácach sa podarilo vygenerovať z ALF-u funkčný kód, za pomoci iných prístupov.

On the Generation of Full-fledged Code from UML Profiles and ALF for Complex Systems

Práca *On the Generation of Full-fledged Code from UML Profiles and ALF for Complex Systems* [14] sa venuje generácii C++ kódu pre zložitejšie systémy. Využíta jazyk ALF, vlastný nástroj CHESS a modelovací jazyk CHESS-ML. CHESS-ML model je model definovaný UML profilom, ktorý popisuje štruktúru. Správanie je definované pomocou jazyka ALF. Práca popisuje celkovú transformáciu, ktorá zahŕňa generáciu štrukturálnych a behaviorálnych aspektov systému. Na demonštráciu generácie kódu bol využitý podstatne zjednodušený Asynchronous Transfer Mode (ATM) Adaptation Layer 2 (AAL2) subsystém, ktorý sa využíva v telekomunikáciách na prenos hlasu. Generácia kódu pozostáva z množiny transformácií ako prevod CHESS-ML modelu do bližšie nešpecifikovaného Intermediate modelu (InterM) pomocou model-to-model transformácie (M2M) s využitím QVT. Ďalej transformácia modelu správavania definovanom v jazyku ALF do InterM za pomoci QVT, s cieľom rozšíriť už existujúci model. O generáciu kódu sa stará model-to-text (M2T) transformácia za využitia jazyka Xpand[10]. Pre potreby transformácie medzi jednotlivými modelmi boli v rámci práce vytvorené transformačné pravidlá, ktoré pozostávali z približne 6000 riadkov kódu.

Unifying Modeling and Programming with ALF

V práci *Unifying Modeling and Programming with ALF* [11], nie sú využité štandardné nástroje pre prácu s jazykom ALF, namiesto toho boli implementované vlastné nástroje. Ako je popísané v práci, pre vývoj v jazyku ALF je možné si vytvoriť vlastný textový editor za pomoci Eclipse frameworku Xtext[11], ktorý primárne slúži na vývoj programovacích a doménovo špecifických jazykov. Umožňuje vybudovať infraštruktúru jazyka priamo nad prostredím Eclipse, vrátane linkovania, parsovania a validácie kódu, zvýrazňovaniu syntaxe a iných záležitostí. Ďalej umožňuje modifikovať štandardné komponenty prostredia za pomoci jazyka Xtend[12]. Pre potreby jazyka ALF je nutné si naimplementovať vlastné validačné pravidlá na zobrazovanie zmysluplných chybových hlášok a upraviť gramatické pravidlá v Xtexte, vzhľadom na to, že ALF používa ľavo-rekurzívne pravidlá a parsovací generátor Xtexu ich štandardne nepodporuje. Taktiež je nutné upraviť pravidlá na určovanie rozsahu platnosti (angl. scope) na potreby importu ALF elementov a naimplementovať si vlastný typový systém [11]. Čo sa týka generácie kódu, kód z jazyka ALF nie je generovaný priamo, ale za pomoci transformácií modelov. Myšlienka transformácií je, že sa ALF model za pomoci transformačných pravidiel prevedie do iného modelu, konkretné do MoDisco Java modelu, z ktorého sa následne vygeneruje kód za pomoci už existujúcich generátorov. Transformácia medzi ALF modelom a MoDisco Java modelom je vykonaná za pomoci jazyka Atlas Transformation Language (ATL)[13], v ktorom je nutné si zadefinovat vlastné pravidlá. V práci sa udáva, že tvorba transformačných pravidel bola zdĺhavá a problémová najmä preto, že abstrakcia jazyka ALF je podstatne vyššia ako v Jave a vznikali problémy

[10] https://www.eclipse.org/modeling/m2t/?project=xpand
[11] https://www.eclipse.org/Xtext/
[12] https://www.eclipse.org/xtend/
[13] https://www.eclipse.org/atl/

s modifikátormi prístupu, navigáciou v kóde a iné. Celkovo implementácia pravidiel v ATL pozostávala z viac ako 9000 riadkov kódu [11].

On the automated translational execution of the action language for foundational UML

V práci *On the automated translational execution of the action language for foundational UML* [13] zvolili iný prístup k tomuto problému. Neboli vytvorené žiadne pravidlá na transformáciu ALF modelu do iného modelu, ale bol vytvorený nástroj, ktorý generuje C++ kód z ALF modelu. Jedná sa o prvé riešenie, ktoré automaticky generuje kód priamo z jazyku ALF. Ich riešenie umožňuje preklad konceptov správania ALF-u vrámci syntaktickej minimimálnej zhody, čiže poskytuje podmožinu jazyka ALF, ktorá je použiteľná na popisovanie správania v rámci UML modelu, to zahŕňa len možnosti dostupné v procedurálnom programovaní [25]. Taktiež umožňuje preklad jednotiek jazyka ALF, slúžiacich na popis štruktúry modelu (menný priestor, balíčky, triedy, operácie a vlastnosti). Ďalej poskytuje systém na správu pamäti založený na princípe inteligentných ukazateľov, mechanizmus na dedukciu typov a mechanizmus na určovanie rozsahu platnosti (scope). Proces transformácie môže prebiehať v dvoch scenároch.

- **Scenár 1** - Štruktúra a správanie sú zapísané za pomoci jazyka ALF. Čo sa týka štruktúry, je priamo genererovaný kód z ALF jednotiek a za pomoci dedukčného mechanizmu sú odvodzované typy. Pre generáciu správania sa môžu vykonať tri akcie. V prípade správania popísaného v ALFe generátor spustí transformáciu modelu na text, ktorej výsledkom je C++ kód. V prípade, že správanie je zadefinované v C++, tento blok sa len prekopíruje do výsledného súboru. V prípade správania popísaného iným jazykom generátor nevykoná žiadnu akciu.
- **Scenár 2** - Štruktúra je definovaná UML elementami, správanie je zadefinované za pomoci jazyka ALF. Preklad štruktúry definovej v UML prebieha za pomoci transformácie, ktorá v práci nie je bližšie špecifikovaná. Pre preklad správania generátor spustí transformáciu modelu na text, tak ako v scenári 1.

Transformácia ALF syntaxe prebieha za pomoci mapovania ALF konceptov na C++. V práci boli vytvorené mapovacie tabuľky na základe, ktorých prebieha preklad. Tieto mapovania sú v tvare `<ALF kód, C++ kód>` a zahŕňajú dvojice kvalifikovaných názvov, rôznych výrazov, deklarácií mien, podmienok, cyklov, definícií tried, operácií, indexovania a ďalších. Funkcionalita tohoto generátora bola otestovaná na systéme robota pozostávajúceho z dvoch tried zapísaných v jazyku ALF [13].

4.4.4 Implementácia

Čo sa týka implementácie konferenčného recenzného systému, nie je možné zachytiť komplexitu klientskej Angular aplikácie za pomoci jazyka ALF. Nebola nájdená ani žiadna práca, ktorá by sa venovala tvorbe webových systémov, ani nástroje, ktoré by to umožnovali. Čo je možné, je vytvoriť kostru serverovej časti v prostredí Papyrus, bez značnej funkcionality. Serverová časť primárne pozostáva z funcionality týkajúcej sa HTTP komunikácie a z operácií týkajúcich sa manipulácie dát uložených v databáze. Na tieto úkony je potrebné mať knižnice, ktoré takéto operácie umožňujú, no žiadne také neboli nájdené. Po neúspešnom pátraní bol vznesený ďalší dotaz organizácii OMG, ktorá potvrdila, že neexistujú knižnice,

ktoré by umožňovali vytvoriť pripojenie k databáze a HTTP komunikáciu. Na začiatku vývoja jazyka ALF existovali plány na návrh verzie ALF-u na štýl Ruby on Rails[14], ktorá by umožňovala vývoj servisovo-orientovaných architektúr (SOA)[15], webových systémov, no z dôvodu iných biznis priorít sa od toho časom upustilo. Jediná možnosť by bola, si potrebné knižnice naimplementovať vlastnoručne. Ďalším problémom je, že neexistujú voľne dostupné nástroje na tvorbu kódu, ktoré by umožňovali generovať kód z jazyka ALF, čiže aj v prípade ak by sa dal v ALF-e popísať celý systém, bolo by nutné si implementovať buď vlastné transformačné pravidlá na prevod ALF modelu do modelu X alebo navrhnúť vlastný generátor kódu ako bolo popísané v prácach [14], [11] a [13].

Ďalšou komunikáciou s OMG bolo zistené, že existuje jazyk Interaction Flow Modeling Language (IFML)[16] slúžiaci na modelovanie webovo založených užívateľských rozhraní. Ďalej z komunikácie vyplynulo, že sa OMG podarilo demonštrovať napojenie IFML frontendu na backend biznis logiku implementovanú za pomoci nimi bližšie nešpecifikovaného xUML v simulovanom prostredí, no zatiaľ možnosti týchto technológií sú predmetom diskusií a aktívneho výskumu. Aktuálne neexistuje riešenie ako generovať webové systémy priamo z ALF kódu, prípadne UML modelov.

Postupným testovaním technológií, študovaním materiálov, komunikáciou v rôznych diskusných skupinách, komunikáciou priamo s organizáciou OMG a technickou podporou MagicDraw, bolo zistené, že je nemožné vytvoriť a vygenerovať webové systémy za pomoci jazyku ALF. Existujú síce štúdie, kde sa podarilo vygenerovať z jazyka ALF funkčný kód, ale aby to bolo možné, je nutné si naimplementovať vlastné nástroje prípadne transformačné pravidlá, ktoré by to umožňovali. Vzhľadom na to, že cieľom tejto práce je porovnať prístupy k vývoju softwaru, vlastné nástroje neboli implementované. Implementácia takýchto nástrojov z dôvodu ich komplexnosti by mohla byť témou samostatného zadania. Okrem toho aktuálne využitie jazyka ALF je iné ako sa pôvodne, pri jeho vývoji zamýšľalo. To môže byť jeden z dôvodov, prečo nie sú k nemu dostupné použiteľné nástroje.

4.4.5 Časová os komunikácie

V tejto podsekcii je popísaná časová os komunikácie s OMG a NoMagic. Ide o stručné zhrnutie, celé znenie komunikácie je dostupné v prílohe A. Okrem komunikácie s OMG a NoMagic bola vynaložená snaha o komunikáciu v rôznych programátorských skupinách na sociálnej sieti Facebook. Komunikácia bola bez relevantného výsledku.

Prvá komunikácia s OMG:

- **Utorok, 10. marec 2020 12:59 - dotaz pre OMG**
  - bol vznesený prvý dotaz ohľadne použitia jazyka ALF na tvorbu webových systémov
- **Streda, 11. marec 2020 16:22 - odpoveď OMG**
  - jazyk ALF je možné využiť ako súčasť celkovej architektúry webových systémov
  - v simulovaných podmienkach sa podarilo demonštrovať napojenie IFML frontendu na xUML backend

[14] https://rubyonrails.org/
[15] https://www.javaworld.com/article/2071889/what-is-service-oriented-architecture.html
[16] https://www.omg.org/spec/IFML/About-IFML/

  - tieto možnosti sú stále predmetom diskusií a výzkumu
  - spoločnosť OMG si nie je vedomá o spôsobilosti generovania webových aplikácií z ALF/UML modelov

- **Štvrtok, 12. marec 2020, 15:49 - dotaz pre OMG (bez odpovede)**
  - bol popísaný problém, ktorý je riešený v rámci tejto práce
  - bol vznesený dotaz o použiteľnosti jazyka ALF na konkrétne riešenie

Druhá komunikácia s OMG:

- **Utorok, 19. máj 2020, 09:14 - dotaz pre OMG**
  - bol vznesený ďalší dotaz ohľadne poskytnutia materiálov pre použitie jazyka ALF a pripravenia prostredia

- **Streda, 20. máj 2020, 23:47 - odpoveď OMG**
  - boli doporučené už testované nástroje MagicDraw a Papyrus poskytnuté materiály k nim
  - boli poskytnuté materiály k týmto nástrojom
  - bola spomenutá referenčná implementácia jazyka ALF a ALF diskusná skupina

- **Štvrtok, 21. máj 2020, 08:34 - dotaz pre OMG**
  - bola popísaná už vykonaná práca s nástrojmi a spomenuté problémy s generáciou kódu
  - bolo popísané použitie jazyka ALF v práci na vytvorenie HTTP serveru
  - bol vznesený dotaz ohľadne generácie kódu

- **Piatok, 22. máj 2020, 20:58 - odpoveď OMG**
  - jazyk ALF bol pôvodne zamýšľaný na reprezentovanie správania v UML modeloch, no časom sa od toho upustilo
  - aktuálne sa ALF používa ako akčný jazyk v kontexte SysML modelových simulácií a to je aj jeho použitie v prostredí MagicDraw
  - v poslednej dobe je akademický záujem o generáciu kódu z jazyku ALF
  - neevidujú práce týkajúce sa webových systémov
  - neexistujú knižnice, ktoré by umožňovali HTTP komunikáciu a pripojenie ALF kódu na databázu

- **Streda, 27. máj 2020, 22:38 - dotaz pre OMG (bez odpovede)**
  - bol vznesený dotaz ohľadne referenčnej implementácie jazyka ALF a nástrojoch potrebných na jej použitie
  - bol vznesený dotaz na generáciu kódu vo viacerých krokoch, ALF kód $\rightarrow$ medzi kód $\rightarrow$ cieľový kód

Komunikácia s NoMagic:

- **Streda, 20. máj 2020, 18:18 - dotaz pre NoMagic**
  - bol založený ticket na technickú podporu NoMagic
  - bol vznesený dotaz ohľadne generácie kódu z jazyka ALF v prostredí MagicDraw
- **Streda, 27. máj 2020, 11:28 - odpoveď NoMagic**
  - MagicDraw nepodporuje generáciu do ALF kódu
- **Štvrtok, 28. máj 2020, 13:05 - dotaz pre NoMagic**
  - z dôvodu nepochopenia odpovede, bol vznesený dotaz ako je to myslené
- **Štvrtok, 28. máj 2020, 13:16 - odpoveď NoMagic**
  - MagicDraw nepodporuje generáciu ALF kódu z UML modelov
- **Štvrtok, 29. máj 2020, 02:05 - odpoveď NoMagic**
  - MagicDraw zatiaľ nepodporuje generáciu kódu z jazyka ALF
- **Štvrtok, 29. máj 2020, 13:26**
  - uzavretie ticketu

Kapitola 5

Porovnanie prístupov k vývoju softwaru

Jedným z bodov tejto práce je vytvoriť rôzne kritériá, na základe ktorých je možné porovnať dva spôsoby vývoja softwaru a pomocou daných kritérií tieto dva prístupy vyhodnotiť. V tejto kapitole sú jednotlivé kritériá popísane a prístupy sú na ich základe vyhodnotené.

5.1 Kritériá vyhodnocovania

V rámci tejto práce bolo zvolených niekoľko kritérií, ktoré budú v nasledujúcej sekcii popísane. Vzhľadom na to, že možnosti jazyka ALF sú značne obmedzené a nebolo možné za jeho pomoci systém implementovať, tieto kritériá sa vzťahujú len na možnosti daných prístupov.

Prístup k riešeniu

Cieľom tohoto kritéria je porovnať možnosti vývoja softwaru za pomoci zvolených prístupov. To zahŕňa použitie rôznych techník k vývoju, porovnanie na aký typ softwaru je možné daný prístup využiť a porovnať možnosti vývoja softwaru v rôznych fázach.

Miera zložitosti vývoja

Miera zložitosti porovnáva, aké zložité je vyvinúť software. Toto kritérium sa vyhodnocuje na základe času stráveného nad danými riešeniami, rozsahu programovania, prípadne modelovania, možnosti využitia rôznych externých frameworkov a knižníc.

Náročnosť zavedenia úpravy systemu

Toto kritérium popopisuje, aké náročné je zaviesť úpravy do už existujúceho softwaru. Kritérium počíta s modifikáciou a pridaním funkcionality v rôznych rozsahoch.

Testovanie

Cieľom tohoto kritéria je porovnať možnosti týkajúce sa testovania softwaru, od ktorej fázy vývoja je možné testovať, aké typy testov sa dajú použiť a rozsah testovania.

Kompatibilita s inými technológiami

Za pomoci tohoto kritéria budeme porovnávať, či je možné zvolené prístupy použiť v kombinácii s inými technológiami, a do akej miery je to možné.

Portabilita

Cieľom portability je porovnať, či je možné používať software vytvorený spomenutými prístupmi na rôznych platformách, ako operačné systémy Windows a Linux.

5.2 Porovnanie kritérií

Prístup k riešeniu

V prípade tradičného programovania je možné použiť hociktorý z modelov životného cyklu softwaru spomenutých v sekcii 2.1. Systém je možné najskôr navrhnúť a podľa návrhu implementovať. Tiež je možné návrh vynechať a rovno začať s programovaním a systém vymýšlať za behu, čo sa ale neodporúča z dôvodou, že predčasným návrhom môžno predísť množstvu problémov pri implementácii. Taktiež je možné implementovať software za pomoci rôznych prístupov k programovaniu, ako napríklad objektovo-orientovaný prístup alebo štrukturálny prístup. V závislosti od programovacieho jazyka je možné využiť ho na vývoj rôznych typov softwaru, ako napríklad vstavané systémy, konzolové aplikácie prípadne rôzne typy systémov a iné. Čo sa týka konkrétne jazyka Typescript (Javascript), za pomoci rôznych frameworkov je ním možné vyvíjať napríklad serverové aplikácie, ďalej interaktívne weby alebo mobilné aplikácie.

Jazyk ALF je možné použiť na simulácie modelov v kontexte SysML čo je jeho primárne využitie. Je ním možné implementovať aj jednoduché programy za pomoci dvoch prístupov, a to buď samostatne na základe referenčnej implementácie ALF-u alebo v kombinácii ALF-u s UML. Tento jazyk ale nie je vhodný na implementáciu komplexných systémov z dôvodu jeho obmedzených možností a chýbajúcich nástrojov.

Miera zložitosti vývoja

Vyvinúť systém programovaním sa javí ako podstatne ľahšie a časovo menej náročné vzhľadom na obmedzené možnosti jazyka ALF. Typescript, teda Javascript je pomerne rozšírený jazyk a existuje k nemu množstvo knižníc a frameworkov. Tieto rozšírenia uľahčujú prácu s ním a pridávajú ďalšiu funkcionalitu, čiže nie je nutné vyvíjať systém od podlahy. Tým, že je rozšírenejší je k nemu aj viac literatúry a návodov. V prípade vzniku problémov počas vývoja sa tieto problémy ľahšie odhaľujú a je jednoduhšie ich odstrániť. Taktiež existuje množstvo prostredí, v ktorých je možné aplikácie vyvíjať. Narozdiel jazyk AFL je pomerne neohrabaný, základné knižnice ponúkajú obmedzené možnosti a neexistujú knižnice tretích strán, ktoré by uľahčovali vývoj. Vzhladom na to, že ALF nie je moc rozšírený, jediná spoľahlivá príručka je oficiálny štandard ALF-u a akademické štúdie, ktoré sa tomutu jazyku venovali. Taktiež existuje len obmedzené množstvo prostredí, v ktorom sa dá za pomoci jazyka ALF vyvíjať. Tieto prostredia sú samostatne nedostačujúce a preto k nim treba doinštalovať rôzne rozšírenia. Spojazdnenie týchto prostredí je podstatne komplikovanejšie ako pri programovaní. Vzhľadom na obmedzenia, jazyk ALF nie je možné použiť na vytvorenie celého sytému, ale iba jeho časti. Neexistujú knižnice, ktoré by umožňovali autorizáciu pomocou JWT, vytvorenie databázového spojenia alebo komunikáciu cez REST API. Ak

sa pozeráme na vývoj z časového hľadiska, štúdium na potreby implementácie a samostatná implementácia celého systému programovaním zabrala zhruba mesiac práce, zatiaľ čo v prípade ALF-u skúmanie jeho možností, štúdium problematiky, hľadanie relevantných zdrojov, čítanie rôznych štúdií, testovanie viacerých technológií a prostredí zabralo viac než tri mesiace.

Testovanie

Systém implementovaný za pomoci Typescriptu je možné testovať už od prvotných fáz vývoja. Je možné napísať rôzne testy ešte predtým, než je samostatný software prípadne jeho časti hotové, tak ako sa to využíva v prístupe k vývoju software zvaným Test Driven Development[1]. Je možné vytvoriť si rôzne typy testov ako napríklad jednotkové, integračné prípadne End-To-End. Na automatické testovanie existuje množstvo knižníc ako napríklad Chai[2] alebo Mocha[3]. Ďalej je možné aj manuálne otestovať API pomocou aplikácie Postman[4], pomocou ktorej môžeme posielať na server API dotazy.

V prípade jazyka ALF je možné vytvárať testy, ktoré sú implementované pomocou aktivít. Na testy je nutné si vytvoriť vlastnú infraštruktúru z dôvodu, že neexistujú knižnice, ktoré by to umožňovali. Porovnávanie hodnôt v testoch je možné klasickými operátormi alebo operáciami ako napríklad `exists`, `isUnique`, `one` a inými.

Náročnosť zavedenia úpravy systému

Náročnosť zavedenia úpravy v programovaní v prvom rade závisí od zvolenej architektúry. Ak je architektúra vytvorená správne, nemali by sa vyskytovať väčšie komplikácie pri zavedení zmien. Vzhľadom na to, že architekrúra systému je robená tak, aby ju bolo možné rozšíriť, nie je problém pridať ďalšie entity, kontroléry, sevirsy pre potrebu rozšírenia funkcionality, prípadne urobiť zmenu už v existujúcej funkcionalite.

Čo sa týka jazyka ALF, je možné pridávať novú funkcionalitu, bez väčších problémov a rovnako upravovať už existujúcu. Napríklad pridať novú metódu do triedneho diagramu a implementovať ju ako aktivitu v ALF-e. Vzhľadom na to, že systém nebolo možné implementovať, nebolo testované ani väčšie zavedenie zmien.

Kompatibilita s inými technológiami

Použitie tradičného programovania má z technologického hľadiska navrch od modelovo riadeného prístupu za pomoci ALF. Konkrétne, čo sa týka Javascriptu, môže byť využitý samostatne. Taktiež je možné využiť ho v kombinácii s inými technológiami, v ktorých môže hrať menšiu alebo väčšiu rolu. Primárne sa využíva v kombinácií s CSS a HTML na tvorbu webových stránok, ale je možné využiť ho aj v kombinácii s inými jazykmi ako PHP, ASP.NET a ďalšími na tvorbu celých systémov. Prípadne je možné využiť ho ako akčný jazyk pre technológie, ako napríklad Java Business Process Management (JBPM[5]), tak ako sa využíva ALF pre UML.

Jazyk ALF je možné využiť priamo len s technológiami od OMG. V prípade dostupných generátorov a rôznych transformácií je možné využiť ALF v kombinácii s inými technoló-

[1] http://agiledata.org/essays/tdd.html
[2] https://www.chaijs.com/
[3] https://mochajs.org/
[4] https://www.postman.com/
[5] https://www.jbpm.org/

giami nepriamo. Tým je myslené, že po vytvorení cieľového modelu z jazyka ALF je možné tento model využiť v kombinácii s technológiami prislúchajúcimi výslednému modelu.

Portabilita

Na rozchodenie systému implementovaného jazykom Typescript je potrebné nainštalovať Node.js a spolu s ním sa nainštaluje aj balíčkový manažér Node Package Manager (NPM)[6], ktorý umožní dotiahnutie všetkých závislostí potrebných na rozchodenie systému. Node.js je multiplatformový, čiže nie je problém rozchodiť ho na systémoch ako Windows, Linux prípadne macOS. V prípade použitia iného programovacieho jazyka, napríklad jazyka C# s .NET[7] frameworkom, by nebolo možné server rozbehať na inom systéme ako Windows. Čo sa týka jazyka ALF, tak vzhľadom na to, že cieľom MDA je vývoj platformovo nezávislého softwaru, je možné z ALF programov vytvárať software spustiteľný na rôznych operačných systémoch v závislosti od dostupných generátorov a technológií.

[6] https://www.npmjs.com/
[7] https://dotnet.microsoft.com/

Kapitola 6

Záver

Cieľom tejto práce bolo oboznámiť sa s prístupom k tvorbe softwaru zvaným Model Driven Engineering a porovnať tento prístup s klasickým prístupom vývoja softwaru a demonštrovať ho na prípade konferenčného recenzného systému. Pre dané prístupy k návrhu vytvoriť kritériá na základe ktorých, budú systémy porovnané.

Vzhľadom na to, že nebolo možné vytvoriť systém za pomoci jazyka ALF, tieto kritériá sa vzťahovali len na prístupy k vývoju a nie na implementované systémy. Bola vytvorená špecifikácia systému popisujúca rôzne požiadavky na systém a urobený rozbor technológií, ktoré je možné použiť. Následne bol systém navrhnutý a naimplementovaný množinou zvolených technológií. Boli preskúmané možnosti MDA za pomoci jazyka ALF a rôzne technológie s týmto súvisiacie. Po neúspešných pokusoch o implementáciu systému v jazyku ALF v rôznych prostrediach bola zahájená komunikácia v rôznych diskusných skupinách, mailová komunikácia s organizáciou OMG a bol iniciovaný dotaz technickej podpore spoločnosti NoMagic. Komunikáciou s jednotlivými stranami bolo zistené, že účel jazyka ALF je aktuálne iný ako sa pôvodne zamýšlalo, nie je možné ním naimplementovať komplexnejšie systémy so špecifickou funkcionalitou z dôvodu, že základné knižnice nepodporujú požadovanú funkcionalitu a knižnice tretích strán nie sú k dispozícii. Dostupné prostredia pre potreby implementácie konferenčného systému sú nedostačujúce aj s rozšíreniami a neumožňujú generovať kód z jazyka ALF. Neboli nájdené verejne dostupné generátory kódu, ktoré by sa dali v testovaných prostrediach využiť. Pre potreby vytvorenia spustiteľného kódu je nutné navrhnúť a naimplementovať vlastné transformačné pravidlá, ktoré by pretransformovali ALF model do iného modelu, pre ktorý už existuje generátor kódu alebo si generátor implementovať vlastnoručne, čo je v oboch prípadoch mimo rozsah zadania práce. Aktuálne je možné jazyk ALF využiť v prostredí MagicDraw pre popis správania UML modelov alebo na popis simulácií v SysML, čo je aj jeho primárny účel. V prostredí Eclipse je možné vytvárať jednoduchší software, ktorý je možné so sadou voľne dostupných rozšírení spúšťať, ladiť a za pomoci aktivít testovať jeho správnu funckionalitu.

K jazyku ALF existujú rôzne alternatívy, ktoré ponúkajú taktiež modelovo riadený prístup k vývoju softwaru. Napríklad modelovací nástroj a programovací jazyk Umple[1], slúžiaci na modelovo-orientované programovanie, umožňujúci generovanie kódu. Prípadne Xcore[2], rozširujúci konkrétnu syntax Ecore[3] modelov v Eclipse Modeling Framework, kde by malo byť možné vygenerovať Java kód z týchto modelov [11].

[1] https://cruise.eecs.uottawa.ca/umple/
[2] https://wiki.eclipse.org/Xcore/
[3] https://wiki.eclipse.org/Ecore/

Myšlienka modelovo riadených prístupov ponúka zaujímavý pohľad na problematiku týkajúcu sa vývoja softwaru, no v aktuálnej dobe sú technológie týkajúce sa tohoto prístupu stále vo fáze vývoja a nie je možné ich plnohodnotne využiť pre potreby vývoja komplexnejších systémov.

Literatúra

[1] Angular CLI. [online; navštívené 6.5.2020].
URL https://cli.angular.io/

[2] Angular Docs. [online; navštívené 6.5.2020].
URL https://angular.io/docs

[3] Domain-Specific Languages. [online; navštívené 6.5.2020].
URL https://www.jetbrains.com/mps/concepts/domain-specific-languages/

[4] GENERAL PURPOSE PROGRAMMING LANGUAGE. [online; navštívené 6.5.2020].
URL https://www.javatpoint.com/general-purpose-programming-language

[5] Model-Based Engineering Visual Glossary. [online; navštívené 2.3.2020].
URL https://modelbasedengineering.com/glossary//

[6] SDLC Tutorial. [online; navštívené 2.3.2020].
URL https://www.tutorialspoint.com/sdlc/index.htm

[7] Software Maintenance Overview. [online; navštívené 2.3.2020].
URL https://www.tutorialspoint.com/software_engineering/software_maintenance_overview.htm

[8] UML - Building Blocks. [online; navštívené 6.5.2020].
URL https://www.tutorialspoint.com/uml/uml_building_blocks.htm

[9] UML - Standard Diagrams. [online; navštívené 6.5.2020].
URL https://www.tutorialspoint.com/uml/uml_standard_diagrams.htm

[10] Booch, G.; Rumbaugh, J.; Jacobson, I.: *Unified Modeling Language User Guide, (2nd Edition)*. Addison Wesley, 1999, ISBN 0-201-57168-4.

[11] Buchmann, T.; Rimer, A.: *Unifying Modeling and Programming with ALF In: SOFTENG 2016 : The Second International Conference on Advances and Trends in Software Engineering*. 2016, ISBN 978-1-61208-458-9.

[12] Bézivin, J.: *Model Driven Engineering: An Emerging Technical Space, In: Lämmel R., Saraiva J., Visser J. (eds) Generative and Transformational Techniques in Software Engineering. GTTSE 2005. Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 2006, ISBN 978-3-540-46235-4.

[13] Ciccozzi, F.: *On the automated translational execution of the action language for foundational UML. Software and systems modeling.* 2018.
URL https://doi.org/10.1007/s10270-016-0556-7

[14] Ciccozzi, F.; Cicchetti, A.; Sjödin, M.: *On the Generation of Full-fledged Code from UML Profiles and ALF for Complex Systems In: 12th International Conference on Information Technology - New Generations).* 2015, ISBN 9781479988297.

[15] Coleman, A.: Playing with Legos: The Inner-Workings of a Model-View-Controller (MVC) Web Application. [online; navštívené 2.3.2020].
URL
https://selftaughtcoders.com/model-view-controller-mvc-web-application/

[16] Guermazi, S.; Tatibouet, J.; Cuccuru, A.; aj.: Executable Modeling with fUML and Alf in Papyrus: Tooling and Experiments. [online; navštívené 10.5.2020].
URL http://ceur-ws.org/Vol-1560/paper1.pdf

[17] Křena, B.; Kočí, R.: *Úvod do softwarového inženýrství.* FIT VUT v Brně, 2010.

[18] Mayerhofer, T.; Langer, P.; Wimmer, M.; aj.: *xMOF: Executable DSMLs Based on fUML In: Software Language Engineering 6th International Conference, SLE 2013.* 2019.

[19] Mellor, S. J.; Balcer, M. J.: *Executable UML: A Foundation for Model-Driven Architecture.* 2002, ISBN 978-0201748048.

[20] Micskei, Z.; Konnerth, R.-A.; Horváth, B.; aj.: On Open Source Tools for Behavioral Modeling and Analysis with fUML and Alf. [online; navštívené 10.5.2020].
URL http://ceur-ws.org/Vol-1512/paper3.pdf

[21] Nejati, S. S.; Maleki, M.: *Report on How to Use ALF Action Language and fUML execution/debugging with Moka.* [online; navštívené 6.5.2020].
URL https://wiki.eclipse.org/images/5/5a/ALF_fUML_Moka.pdf

[22] OMG: *Object Constraint Language, Version 2.4.* 2014, [online; navštívené 6.5.2020].
URL https://www.omg.org/spec/OCL/2.4/PDF

[23] OMG: *OMG Meta Object Facility (MOF) Core Specification, Version 2.5.* 2015, [online; navštívené 6.5.2020].
URL https://www.omg.org/spec/MOF/2.5/PDF

[24] OMG: *XML Metadata Interchange (XMI) Specification, Version 2.5.1.* 2015, [online; navštívené 6.5.2020].
URL https://www.omg.org/spec/XMI/2.5.1/PDF

[25] OMG: *Action Language for Foundational UML, Version 1.1.* 2017, [online; navštívené 6.5.2020].
URL https://www.omg.org/spec/ALF/1.1/PDF

[26] OMG: *Semantics of a Foundational Subset for Executable UML Models (fUML), Version 1.4.* 2018, [online; navštívené 6.5.2020].
URL https://www.omg.org/spec/FUML/1.4/PDF

[27] Packer, D.: Verification vs Validation: Do You know the Difference? [online; navštívené 2.3.2020].
URL https://www.plutora.com/blog/verification-vs-validation

[28] Seidewitz, E.; Tatibouet, J.: Tool Paper: Combining Alf and UML in Modeling Tools – An Example with Papyrus –. [online; navštívené 10.5.2020].
URL http://ceur-ws.org/Vol-1512/paper09.pdf

[29] Sommerville, I.: *Software Engineering, Tenth Edition*. Pearson, 2016, ISBN 978-1-292-09613-1.

[30] Stachowiak, H.: *Allgemeine Modelltheorie [General Model Theory]*. Springer, 1973, ISBN 3-211-81106-0.

Príloha A

Obsah priloženého pamäťového média

- Zdrojové kódy systému
- UML modely
- E-mailová komunikácia s OMG
- Ticket z technickej podpory NoMagic